UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA

FRAMEWORK PARA DESARROLLO DE SITIOS DE COMERCIO ELECTRÓNICO EN JAVA

Memoria presentada por: **Javier Alejandro Villalobos Arancibia**
Como requisito parcial para optar al título de **Ingeniero Civil Electrónico Mención Computadores y Sistemas Digitales**

Profesor Guía: **Agustín González V.**

Abril, 2004

A mi padre **Hérman Villalobos Rojas**,
que con sus propios logros y sacrificios me dio una oportunidad que no muchos tienen...

A mi madre **Ana Arancibia Henriquez**,
que con su esfuerzo me mantuvo con vida y siempre me dio protección...

A mis abuelos
Ambrosio Villalobos, Alberto Arancibia, Luisa Rojas y Marina Henriquez,
de quienes heredé a estas dos maravillosas personas...

Indice

Indice de figuras

Indice de tablas

Resumen

El presente trabajo hace un estudio, diseño e implementación de un framework para sitios de comercio electrónico en JAVA. Comenzando con una breve descripción de las etapas de desarrollo y seguido de una revisión conceptual de los elementos que componen un sitio de comercio electrónico.

Posteriormente se incluyen diagramas sobre los procesos que acontecen, breve descripciones sobre las interacciones entre el framework y los usuarios de éste, diagramas conceptuales de los elementos a desarrollar, criterios de decisión sobre la tecnología a utilizar y diagramas de clase para la construcción final del código en Java.

Finalmente se expone brevemente el desarrollo empírico para probar el funcionamiento del framework, conclusiones y resultados.

En forma anexa se incorpora la documentación de las clases utilizadas.

Introducción

Un framework puede definirse como la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas.

Un sitio de comercio electrónico es un entorno de acceso remoto, principalmente vía Internet, que permite el intercambio de bienes y servicios entre una empresa con sus proveedores, y/o sus clientes.

Hoy en día, frente a la apertura masiva de mercados internacionales, además del gran auge que experimentó la Internet hace años atrás y su actual proceso de mejoramiento, los sitios de comercio electrónicos surgen como una poderosa herramienta de marketing para que las empresas lleguen a sus clientes y por sobre todo, potenciales clientes lleguen a ellas. En base a esto, numerosas empresas desarrollan productos para la elaboración de aplicaciones que apuntan a ese objetivo.

A medida que las necesidades de recursos para estas aplicaciones se van haciendo globales, también el concepto de estandarización se hace necesario, pues ante la complejidad y diversidad de sistemas, emplear diversas tecnologías involucra invertir en interfaces que relacionen a estas tecnologías entre sí.

Dentro de este contexto, se expone a continuación un estudio, diseño e implementación de una solución cuyo objetivo es generar un framework en JAVA, que implemente funciones básicas de un sitio de comercio electrónico y permita el desarrollo de funcionalidades adicionales.

La metodología empleada para el desarrollo de este trabajo contempla utilizar Modelamiento de Lenguaje Unificado (UML) para el estudio de conceptos, diseño e implementación de código JAVA.

La tecnología seleccionada está basada en aplicaciones Cliente-Servidor Java Servlets y Java Server Pages.

Capítulo 1
Las etapas de desarrollo

Toda materialización de una idea, crear un producto o bien, implementar un servicio o desarrollar una aplicación que interactúa con otros sistemas requiere una metodología. Una metodología permite planificar un desarrollo, evaluar el avance y poder identificar factores importantes para tomar la decisión de dar el siguiente paso.

El Unified Process (UP) [JBR99] o Proceso Unificado describe diversas etapas en el desarrollo de una aplicación asignándoles un ciclo de vida y permitiendo el manejo de riesgos[1]. La idea más importante es el desarrollo iterativo [CL2002] el cual organiza el desarrollo en una serie de cortos mini-proyectos llamados iteraciones. Cada iteración tiene un ciclo de vida, del cual se obtendrá un mini-sistema probado e integrado. Este tipo de metodología es conocida como Desarrollo Iterativo e Incremental

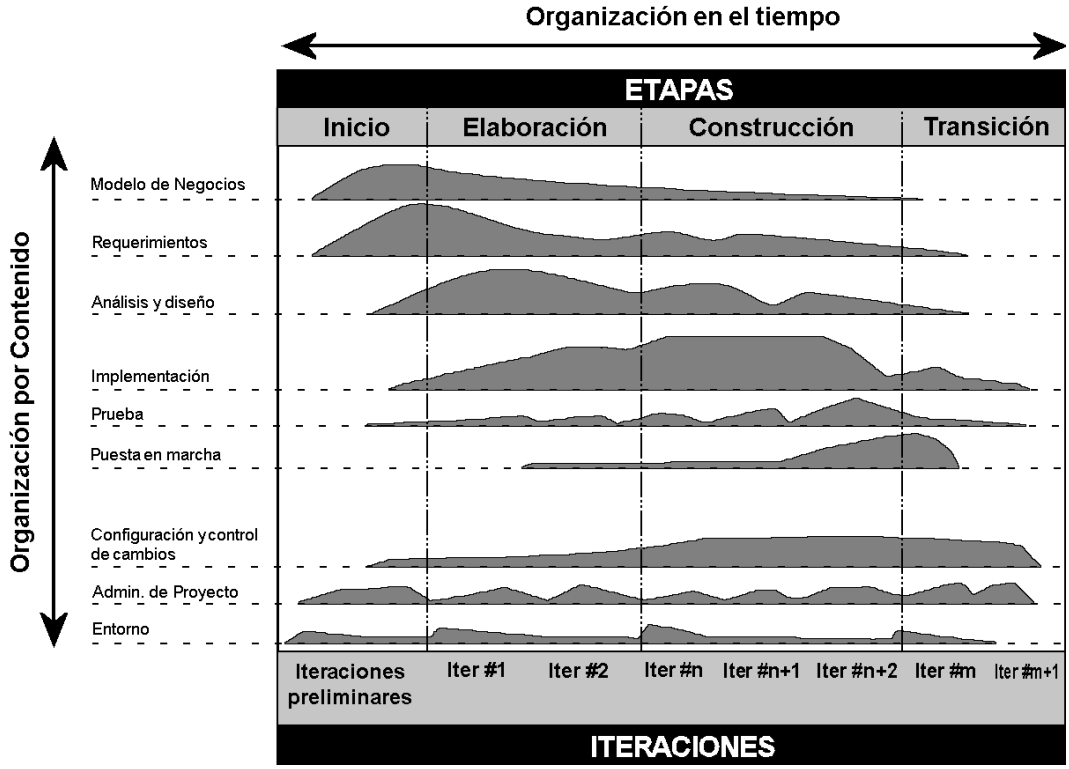

Figura 1.1: Desarrollo Iterativo e Incremental

[1]El concepto riesgo es utilizado para agrupar aquellos factores importantes que a largo plazo pueden influir en el desempeño de la aplicación, por ejemplo, el manejo de concurrencia, la cantidad de usuarios a utilizar la aplicación, etc.

La figura 1.1 muestra una forma de organizar el desarrollo de la aplicación desde un punto de vista de los procesos[2], donde un *Modelo de negocios* explica en forma generalizada elementos y procesos que participan en el entorno de la aplicación. Un segundo punto son los diversos *Requerimientos* que, esencialmente, corresponden al QUÉ de la aplicación o el fin para el cual se desarrolla. Junto con ello se encuentra el *Análisis y diseño* que decide la tecnología a utilizar y traslada al plano abstracto las ideas recopiladas en los primeras dos etapas. Ya en la etapa de *Implementación* la abstracción cobra cuerpo al desarrollarse código o sistemas que previamente se diseñaron. Finalmente en la etapa de *Prueba* se evalúa el funcionamiento de lo implementado y se van acoplando los distintos módulos desarrollados. Cuando un módulo o conjunto de módulos ha superado esta etapa pasan a la etapa de *Puesta en marcha* que simplemente significa contrastar la aplicación con el mundo real.

Las tres etapas siguientes son procesos empresa cliente, donde se hacen consideraciones, adaptaciones y traspaso de diversa información para la óptima operación de la aplicación. Estas etapas no serán tema de discusión en este documento.

Un aspecto interesante es considerar que las etapas pueden ocurrir simultáneamente, debido a la flexibilidad que se requiere ante posibles cambios. Otro aspecto tomado en cuenta para organizar el desarrollo es el tiempo, en este caso el tiempo se divide en cuatro etapas, las cuales relacionan el desarrollo con la madurez del proyecto e indican cualitativamente cuánto esfuerzo o dedicación de cada etapa organizada por contenido se requiere.

1. **Inicio**

   En esta etapa se identifican aspectos a un alto nivel del negocio y los procesos que están involucrados en una aplicación de comercio electrónico. Definición de términos, requerimientos y prototipos. Para el caso desarrollado aquí, se establecerán los elementos, actores y procesos comunes para sitios de comercio electrónicos.

2. **Elaboración**

   Esta etapa corresponde a la más importante de las cuatro etapas, pues se sientan las bases del proyecto, se mitigan considerables riesgos y se establece la arquitectura que soportará la implementación. Para el caso de un sitio de comercio electrónico estos aspectos corresponderán al análisis de la arquitectura del software como es la utilización Servlets y Java Server Pages, diagramas conceptuales y casos de uso.

[2]Un *proceso* puede ser entendido como un conjunto de pasos a seguir, iniciados por una condición de entrada o evento, con la finalidad de obtener un resultado u otro evento.

3. **Construcción**

   En esta etapa, se implementa lo elaborado en la etapa anterior y se completan detalles en los requerimientos. Adicionalmente a medida que se implementa, en esta etapa se realizan pruebas (iteraciones). Aquí se harán las implementaciones de las clases diseñadas, documentación, completar análisis y diseños en los distintos procesos y transacciones del sitio de comercio electrónico.

4. **Transición**

   En esta etapa, se concluyen las implementaciones y se integra lo desarrollado. Se realizan las pruebas finales y globales. En la aplicación de comercio electrónico se integrarán los diversos procesos y se realizarán pruebas para el cumplimiento de los casos de usos necesarios.

En [RUP1998] se establece con más detalles las etapas aquí descritas, incluyendo ejemplos.

Capítulo 2
Inicio : *Requerimientos y modelo de negocios*

Lo importante en esta etapa, es agrupar los diversos elementos y entender el contexto que envuelve a los sitios de comercio electrónico. Esto involucra conocer terminología que se empleará para definir los elementos, actores, procesos, etc., modelar el negocio en base a esquemas que identifiquen partes o sistemas. Continuando con la etapa anterior, aquí se elaboran los casos de uso posibles y realizables. Para el caso de este documento, se desarrollarán tres casos de uso, de los cuales se implementará uno.

2.1 Elementos de un sitio de comercio electrónico

El siguiente paso es establecer el tipo de comercio electrónico con el que se trabajará. Esencialmente existen dos tipos: Comercio para Empresas (Business to Business o B2B) y Comercio para Consumidores (Business to Customers o B2C). Un comercio del tipo B2B está enfocado hacia las relaciones comerciales entre un empresa y sus proveedores. Mientras que un comercio B2C se enfoca hacia un cliente consumidor de los productos, bienes o servicios que la empresa ofrece. El framework a implementar está enfocado al comercio de tipo B2C.

Así, se procede a identificar y conceptualizar los diversos elementos que pueden componer un sitio B2C. Estos pueden variar bastante, pues dependen de cómo las transacciones, procesos y negocios se manejan en las diversas empresas. Sin embargo podemos rescatar ciertas similitudes entre las diversas empresas ya abocadas a este tipo de negocios.

Catálogo : Un catálogo es un conjunto de bienes, productos o servicios agrupados bajo algún criterio. Su composición descriptiva puede ser diversa y depende de aquellos atributos que se desean visualizar (imágenes, textos, enlaces Web, etc.) y los atributos de manejo interno (códigos, subclasificaciones, etc.). Un catálogo puede a su vez poseer otros subcatálogos.

Producto : Se llamará *Producto* a los bienes o servicios finales individualizados en la empresa. Esencialmente un producto puede ser identificado mediante un código, una descripción, atributos y diferentes tipos de precios.

Artículo : Un artículo corresponderá a la visualización de un producto, bien o servicio final. Es probable que un conjunto de productos de características similares, visualmente puedan ser descritos por un sólo artículo. Por ejemplo, una zapatilla de una determinada marca y modelo denominada "zapatilla x" puede tener diferentes números y colores. Cada zapatilla de un

número y color determinado (41, color azul) sería un producto ("zapatilla x, 41, color azul") mientras que el conjunto de todos los números y colores pueden ser descritos por el artículo "zapatilla x".

Detalle de Artículo : Esta definición se incorpora como elemento adicional, pues permite simplificar una lista de artículos mostrando información general, y adicionalmente dar la opción de obtener una información más explícita o detallada de un artículo de la lista presentada. Sin embargo, su utilización será una decisión particular de la empresa.

Banner : Un banner es un logotipo de especiales características que presenta diversos tipos de información. Puede contener un aviso informativo, un enlace hacia otro servicio, una representación de un artículo o un enlace hacia una promoción.

Promociones : Este elemento cada día tiene más importancia dentro de los sitios B2C, porque es el punto de atención para atraer a los clientes. Consiste básicamente en agrupar dos o más productos (o artículos) bajo ciertas condiciones, por ejemplo, precio con descuento, producto de regalo, descuentos por cantidad, etc.

Canasta de compra : Este concepto identifica a todos aquellos productos, bienes o servicios que el cliente final ha seleccionado para comprar.

Proceso de compra : Este concepto involucra aquellos pasos necesarios de interacción entre el sistema y el cliente para establecer un proceso comercial a nivel del sitio B2C, es decir, la compra virtual.

Tipo de Pago : El tipo de pago empleado es muy importante pues es el medio que permite el intercambio del bien, producto o servicio entre el sistema (la empresa) y el cliente final. Es así que éstos pueden variar dependiendo de aquellos medios de pago aceptados por la empresa.

Información de despacho : Es importante tener en consideración la información de despacho dado que se trata de un intercambio de productos, bienes o servicios realizado a distancia. Esto involucra saber los valores por concepto de flete, los descuentos por tamaño, peso o dimensiones que son aplicados a los productos, bienes o servicios.

Registro de Cliente : Llevar un registro de los datos del cliente es de suma importancia pues individualiza al cliente final (potencial comprador), manteniéndolo informado de nuevos productos, resguardando su seguridad como comprador y, desde el punto de vista del vendedor, saber el destino de los productos comprados.

Motor de Búsqueda : Si bien, navegar a través de catálogos permite a los clientes un fácil acceso hacia productos de interés, un motor de búsqueda es esencial para agilizar la búsqueda de productos. Buscar un producto puede hacerse ingresando un tipo de producto, un rango de precios, un código de producto, etc.

Atención al Cliente : Permitirle al cliente emitir comentarios, solucionar problemas y establecer reclamos es un importante concepto de marketing. Permite tener contacto con el cliente, mejorar el sistema de acuerdo a las personas que interactúan con él, saber inquietudes, etc.

Secciones Especiales : Muchas empresas o áreas de ésta, querrán destacar ciertos aspectos o tendrán sus propios modelos de promocionar sus productos, bienes o servicios. Es así, que se puede tener una sección específica con catálogos propios, precios y procesos de compra especiales.

Estadísticas : Las estadísticas cobran mucha relevancia a la hora de evaluar la rentabilidad del negocio. Es así que importa obtener el número de visitantes, estadísticas de venta, productos más vendidos, etc.

2.2 Modelo de negocios

Esencialmente un usuario cliente (visitante) cursa tres etapas de negociación en el comercio electrónico. Una primera etapa que corresponde a la búsqueda de un producto de interés (navegación por el sitio) que será el objetivo principal que cubrirá este framwework, un proceso de compra del producto y la transacción comercial final con la empresa que posteriormente significará recibir el producto comprado. En estas tres etapas ocurren diversos procesos que pueden dividirse en Procesos Visitante-Sistema (PVS), Procesos Sistema-Sistema (PSS) y Procesos Sistema-Negocio (PSN).

En la figura 2.1 identificamos dos actores principales que son un Visitante (cliente final) y el Sistema (sitio B2C). Dentro de este esquema están incluidos los dos primeros grupos de procesos (PVS y PSS). El tercer grupo no será considerado, pues involucra a cada empresa en particular y en cómo cada una trata el negocio.

Dentro de los elementos del sistema se identifica un *Filtro de requerimientos* encargado de recibir las peticiones desde los clientes (vía Internet, en este caso), identificar aquellas peticiones relacionadas con el sitio y evitar el ingreso de información maliciosa (seguridad). Cuando el requerimiento se encuentra dentro de lo esperado, es guiado hacia un *Encausador de procesos* que se encarga de identificar qué procesos deben llevarse a cabo. Nótese que se puede ir directamente al *Administrador de vitrina* que es quien genera la información de salida hacia el cliente.

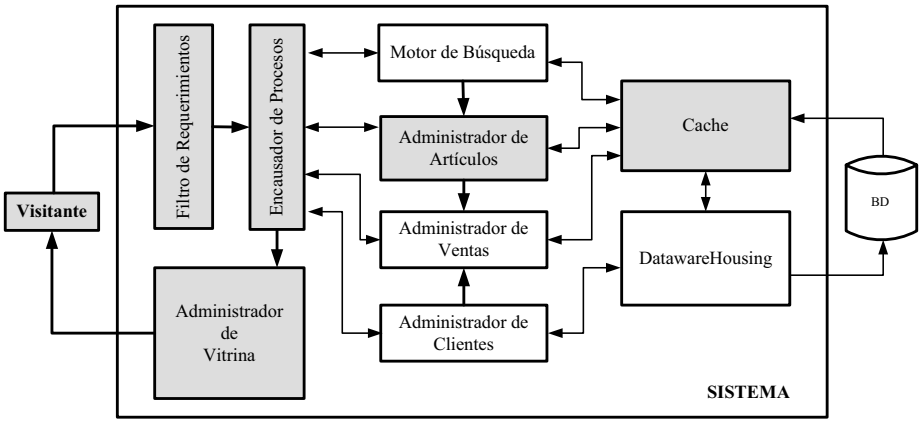

Figura 2.1: Modelo de Negocios

Los cuatro módulos siguientes cumplen funciones generalizadas que son: *Motor de búsqueda* encargado de realizar búsquedas diversas en el sistema, *Administrador de artículos* que ejecuta aquellos procesos relacionados con catálogos, artículos y productos, un *Administrador de ventas* que genera las solicitudes de compras y administra medios de pago, y finalmente un *Administrador de clientes* que administra la información cliente como son sus datos personales y direcciones de despacho.

Adicionalmente debe existir un módulo *Datawarehousing* para el manejo de información estadística y eventos en general, y un módulo *Cache* que administrará la información del sitio y permitir una navegación[1] más rápida. La versión del framework aquí desarrollada contemplará los elementos en gris en la figura 2.1

2.2.1 Procesos Visitante-Sistema

Estos procesos son interactivos entre el visitante y el sistema. Un visitante visualiza catálogos, productos y promociones hasta obtener la descripción de un producto de interés. Adicionalmente se registra en el sitio para entregar sus datos personales y direcciones de despacho. Eventualmente realiza la compra del producto y espera su recepción. El sistema por su parte, entrega la información necesaria al visitante, lo guía por el proceso de compras y guarda un registro de él.

1. **Obtención de Catálogos :** El visitante puede acceder a los productos, bienes o servicios a través de agrupaciones de éstos denominadas Catálogos. De esta forma el visitante solicita información de un Catálogo determinado, el cual es generado en un *Administrador*

[1]El término navegación se emplea hace bastante tiempo, y aquí se utilizará para expresar la idea de la interacción entre un Visitante y el Sistema

de Artículos y que obtiene la información desde un *Cache* de catálogos, luego genera los aspectos visualizables (y enlaces a subcatálogos si los posee) para el visitante (puede encontrarse en última instancia de almacenamiento, una base de datos). La figura 2.2 muestra un esquema del proceso.

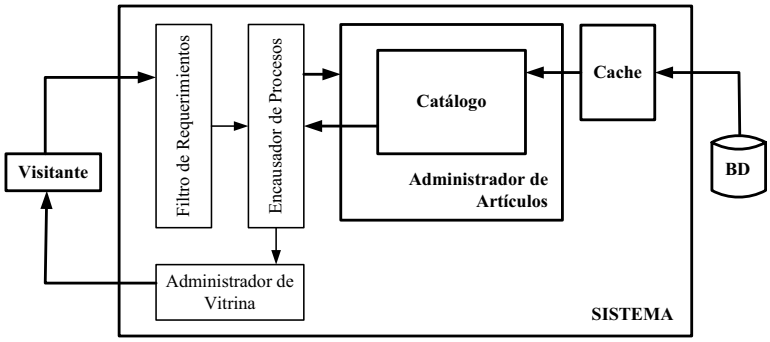

Figura 2.2: Obtención de Catálogos

2. **Obtención de Artículos :** El visitante mientras navega por los distintos catálogos y subcatálogos, finalmente obtendrá una lista de los artículos contenidos en los catálogos. Esto es obtenido a través de un *Administrador de artículos* que es referenciado por catálogo.

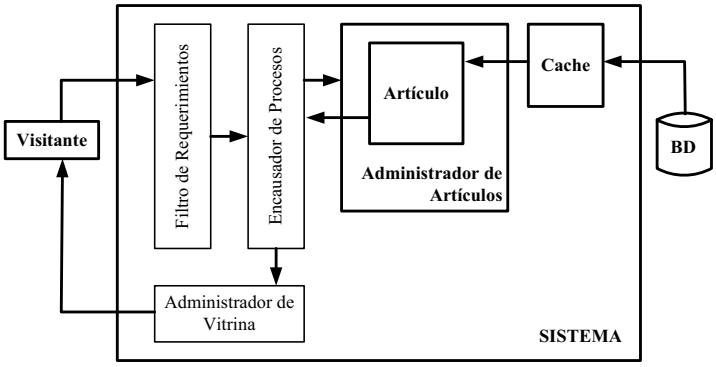

Figura 2.3: Obtención de Artículos

3. **Registro de Cliente :** La información que se puede registrar de un cliente, puede ser variada y dependerá de cada empresa, sin embargo esta información puede ser dividida en un *Registro Personal* y *Direcciones* asociadas. Es así que en una primera etapa de registro (registro voluntario) sólo será necesario obtener un registro personal (datos personales del cliente) y posteriormente, durante el proceso de compras obtener la dirección personal y direcciones de despacho que el cliente estime necesario.

4. **Búsqueda :** Un motor de búsqueda puede ser muy sofisticado y complicado de implementar, pero básicamente se puede realizar una búsqueda en base a un texto ingresado,

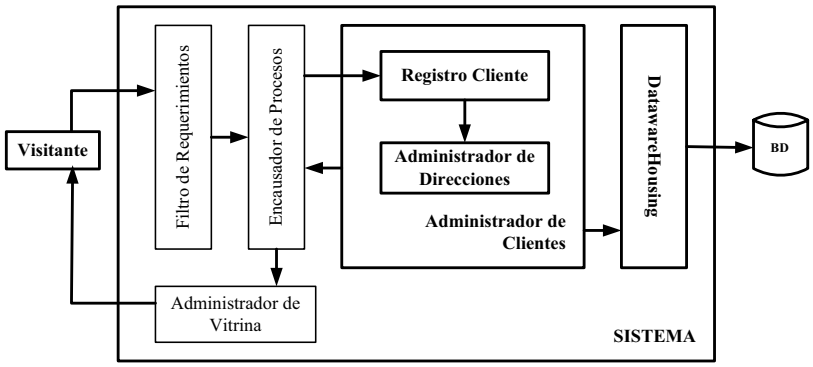

Figura 2.4: Registro de Cliente

y dado que el interés principal es orientado hacia la obtención de información de un artículo, la búsqueda se realiza sobre los atributos visualizables contenidos en el sitio. Adicionalmente, para agilizar búsquedas posteriores se crearán catálogos de búsqueda, así, en una próxima búsqueda se consultará la existencia de algún catálogo referenciado por el texto ingresado.

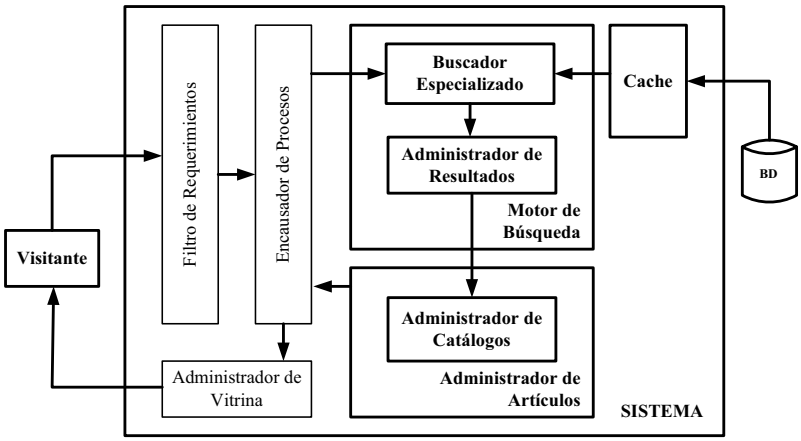

Figura 2.5: Búsqueda de Artículos

5. **Navegar el sitio :** El sitio puede contener otras informaciones, páginas estáticas o enlaces hacia otros sitios. Un *Administrador de vitrina* se encarga de presentar o servir al cliente de este tipo de contenidos.

6. **Contactar empresa :** Esencialmente este proceso involucra ingresar una información para contacto del visitante, junto con su comentario, sugerencia o reclamo que es enviada de alguna forma hacia la empresa. De esto se encarga el *Administrador de clientes*.

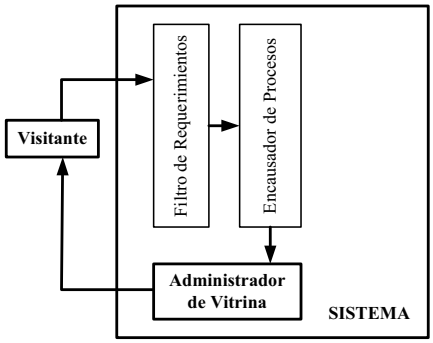

Figura 2.6: Navegar el Sitio

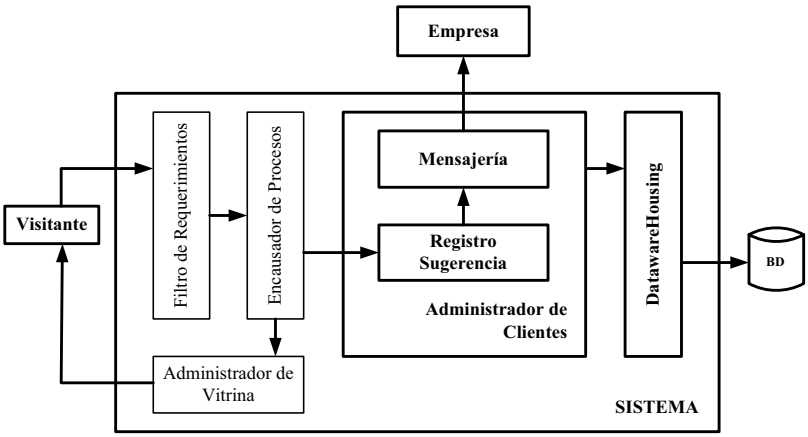

Figura 2.7: Contactar a Empresa

7. **Proceso de compras** El proceso de compras esencialmente son cinco pasos que el visitante debe seguir para generar la solicitud de compra. Estos pasos son *Confirmar canasta de compras*, *Establecer medio de pago*, *Confirmar o registrar datos personales*, *Confirmar o definir datos para envío* y *Confirmar solicitud de compra*. Posteriormente está la fase de logística donde en muchos casos, el cliente puede ver el estado del despacho de su producto o la empresa le informa al cliente. Como esta fase depende de los sistemas y procedimientos logísticos de la empresa, no se incluirá modulo alguno que trate la información del estado de la solicitud.

8. **Confirmar canasta de compras :** Para lo que es el *Proceso de compras* el visitante puede visualizar todos aquellos productos que ha seleccionado para la compra. En este ambiente puede modificar cantidades y eliminar productos. Estas acciones son manejadas por un *Administrador de canasta*. Siempre que el visitante visualiza el contenido de la canasta se encontrará en este proceso, que es el primero en el Proceso de Compras.

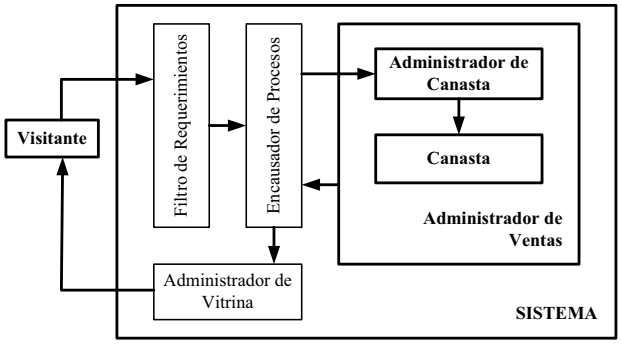

Figura 2.8: Canasta de Compras

9. **Establecer medio de pago :** Principalmente este proceso involucra escoger, dentro de un grupo de alternativas, la forma de pago que se empleará para la compra, y dependerá de lo que la empresa estime conveniente. Un *Administrador de pagos* se encargará de generar las alternativas y validar inicialmente el medio de pago seleccionado.

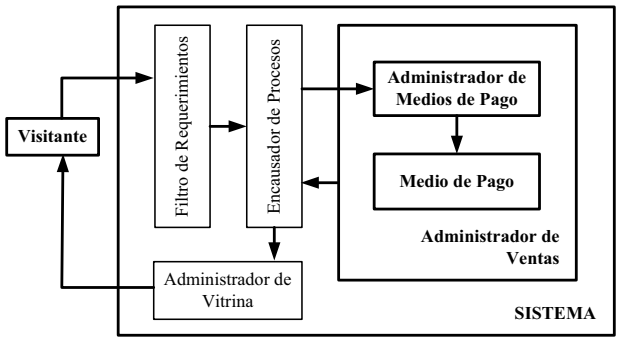

Figura 2.9: Medio de Pago

10. **Confirmar o registrar datos personales :** Si el visitante ya se registró previamente, debe confirmar sus datos personales y/o agregar nueva información para un futuro contacto. El proceso llevado a cabo es similar al de la figura 2.10

11. **Confirmar o definir datos para envío :** Adicionalmente, cada visitante puede definir varias direcciones de despacho. Estas se pueden definir en un registro previo o en esta etapa. Similar proceso al de la figura 2.10.

12. **Confirmar solicitud de compra :** Finalmente, el visitante visualiza un resumen de la compra que efectuará, sus datos para contacto y los datos para envío que deberá confirmar. Una vez que da su respuesta afirmativa, se genera la solicitud para uso interno de la empresa. Este el proceso final entre visitante y sistema.

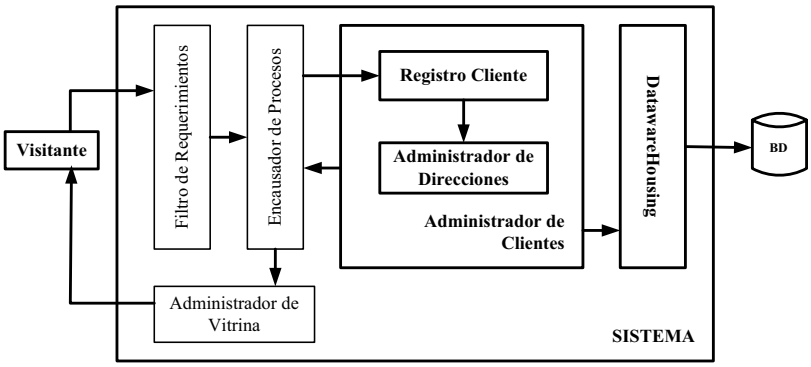

Figura 2.10: Registro de Datos Comprador y Despacho

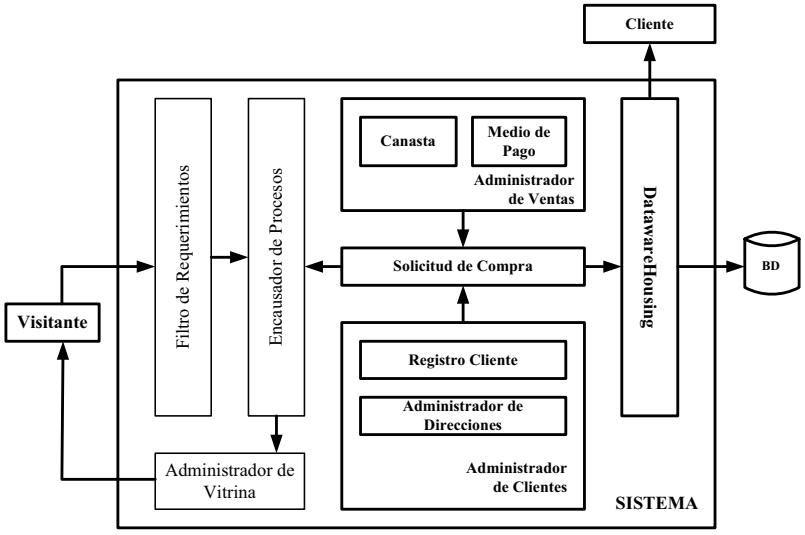

Figura 2.11: Confirmar solicitud de compra

2.2.2 Procesos Sistema-Sistema

Esencialmente un PSS se referirá a los procesos internos del sistema, como son el manejo de sesión de un cliente, administración de cache, transacciones con la base de datos, generación de datos para visualización, mensajería y datawarehousing.

1. **Manejo de sesión :** El sistema debe ser capaz de individualizar a cada visitante, y a este concepto se le denominará *Sesión*. Existen diversas alternativas para el manejo de sesión, para este caso se utilizará el manejo de sesión via *Cookies* que son objetos únicos que son intercambiados entre el sistema y la aplicación (browser) del cliente. Al momento de que un visitante ingresa al sitio es generada una sesión única que lo identifica, de esta manera el cliente puede ver su estado con respecto al sitio (canasta, registros, etc.).

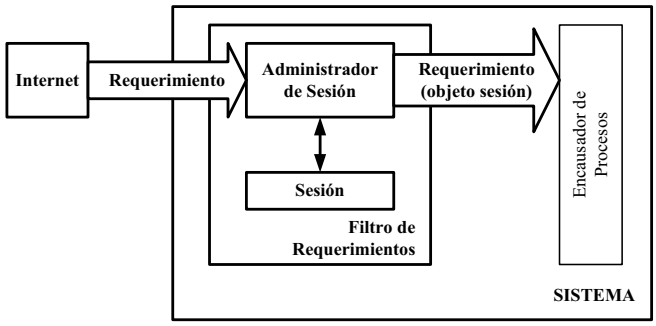

Figura 2.12: Manejo de Sesión

2. **Administración de Cache :** Una cache agiliza la obtención de información y esto es muy importante en un sitio de comercio electrónico, ya que el servidor se sometido a una importante carga debido a los múltiples accesos de diversos clientes conectados al sitio simultáneamente. Bajo estas circunstancias, una cache es ideal para almacenar aquellos artículos, catálogos, resultado de búsquedas y proceso de compras en un almacén de rápido acceso. Cada administrador solicitará un dato a la cache y éste a su vez utilizará el método LRU[2] para actualizar su contenido y entregará el dato requerido. La figura 2.13 muestra un esquema de los elementos involucrados.

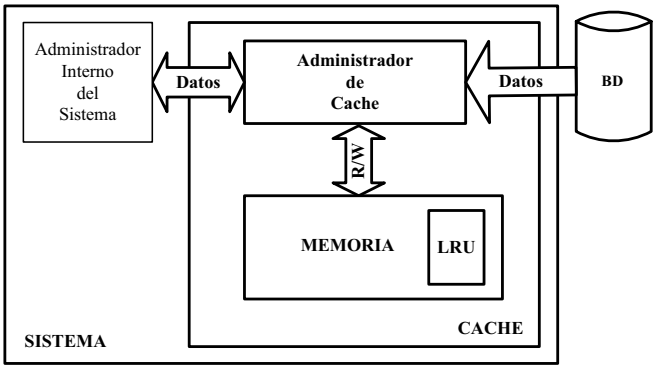

Figura 2.13: Manejo de Datos en Cache

3. **Generación de datos visualizables :** A medida que se presentan las diversas páginas, elementos visualizables son generados dentro del sistema, entre éstos podemos destacar elementos html de selección, contenidos de canasta, contenidos de artículos, catálogos, solicitud de compra, medios de pago, banners, texto y datos del visitante.

[2] *LRU* Least Recently Used. Método que determina cuál es el recurso menos usado recientemente. Esto es, por ejemplo, saber qué artículo ha sido el menos visitado, dentro de un conjunto de artículos, y reemplazarlo en la cache por otro requerido recientemente.

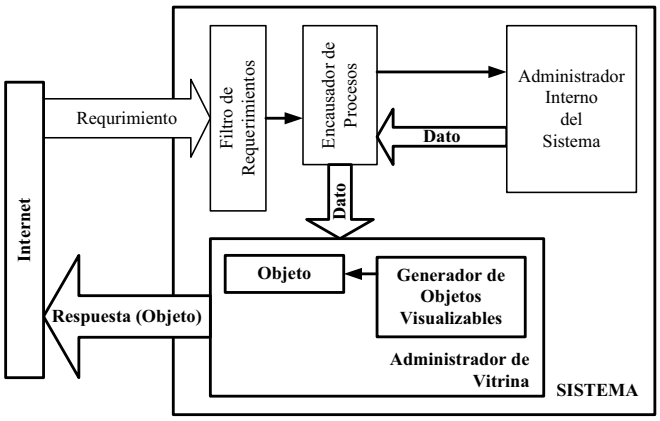

Figura 2.14: Generación de datos visualizables

4. **Mensajería y Datawarehousing:** La mensajería se realiza en base a envíos de correo electrónico y/o escritura de archivos de tipo logs. Este proceso se realiza ante eventos de los cuales se desean estadísticas, como son las visitas realizadas, solicitudes hechas, contactos del visitante, catálogos visitados o artículos solicitados.

Capítulo 3
Elaboración : *Análisis y diseño*

3.1 Casos de Uso

Los casos de usos son descripciones sobre la interacción del sistema con su entorno. Los siguientes casos de uso siguen la estructura descrita por Craig Larman [CL2002].

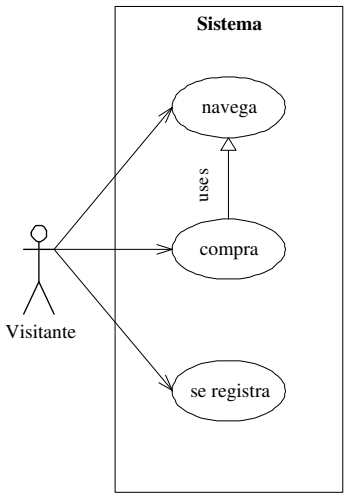

Figura 3.1: Diagrama de Casos de Uso

La figura 3.1 es un diagrama muy usado para visualizar los casos de uso. Sin embargo, no entrega mucha información para los futuros desarrollos. La información real, está contenida en las descripciones realizadas. De los tres casos de uso siguientes, el framework a desarrollar considerará sólo el primer caso de uso (CU.01) que contempla la creación de los elementos visualizables necesarios para la navegación, junto con el manejo de la cache descrito en el modelo de negocios. No se incorporará en esta version del framework el manejo de sesión.

Caso de Uso (CU.01)

Navega

Actor

Visitante

Descripción

El visitante recorre catálogos, promociones e información contenida en el sitio en búsqueda de un producto de interés.

Precondiciones

Sin precondiciones.

Postcondiciones

Visitante visualiza lista de artículos de interés.

Flujo Principal

(1) Visitante ingresa al sitio de comercio electrónico.
(2) Sistema muestra página principal.
(3) Visitante selecciona catálogo.
(4) Sistema muestra página de catálogo.
(5) Visitante selecciona subcatálogo de interés.
(6) Sistema muestra página de artículos.

Flujo Alternativo

(3A) Visitante selecciona un enlace, banner, realiza búsqueda, se registra, visualiza canasta o realiza un contacto.
(4A) Catálogo seleccionado no contiene subcatálogos. Continuar en paso (6).
(5A) Visitante selecciona un enlace, banner, realiza búsqueda, se registra, visualiza canasta o realiza un contacto.
(6A) Subcatálogo contiene a su vez subcatálogos. Volver al paso (4).
(7A) Visitante selecciona un enlace, banner, realiza búsqueda, se registra, visualiza canasta o realiza un contacto.

Flujo Excepcional

(4E) Catálogo se encuentra discontinuado. Actualizar catálogos. Mensaje a visitante. Regresar a página anterior.
(6E) Catálogo o subcatálogo no posee artículos (discontinuados). Actualizar artículos y catálogos. Regresar a página anterior.

Caso de Uso (CU.02)
Se registra
Actor
Visitante
Descripción
El visitante ingresa sus datos personales al sitio.
Precondiciones
Sin precondiciones.
Postcondiciones
Visitante visualiza sus datos ingresados y recibe una notificación por correo electrónico
Flujo Principal
(1) Visitante ingresa al sitio de comercio electrónico
(2) Sistema muestra página principal
(3) Visitante selecciona registro de clientes
(4) Sistema muestra página de acceso a clientes
(5) Visitante selecciona ingresar nuevo registro
(6) Sistema muestra formulario con datos personales
(7) Visitante ingresa información obligatoria
(8) Sistema muestra página con datos ingresados y notifica a visitante.
Flujo Alternativo
(3A) Visitante selecciona un enlace, banner, realiza búsqueda, selecciona un catálogo, visualiza canasta o realiza un contacto.
(5A) Visitante ingresa datos para ser identificado por el sistema como Cliente existente.
(5B) Visitante selecciona un enlace, banner, realiza búsqueda, selecciona un catálogo, visualiza canasta o realiza un contacto.
(7A) Visitante selecciona un enlace, banner, realiza búsqueda, selecciona un catálogo, visualiza canasta o realiza un contacto.
Flujo Excepcional
(4E) Visitante ya se encuentra con acceso. Mensaje al Visitante. Página con datos personales.
(8E1) Visitante ya se encuentra registrado. Mensaje de confirmación para sobreescribir datos.
(8E2) Visitante ingresó datos incorrectos. Mensaje al Visitante. Página de formulario anterior.

Caso de Uso (CU.03)

Compra

Actor

Visitante

Descripción

El visitante comienza el proceso de compra.

Precondiciones

El visitante tiene acceso y posee una canasta de compra con productos.

Postcondiciones

Visitante visualiza información de la solicitud de compra y recibe un mail.

Flujo Principal

(1) Visitante visualiza su canasta de compras y escoge Iniciar compra

(2) El sistema muestra página Medio de Pago

(3) Visitante selecciona medio de pago e ingresa información relacionada

(4) Sistema muestra página para confirmar datos personales

(5) Visitante modifica o confirma datos personales

(6) Sistema muestra página para confirmar datos de envío

(7) Visitante modifica, agrega o confirma datos de envío

(8) Sistema muestra resumen con Solicitud de Compra, Datos Personales y Datos de envío.

(9) Visitante confirma solicitud de compra y recibe mail con resumen.

Flujo Alternativo

(1A) Visitante selecciona un enlace, banner, realiza búsqueda, se registra, selecciona un catálogo o realiza un contacto.

(3A) Visitante selecciona un enlace, banner, realiza búsqueda, visualiza canasta, se registra, selecciona un catálogo o realiza un contacto.

(5A) Visitante selecciona un enlace, banner, realiza búsqueda, visualiza canasta, se registra, selecciona un catálogo o realiza un contacto.

(7A) Visitante selecciona un enlace, banner, realiza búsqueda, visualiza canasta, se registra, selecciona un catálogo o realiza un contacto.

(8A) Visitante selecciona un enlace, banner, realiza búsqueda, visualiza canasta, se registra, selecciona un catálogo o realiza un contacto.

Flujo Excepcional

(3E) Información ingresada es incorrecta. Mensaje. Página anterior.

(5E) Información ingresada es incorrecta. Mensaje. Página anterior.

(7E) Información ingresada es incorrecta. Mensaje. Página anterior.

3.2 Elección de la tecnología a utilizar

Entre las variadas tecnologías que soportan un Sitio B2C se destaca el empleo de APPLETS, CGI, SERVLETS y JSP.

En el caso de los CGI (Common Gateway Interface) una aplicación se ejecuta en una máquina servidor que provee el servicio para los diversos procesos ya descritos. En general esta aplicación se desarrolla con programación en C o Perl, sin embargo para una arquitectura tan compleja como la aquí descrita estos lenguajes no aportan las herramientas necesarias que faciliten el concepto de desarrollo orientado a objeto que se requiere para una óptima implementación [JSP98].

Los APPLETS, mayoritariamente desarrollados en JAVA tiene la desventaja que son aplicaciones que se ejecutan en la maquina cliente, pues sobrecargan la máquina del cliente y no permiten una transacción segura al momento de transferir numerosa cantidad de datos, teniéndose que incurrir en diversas metodologías para asegurar una correcta transacción de información entre cliente y el sistema que almacena la información de las órdenes de compra.

Los SERVLETS son módulos escritos en Java, que al igual que los CGI son ejecutados en una máquina servidor, pero que poseen la ventaja de generar documentos dinámicos, manejo de sesiones, utilización de cookies, tecnología multihilo, y por sobre todo el concepto de orientación a objeto que brinda el lenguaje JAVA. Adicionalmente se utiliza JSP (Java Server Pages) que brinda la interfaz gráfica con la que interactúa el cliente. Estas dos herramientas permiten a un cliente visualizar (mediante un browser en Internet) la información necesaria de productos y catálogos, para luego escoger, registrarse y generar solicitudes de órdenes, cuya complejidad es manejada en la máquina servidor a través de un Servlet [SUN2003].

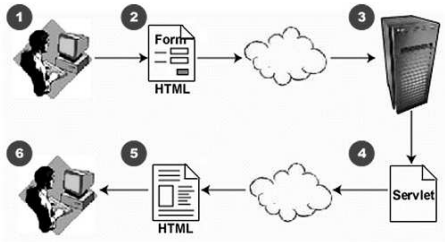

Figura 3.2: Usando Servlets

La figura 3.2 muestra la forma más común del uso de Servlets. (1) Un usuario solicita alguna información llenando algún formulario o haciendo click en algún enlace dentro de la página que visualiza en (2). Un servidor (3) escuchando protocolo HTTP recibe la petición del usuario y realiza procedimientos para reunir la información solicitada. Una vez reunida la información, mediante el Servlet (4) se genera una página con el contenido solicitado (5) que

es enviado al usuario (6).

Dado que este esquema nos proporciona una vía de implementación hacia lo que se requiere, es este modelo el que se utilizará para el desarrollo del framework para sitios de comercio electrónicos B2C.

3.3 Diagrama conceptual

Un diagrama conceptual es la primera instancia que se realiza para traspasar aquellas ideas de la realidad presentadas en los requerimientos y modelo de negocios hacia un entorno asbtracto. Esencialmente significa identificar la relación entre objetos y conceptos. Un primer paso para elaborar este diagrama es centrarse en los elementos que se describieron en la sección 2.1. A partir de aquí sólo se enfocará a resolver el caso de uso CU.01

3.3.1 Productos

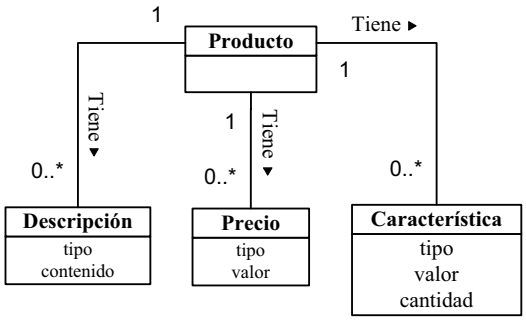

Figura 3.3: Diagrama Conceptual para Productos

Un producto, como bien final, contiene esencialmente tres conceptos adicionales. Por un lado, puede o no tener múltiples descripciones, como por ejemplo el nombre de la marca, el modelo, o el mismo nombre del producto. A su vez, cada una de estas descripciones le pertenecerá sólo a ese producto determinado. De igual forma, para cada producto se pueden definir diversos precios y características (precio de venta, precio de referencia, precio de costo, tamaño, color, piezas, etc.)

3.3.2 Artículos

Los artículos representan visualmente productos. Pueden contener descripciones, características y precios adicionales, para diferenciarlas de las cualidades propias de cada producto. Como elemento visualizable, el artículo puede tener diversos tipos imágenes asociadas (pequeña, mediana, grande).

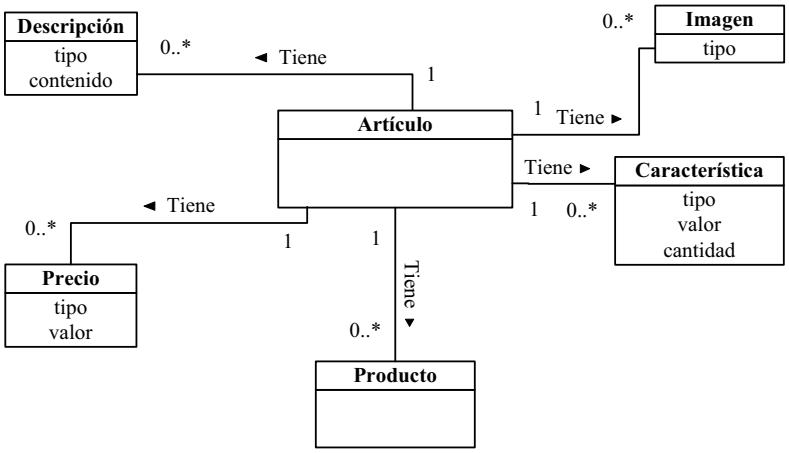

Figura 3.4: Diagrama Conceptual para Artículos

3.3.3 Promociones

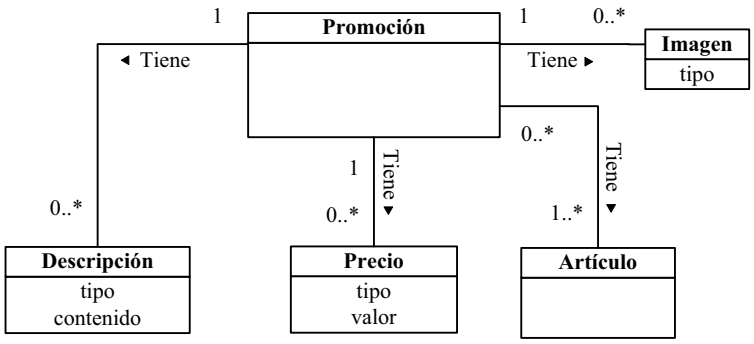

Figura 3.5: Diagrama Conceptual para Promociones

Una promoción, puede tener descripciones, precios, características e imágenes como los elementos anteriores. Maneja la información de uno o más artículos con precios diferenciados, esto es, cada vez que un artículo está en promoción, su precio puede cambiar.

3.3.4 Catálogos

Un catálogo puede contener artículos y/o promociones, y como elemento visualizable, imágenes y descripciones. El significado de que un catálogo pueda o no tener elementos (artículos y/o promociones) indica que pueden existir catálogos vacíos con fines administrativos (elaboración de catálogos, previa presentación en el sitio).

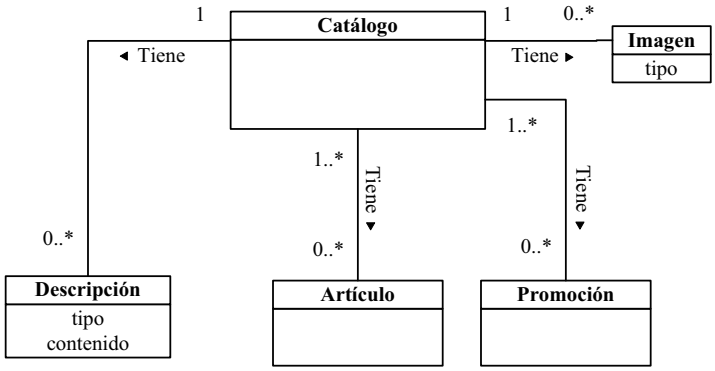

Figura 3.6: Diagrama Conceptual para Catálogos

3.4 Interfaz Cliente - Sistema

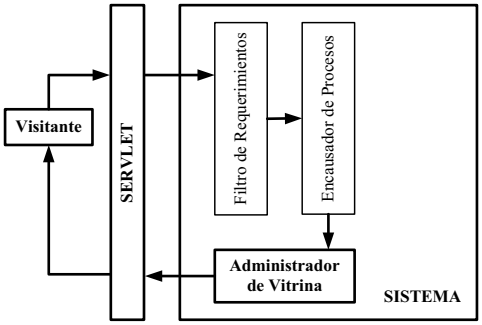

Figura 3.7: Servlet, la interfaz entre el cliente y el sistema

Entre el cliente y el sistema existe una interfaz que recibe las peticiones HTTP, las analiza y las dirige dentro del sistema. Esta interfaz es el Servlet, ver figura 3.7.

Un Servlet consiste en una clase que implementa *javax.servlet.Servlet*. En muchos casos, se utiliza la clase *javax.servlet.http.HttpServlet* que además de implementar la clase mencionada, incorpora otras funcionalidades. Esta clase posee un método llamado *service()* que recibe dos parámetros, un objeto *request* y un objeto *response*, ambos relacionados con el mensaje de llegada al servlet y el mensaje de salida hacia el cliente respectivamente. Este método llama a su vez a otros métodos dependiendo del método utilizado en el mensaje HTTP (GET, POST, PUT, DELETE, OPTIONS, ver más en [JSP98]). Para el caso aquí tratado, sólo será necesario utilizar los métodos que manejan GET y POST, estos métodos son:

doGet()
doPost()

Estos dos métodos poseen como parámetros dos objetos cuyos tipos también pertenecen al paquete *javax.servlet.http* y son *HttpServletRequest* y *HttpServletResponse* que manejan la información proveniente del cliente (párametros, formularios, cookies, encabezados, etc.)

Adicionalmente, un servlet posee el método *init()* que se ejecuta cada vez que se crea una clase *javax.servlet.http.HttpServlet*, por lo tanto en este método se incluyen aquellos procesos de inicialización. En general, por razones de seguridad y cantidad de datos involucrados, se utiliza mayoritariamente el método *doPost()*. En casos especiales, cuando es necesario permitir al cliente hacer llamadas parametrizadas a través de la URL, se utiliza el método *doGet()* separadamente. Sin embargo, hay que tomar en cuenta, que una llamada inicial al Sitio de Comercio (página inicial del sitio) usa por defecto el método GET, de tal forma que se debe encausar el método *doGet()* hacia el método *doPost()*. Para seguridad (y esto es una decisión particular) puede crearse en el método *doGet()* un filtro que evite la entrada de datos maliciosos[1]

Para la generación de páginas con código HTML que es lo que finalmente el cliente recibe como visualización, se incorpora estructuras que envían flujos de información hacia los servicios web encargados de generar la respuesta hacia el cliente, pero para efectos de este framework se utilizará la salida incovando un JSP (Java Server Page) que es un archivo estructurado, similar a un documento HTML pero que además puede contener código JAVA en su estructura (no confundir con JavaScript). De esta forma, a través de los objetos *HttpServletRequest* y *HttpServletResponse* podemos transferir objetos con información desde el sistema y dentro de un JSP procesar esa información para efectos de visualización. Esto permite poder separar la implementación de páginas web con diseño especializado de la implementar el funcionamiento propiamente tal del sistema.

3.5 Variables de entorno

Para darle más flexibilidad al sistema es importante considerar las llamadas *Variables de entorno*, estas variables permiten ajustar diversas funcionalidades a distintos entornos, por ejemplo, la conexión del sistema a una base de datos externa utilizando los parámetros de conexión como variables de entorno.

Se puede considerar que existen tres instancias o momentos para establecer o definir variables de entorno, que da origen a tres tipos de variables.

1. **Variables de Inicialización**: Son aquellas cuyos valores sólo se definen al iniciar el sistema ya que la información utilizada de estas variables no cambia con el tiempo y deban cambiarse, por ejemplo, cuando la base de datos es cambiada[2].

[1] Hoy en día, muchos sitios son vulnerables a ataques via el método GET, mediante el cual se incorporan datos que son interpretados por el sistema para realizar operaciones que pueden dañar el desempeño de la interfaz.

[2] Generalmente, en estos casos, se realizan otras configuraciones externas como son el cambio de servidores, ajuste de clientes de Base de Datos, configuraciones de red, etc. Que implica detener el sistema.

2. **Variables Dinámicas**: Este tipo de variables son utilizadas cuando se espera que su información cambie durante el funcionamiento del sistema, por lo tanto el valor de estas variables es definido en cada requerimiento realizado al Sistema. Por ejemplo, direcciones de destino de correo, cantidad de artículos visualizables por pantalla, etc.

3. **Variables Circunstanciales**: Estas son un tipo de variables que son definidas en determinados momentos y pueden o no perdurar en el tiempo. Por ejemplo, parámetros que dependen de festividades del año (promociones, regalos, etc) cuyos valores cambian en determinadas épocas del año y perduran días o semanas.

La forma usual para definir u obtener el valor de estas variables es mediante la lectura de un archivo, que contiene el identificador de la variable y su valor. Para efectos de este framework, no será necesario la utilización de estas variables, pero si en la etapa de transición cuando se ponga a prueba el framework a través de una aplicación web.

Capítulo 4
Construcción : *Implementación de código*

4.1 Diagramas de Clase

4.1.1 Elementos básicos

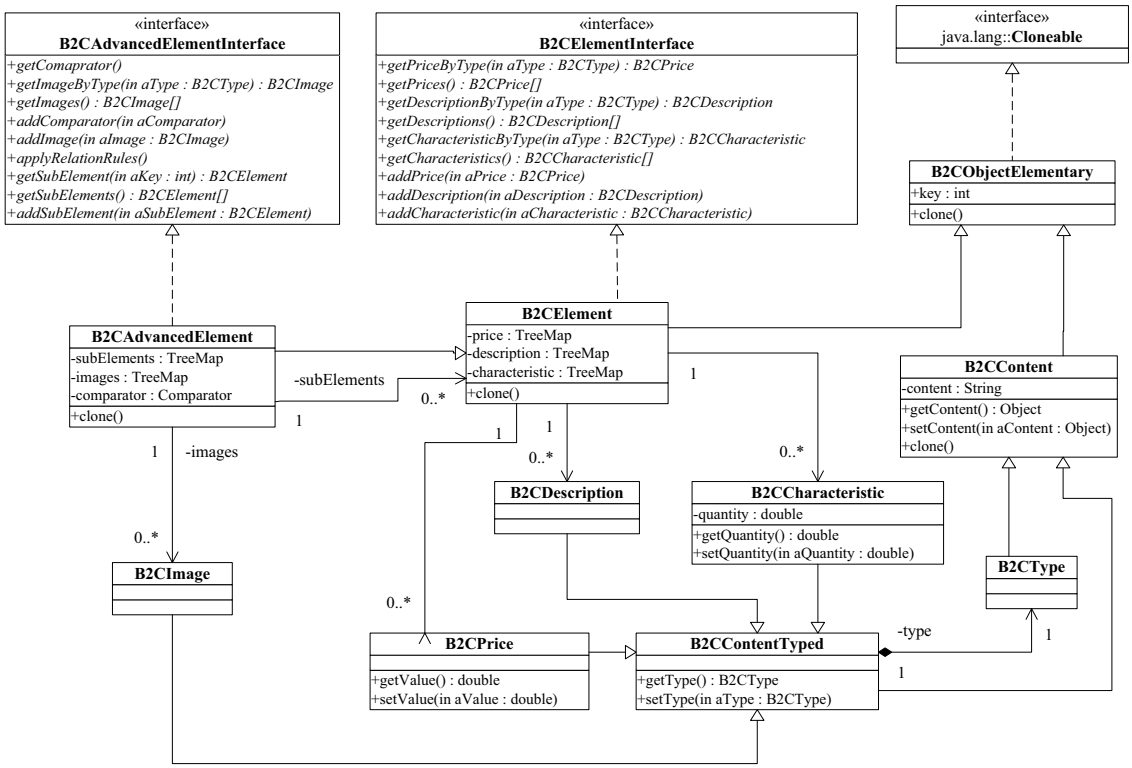

Figura 4.1: Diagrama de clases para elementos básicos

El diagrama de clases de la figura 4.1 muestra la jerarquía de elementos básicos para la construcción de los objetos que sustentarán el framework. La clase *B2CObjectElementary* corresponde a la más elemental de las clases, posee una propiedad denominada *key* que permitirá identificar a las instancias creadas y además implementa la interfaz *Cloneable* que permitirá definir el método *clone()* necesario para obtener copias de los objetos. Esto es ya que los objetos en Java son pasados por referencia, lo que indica que cualquier modificación a una instancia y que se almacene en la cache permanecerá con esta modificación, y lo que se quiere en realidad es que los objetos iniciales que se guarden en la cache perduren. Extienden a esta clase *B2CContent* y *B2CElement*, la primera es una clase con un atributo para contenidos y

sus respectivos métodos, de ésta descienden *B2CType* que es una clase para definir el tipo de un objeto, y *B2CContentType* que es una clase con contenido y tipo. De esta última descienden *B2CPrice*, *B2CDescription* y *B2CCharacteristic* que son utilizados por *B2CElement*

B2CElement implementa la interfaz *B2CElementInterface* que contiene los métodos necesarios para obtener y agregar precios, descripciones y características. Para almacenar estas cualidades se utilizarán objetos de la clase *java.util.TreeMap*, que permite contener objetos en forma de mapeo ordenado mediante una clave de entrada y el objeto a almacenar. El orden se realiza en base a estas claves y se puede establecer al momento de crear el mapa ingresando un objeto que implemente *java.util.Comparator*, sin embargo para este caso, bastará con el orden natural (orden de ingreso al mapa). Las claves de mapeo aquí utilizadas corresponderán a la *key* del tipo de elemento, y ya que el mapeo debe ser de objetos, se generará un objeto *java.lang.Integer* con la *key*.

B2CAdvancedElement extiende a *B2CElement* y posee dos mapas que podrán contener subelementos del tipo *B2CElement* y objetos *B2CImage* que son del tipo *B2CContentTyped* para describir imágenes. El mapeo de las imágenes se realizan en base a la *key* del tipo de imagen encapsulada en un objeto *java.lang.Integer*. Al implementar *B2CAdvancedElementInterface* obtiene los métodos para obtener y agregar subelementos e imágenes, adicionalmente de asignar o utilizar el comparador.

4.1.2 Elementos principales

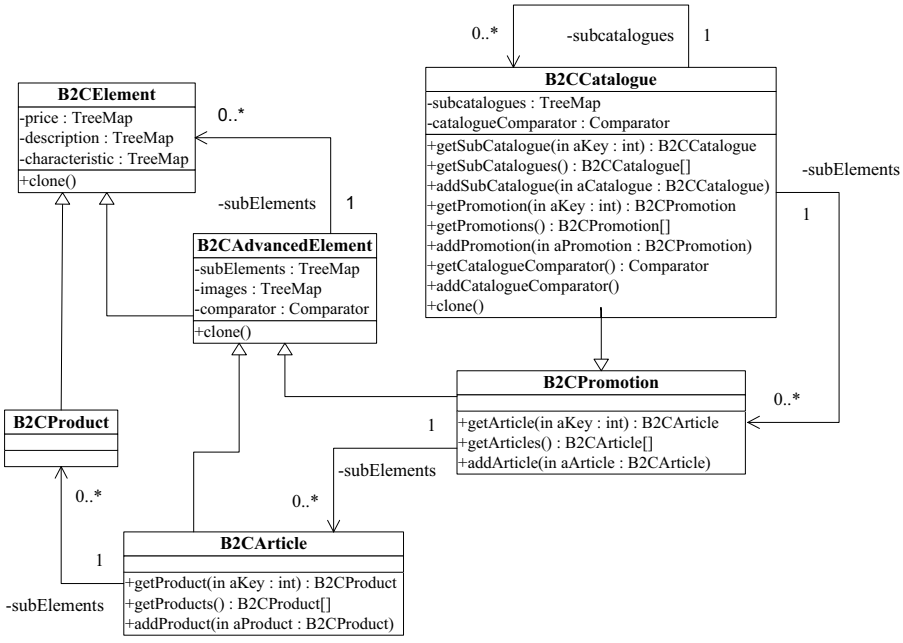

Figura 4.2: Diagrama de clases para elementos principales

Con los elementos de la figura 4.1 procedemos a diseñar los elementos principales para productos, artículos, promociones y catálogos.

Un producto implementado por *B2CProduct* extiende a *B2CElement* por lo que contiene descripciones, características y precios.

El artículo lo implementa *B2CArticle* que extiende a *B2CAdvancedElement* y del cual hereda manejar subelementos (en este caso, productos), imágenes y un comparador para ordenar los subelementos. Se incorporan los métodos para agregar y obtener productos que simplemente utilizan los métodos heredados pero referido a elementos del tipo *B2CProduct*.

Una promoción extiende también a *B2CAdvancedElement*, a diferencia de tener como subelementos objetos del tipo *B2CArticle* (artículos).

Un catálogo *B2CCatalogue* es prácticamente una promoción pues su objetivo es contener artículos, así, hereda las funcionalidades para manipular artículos, pero adicionalmente incorpora un mapa y métodos adicionales para manejar promociones (*B2CPromotion*) y subcatálogos (*B2CCatalogue*). Nótese que para un catálogo, una promoción es igual a un artículo, por lo tanto se utiliza el mismo mapa (subElements) para almacenar ambos elementos. Adicionalmente se incorpora un comparador para ordenar los subcatálogos.

4.1.3 Elementos de Cache

El principal objetivo para la construcción de los elementos de la cache, son generar los componentes para el funcionamiento del algoritmo LRU de cache y permitir la utilización de métodos personalizados para la inicial obtención de elementos.

El principal elemento en la figura 4.3 es *B2CElementCache*, esta clase representa a la cache propiamente tal. Contiene un mapa de elementos (*elementCacheMap*), una tamaño máximo (*maxSize*) y un tamaño preventivo o de emergencia (*sizeWarning*) que se ha establecido como el 5% del tamaño máximo para evitar que la cache se sature ante eventuales accesos múltiples. La elección del porcentaje de resguardo se escogió sin ningún criterio objetivo, sino más bien, para obtener una proporción pequeña cercana al límite máximo y aprovechar en forma más completo el recurso.

La clase posee dos métodos, que son el constructor único, mediante el cual se asigna el tamaño que tendrá la cache y un método para obtener algún elemento contenido. Los elementos son del tipo *B2CElement* para garantizar poder incluir los elementos descritos en las secciones anteriores.

B2CElementCache hace uso de la clase, de tipo singleton, *B2CLRUElement* que es el responsable de implementar el algoritmo LRU de la cache. Esta clase, posee un mapa para guardar estadísticas de la cantidad de peticiones de elementos, y métodos para agregar, remover y calcular el mínimo de peticiones. El mapa contiene elementos del tipo *B2CLRUObject* que posee información sobre la *key* del elemento en la cache y las veces que ha sido solicitado. *B2CElementCache* utiliza el método *getLruElement()* de *B2CLRUElement* para obtener la *key* del elemento menos solicitado, y asi removerlo de la cache, cuando sea necesario.

Cuando la cache no tiene elementos, los obtiene utilizando los métodos personalizados de

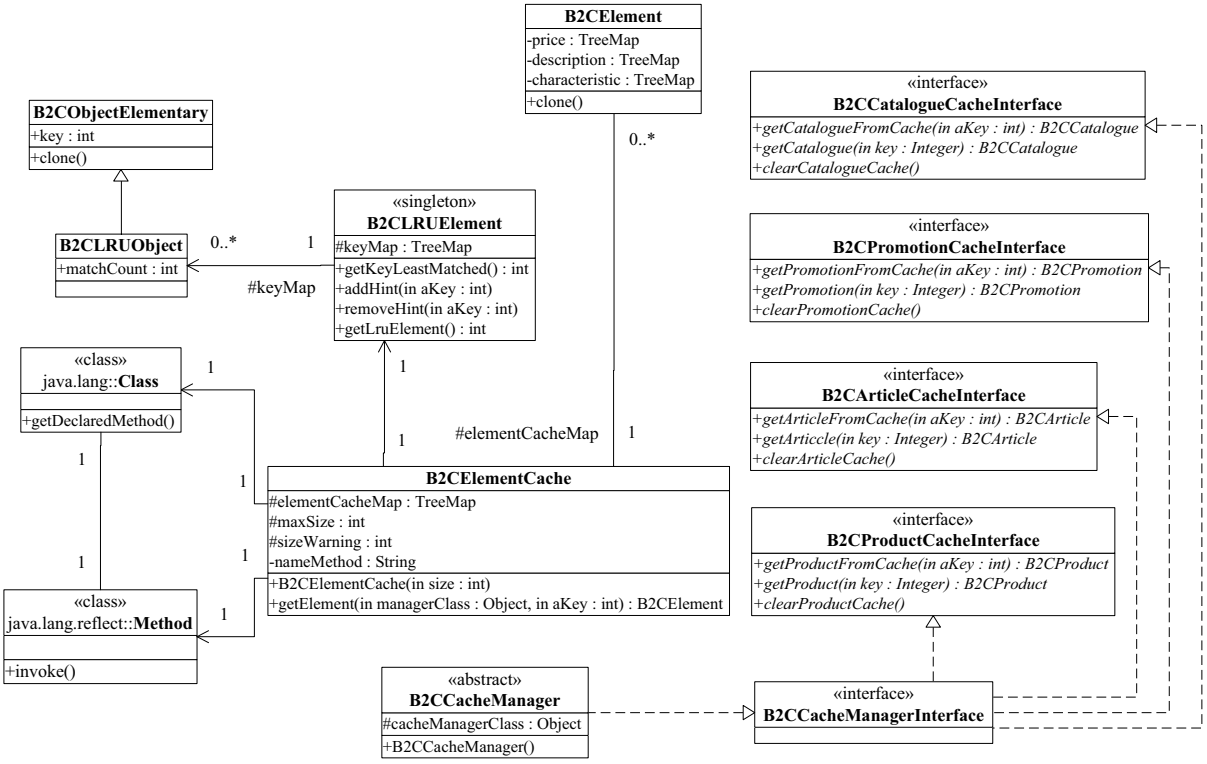

Figura 4.3: Diagrama de clases para elementos de la cache

quien utilice el framework. Esto se logra, a través de *B2CCacheManager* que contiene métodos que deben ser implementados para obtener estos elementos. De esta forma, *B2CElementCache* detecta la clase implementada, busca los métodos implementados y los invoca. Extenderán de *B2CElementCache* clases para cada elemento (Productos, Artículos, Promociones y Catálogos), permitiendo tener para cada uno de estos elementos una cache independiente. Así mismo, *B2CCacheManager* implementa las interfaces que contienen los métodos ya construidos y los que deben construirse para estos elementos.

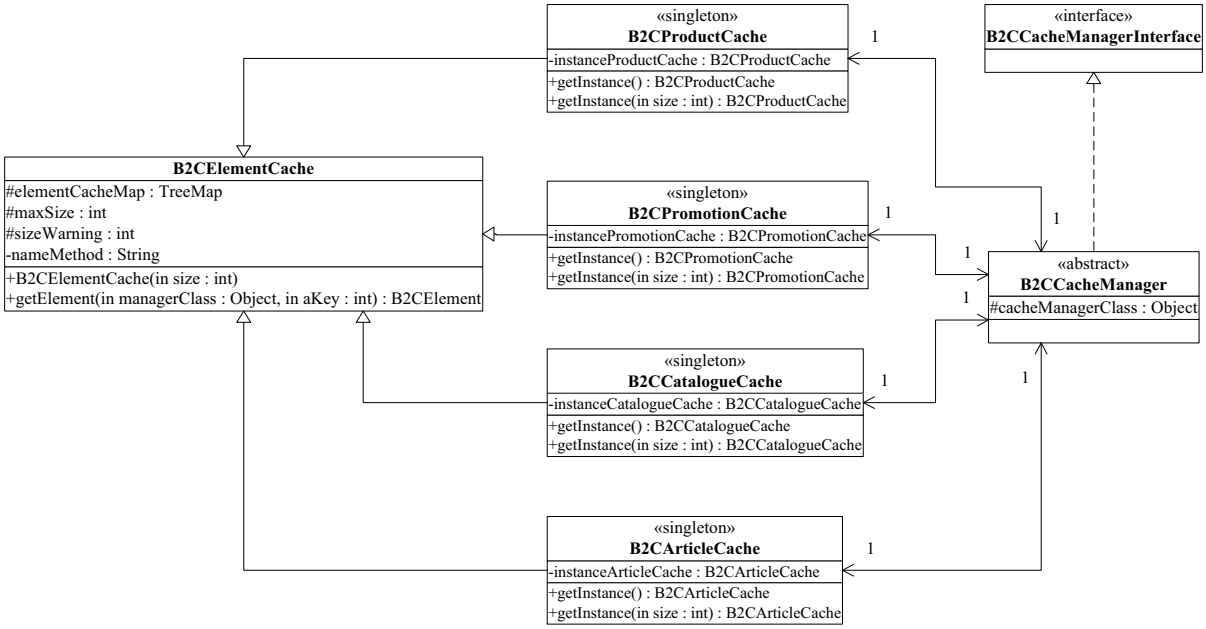

Figura 4.4: Diagrama de clases para elementos principales de la cache

Las clases que extienden a *B2CElementCache* se muestran en la figura 4.4 y son de tipo *singleton*. En su constructor único se debe especificar el tamaño de la cache, y adicionalmente se establece el atributo *nameMethod* heredado de *B2CElementCache* con el nombre del método en *B2CCacheManager* que implementa la obtención del recurso en forma externa (por ejemplo, desde una base de datos).

4.1.4 Elementos de la Interfaz Cliente-Sistema

De la figura 4.5, la clase *B2CServlet* hereda de *HttpServlet* que implementa la funcionalidad básica para el manejo de la información entre un cliente y el servidor a través del protocolo HTTP. Esta clase, se relaciona con *B2CRequestFilter* que será la clase que implementa al Filtro de Requerimientos, y con *B2CVisualManaher* que se encargará del Administrador de Vitrina. *B2CServlet* posee dos métodos que se deben implementar y que son: *setVisualManager()* y *setRequestFilter()* cada uno establece el nexo hacia el objeto que implementará el Administrador de Vitrina y el Filtro de Requerimientos respectivamente, esto se hace necesario a modo de que se pueda personalizar el uso de alguna otra clase hija. *B2CServlet* en su método *init()* llamará a estos métodos.

También se incorporan métodos para establecer el tamaño de las cache de productos, artículos, promociones y catálogos. Finalmente posee los métodos *doFilter()* que pasa el control al Filtro de Requerimientos y *goToPage* que redirecciona el flujo de salida hacia una página JSP.

La clase *B2CRequestFilter* posee un constructor que al generarse debe referenciar al

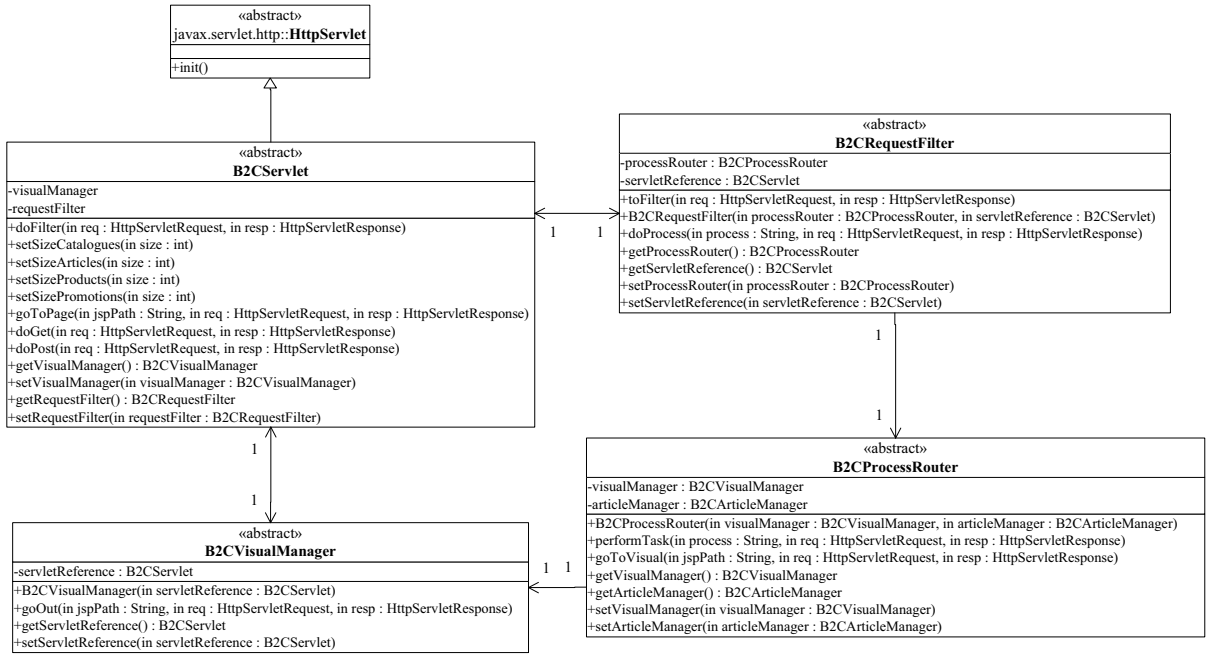

Figura 4.5: Diagrama de clases para elementos en la interfaz cliente-sistema

Servlet que se implemente, así como también su enlace con el Encausador de Procesos. Posee un método llamado *doProcess()* que sede el control al Encausador de Procesos enviándole como parámetros, una etiqueta que identifica al proceso que se realiazará, además de los objetos que contienen la información en entrada y salida hacia el cliente. Para esta clase se debe implementar el método *doFilter()* que realizará la tarea del filtraje de requerimientos.

El Encausador de procesos está implementado por la clase *B2CProcessRouter* que posee un enlace con el Administrador de Vitrina mediante su método *goToVisual()* donde se especifica la ruta de la página JSP de salida, en el contexto del Sitio Web. Los procesos los ejecuta mediante el método *performTask()* que recibe la etiqueta alusiva al proceso de interés y busca aquellos métodos contenidos en su cuerpo cuyos nombres coincidan con el nombre de la etiqueta. Estos métodos se deben implementar al momento de utilizar el framework.

El Administrador de Vitrina posee un sólo método implementado y que corresponde a delegar el control final del flujo al Servlet Principal para generar la información de salida hacia el cliente.

Capítulo 5

Transición : *Probando la aplicación*

5.1 Creando la interfaz cliente-sistema

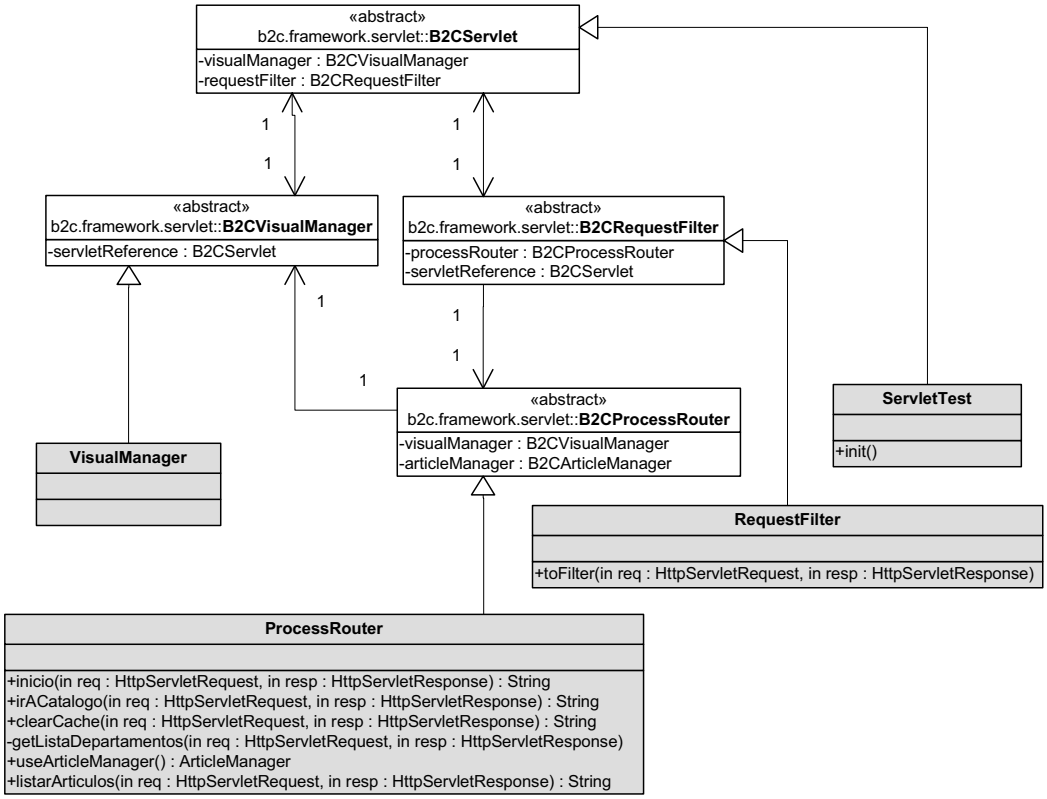

Figura 5.1: Diagram de clases para Interfaz cliente-sistema

En la figura 5.1, en gris, se muestran las clases que extienden del framework y que se utilizarán para el test. *ServletTest* implementa *init()* que inicializa el tamaño de las cache de producto, artículos, promociones y catálogos, genera los enlaces necesarios entre el Servlet, el Filtro de Requerimientos y VisualManager. *RequestFilter* implementa el método *toFilter* que busca en el parámetro de entrada *req* el atributo *"process"*. En caso de existir llama al método *doProcess()* (heredado de *B2CProcessRouter*) en *ProcessRouter* con el valor encontrado en al atributo *"process"*.

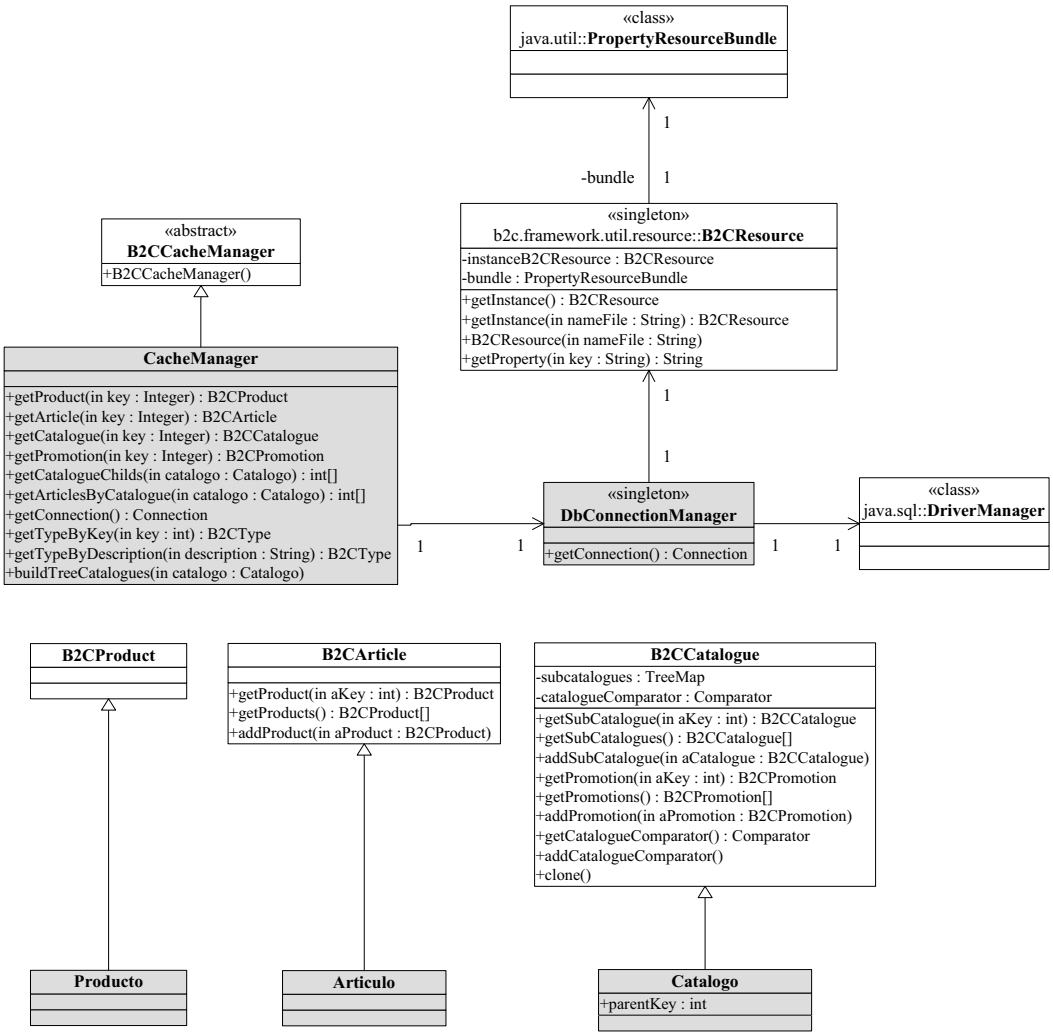

Figura 5.2: Diagrama de clases para elementos principales

5.2 Elementos principales

En la figura 5.2 se muestran cuatro elementos (clases) principales que se crean a partir del framework, que son: *Catalogo*, *Articulo*, *Producto* y *CacheManager*. A *Catalogo* se le agrega adicionalmente un atributo *parentKey* que relaciona a un determinado catálogo con su catálogo padre, pensando en una jerarquía de catálogos tipo árbol. *CacheManager* deberá implementar cuatro métodos para la obtención de productos, artículos, promociones y catálogos. Estos métodos obtendrán estos elementos desde una base de datos. Respecto a los métodos de *CacheManager* se agregan métodos utilitarios para obtener información, como por ejemplo los métodos *getTypeByKey()* y *getTypeByDescription* necesarios para crear los tipos de datos que clasificarán las descripciones, características y precios de los elementos principales. También

está el método *getCatalogueChilds()* que busca todos los subcatálogos directos para un determinado catálogo y *getArticlesByCatalogue* que obtiene los artículos asociados a un catálago. El método *buildTreeCatalogues* construye, a partir de un catálogo la jerarquía completa de catálogos.

5.3 Conexión a Base de Datos

Se requieren cuatro parámetros para establecer una conexión a la Base de Datos a utilizar, que esencialmente manejan el nexo con el driver responsable de la comunicación con la Base de datos.

Parámetro	Valor	Descripción
dbDriver	com.mysql.jdbc.Driver	JDBC Driver
dbUrl	jdbc:mysql://localhost:3306/	Url de conexión a Base de Datos MySQL
dbName	b2c	Nombre de la Base de Datos
dbUser	b2c	Usuario de acceso a Base de Datos
dbPasswd	b2c	Contraseña de Usuario

Tabla 5.1: Parámetros para conexión a Base de Datos

La clase java.util.DriverManager es un servicio básico para el manejo de drivers JDBC. De esta manera, lo primero que se hace es cargar el driver para que esta clase la administre. Ya que el método para obtener una conexión desde DriverManager solicita como parámetros de entrada dBUrl, dbUser y dbPasswd, es necesario crear una clase estática que realice este procedimiento. La clase a implementar será de tipo Singleton y de esta manera, obtendremos una conexión inmediata invocando un método getConnection desde la única instancia de esta clase. Y de esta manera utilizar las clases del paquete java.sql para nuestras acciones sobre la base de datos.

Los parámetros descritos en la tabla 5.1 podemos localizarlos en un archivo de propiedades que puede ser leído en el método init() del Servlet principal o cuando se estime necesario. De esta manera, la configuración de la conexión a la Base de Datos queda sujeta a modificar dichos parámetros en el archivo de propiedades y no tener que modificar el código, siendo esencial si, que la aplicación deberá ser reiniciada al realizar un cambio en los parámetros. En la figura 5.2 el manejo de esta conexión la establece la clase *DBConnectionManager* que utiliza la clase *B2CResource*, utilidad del framework para leer variables de entorno iniciales de la Base de Datos, y la clase *DriverManager* que inicializa la conexión.

Se utilizará un modelo sencillo para almacenar catálogos, promociones, artículos y productos. La Figura 5.3 muestra el modelo de la Base de Datos B2CTest y sus relaciones. La tabla *CATALOGUE* contiene la información de los catálogos y su jerarquía a través del campo *CA_PARENT*. El campo *TP_KEY* indica que los catálogos tienen un tipo, dado por la tabla *TYPE*. En general esta tabla contendrá las descripciones para todos los tipos utilizados. La

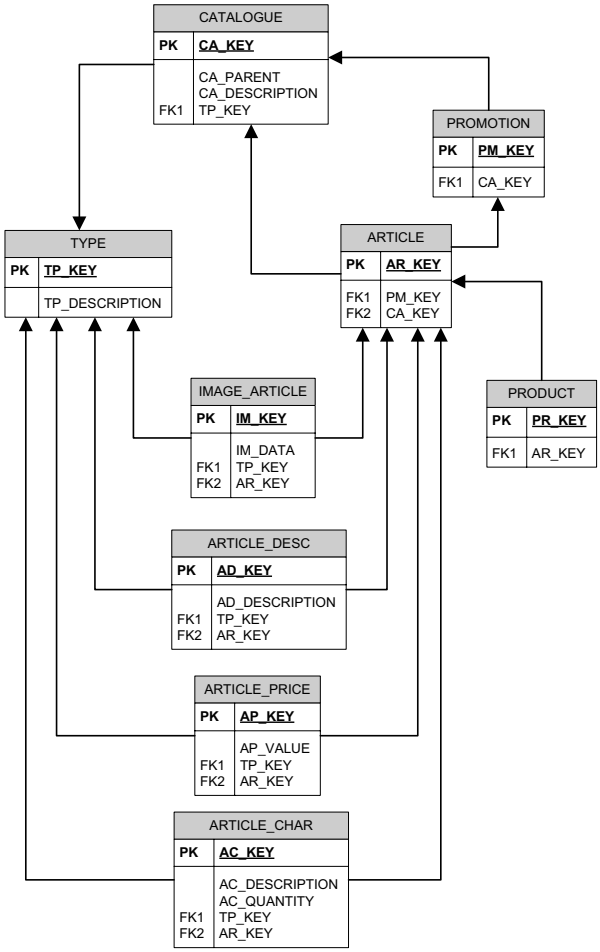

Figura 5.3: Modelo de Base de Datos B2CTest

tabla *PROMOTION* contiene un indicador (*CA_KEY*) sobre a qué catálogo pertenece, mientras que en la tabla *ARTICLE* además de este indicador, existe un campo que indicando a qué promoción pertenece (*PM_KEY*). A esta tabla están asociadas las tablas *IMAGE_ARTICLE* que contiene la ubicación de las imágenes, *ARTICLE_DESC* que contiene descripciones del artículos, *ARTICLE_PRICE* con los precios y *ARTICLE_CHAR* con las características.

5.4 Test del framework

El diagrama de secuencia de la figura 5.4 muestra la interacción entre los objetos, para el caso que ingresa por primera vez al sitio, y en el caso de que el cliente navegue por catálogos. En la clase *ProcessRouter* se implementan tres métodos esenciales ya personalizados, desde el punto de vista de la utilización del framework. Estos son *inicio()*, *irACatalogo()* y

clearCache(). El método *inicio()* realiza un llamado al método privado *getListaDepartamentos()* que obtiene el listado de catálogos principales, lo introduce como atributo en el objeto *req* y devuelve la etiqueta "index.jsp" indicando que esa será la página a desplegar. *irACatalogo()* se utiliza para obtener información de algún catálogo determinado, busca en el objeto *req* el atributo "catalogo" que le indicará de qué catálogo debe obtener la información. Por defecto esta información se refiere a subcatálogos, sin embargo puede ocurrir que no posea subcatálogos. Si este es el caso, se obtiene información de la lista de artículos contenida en ese catálogo. Por lo tanto, utilizando a *ArticleManager* se piden subcatálogos por tipo y ordenados (el tipo, dependerá de cómo se registren los catálogos en la base de datos y artículos. El ordenamiento es por nombre y en forma ascendente). El método *clearCache()* es accesible desde Internet pero con fines administrativos para limpiar las cache.

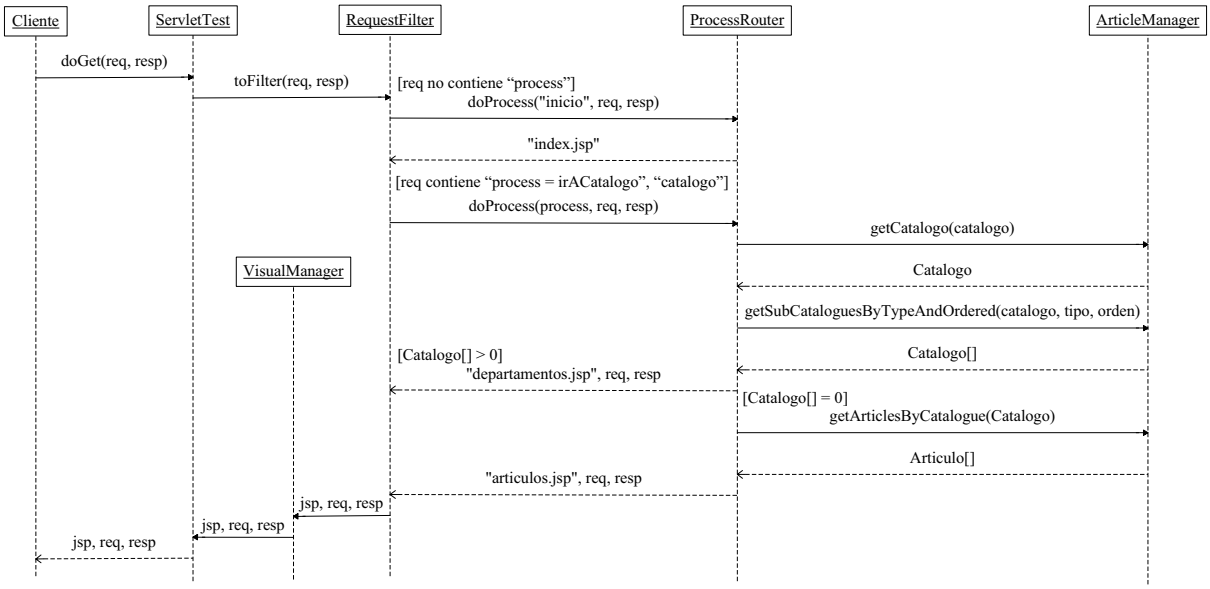

Figura 5.4: Diagrama de Secuencia para Navegación de Prueba

Según el diagrama de la figura 5.4, si el parámetro "process" no existe, por defecto el valor del parámetro será *inicio* al llamar a *ProcessRouter*, esto significa que al ingresar por primera vez al sitio, el objeto *req* no tendrá parámetros de entrada, por lo que se asumirá que se está ingresando a la página Home del Sitio.

Capítulo 6
Conclusiones y resultados

6.1 Midiendo resultados

Para medir las interacciones y resultados, se utilizarán dos instancias. La primera corresponde a incluir sentencias del tipo *System.out.println()* para detectar en qué puntos del código se encuentra la aplicación y en segundo lugar un gráfico generado por la aplicación *MySQL Administrator* respecto a las consultas realizadas a la base de datos. Se detallan y se explican a continuación dos casos, en el cual el tamaño de la cache utilizada fue de 3 y 10 elementos respectivamente.

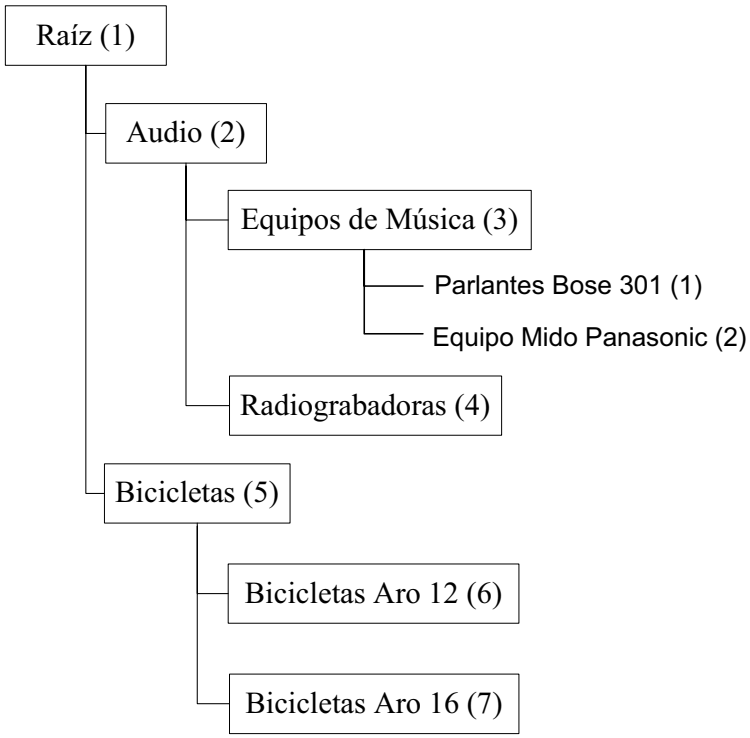

Figura 6.1: Diagrama para árbol de catálogos

La figura 6.1 muestra el árbol de catálogos creados para el test. Las etiquetas enmarcadas corresponden a catálogos y las sin enmarcar a artículos. Entre paréntesis se indica el identificador o 'key' de cada elemento. A continuación se muestra la lista de mensajes (en cursiva) y algunas notas (letra normal). El tamaño de la cache para esta medición es de 5

SystemOut U Inicializando Servlet

El primer acceso, gatilla la inicialización del Servlet

SystemOut U Utilizando doFilter() desde class test.ServletTest
SystemOut U Filtro de Requerimientos: Process = 'inicio'
SystemOut U Utilizando doProcess desde class test.RequestFilter
SystemOut U Utilizando performTask desde class test.ProcessRouter
SystemOut U Obteniendo lista de departamentos

Para obtener la lista de departamentos, primero se obtiene el catálogo 'Raíz' y sus descendientes directos.

SystemOut U class test.CacheManager pidiendo elemento: 1
SystemOut U Elemento 1 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 1 consultas

Se obtuvo y se actualizó la cache de catálogos para el catálogo 'Raíz'.

SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Elemento 2 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 1 consultas
SystemOut U Key 2 ha tenido 1 consultas

Se obtuvo el catálogo 'Audio', se actualizó cache y se muestran las estadísticas.

SystemOut U class test.CacheManager pidiendo elemento: 5
SystemOut U Elemento 5 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 1 consultas
SystemOut U Key 2 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 1 consultas
SystemOut U Creando visualizadores para departamentos
SystemOut U Direccionando hacia 'index.jsp'

Luego que se obtiene el último catálogo directo, se direcciona hacia el cliente con la página 'index.jsp' que muestra la lista de departamentos, excepto el catálogo 'Raíz'. En seguida en la página, el cliente selecciona ver el catálogo 'Audio'.

SystemOut U Utilizando doFilter() desde class test.ServletTest
SystemOut U Filtro de Requerimientos: Process = 'irACatalogo'
SystemOut U Utilizando doProcess desde class test.RequestFilter
SystemOut U Utilizando performTask desde class test.ProcessRouter
SystemOut U Obteniendo lista de departamentos
SystemOut U class test.CacheManager pidiendo elemento: 1
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 1 consultas
SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 1 consultas
SystemOut U class test.CacheManager pidiendo elemento: 5
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 2 consultas

Hasta acá, se ha repetido el proceso para obtener la lista de departamentos. El rendimiento de este proceso puede mejorarse, eventualmente, dejando este árbol de catálogos en memoria, para ser obtenido en forma directa cada vez. Como puede verse, al estar estos elementos en cache, no se requiere hacer accesos a la Base de datos, salvo, en este caso, para buscar las etiquetas del tipo 'departamento' y filtrar los catálogos. Esto se hizo a propósito para efectos de visualizar las diferencias en el gráfico del *MySQL Administrator*.

SystemOut U Búsqueda de información para catálogo: 2
SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 3 consultas
SystemOut U Key 5 ha tenido 2 consultas
SystemOut U class test.CacheManager pidiendo elemento: 3
SystemOut U Elemento 3 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 3 consultas
SystemOut U Key 3 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 2 consultas

SystemOut U class test.CacheManager pidiendo elemento: 4
SystemOut U Elemento 4 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 2 consultas
SystemOut U Key 2 ha tenido 3 consultas
SystemOut U Key 3 ha tenido 1 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 2 consultas
SystemOut U Creando visualizadores para departamentos
SystemOut U Direccionando hacia 'departamentos.jsp'

Ya que el catálogo 'Audio' posee subcatálogos, se obtiene esta información que se presentan en la página 'departamentos.jsp'. En esta página el cliente pide información para 'Equipos de Audio' que no posee subcatálogos, sólo artículos.

SystemOut U Utilizando doFilter() desde class test.ServletTest
SystemOut U Filtro de Requerimientos: Process = 'irACatalogo'
SystemOut U Utilizando doProcess desde class test.RequestFilter
SystemOut U Utilizando performTask desde class test.ProcessRouter
SystemOut U Obteniendo lista de departamentos
SystemOut U class test.CacheManager pidiendo elemento: 1
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 3 consultas
SystemOut U Key 2 ha tenido 3 consultas
SystemOut U Key 3 ha tenido 1 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 2 consultas
SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 3 consultas
SystemOut U Key 2 ha tenido 4 consultas
SystemOut U Key 3 ha tenido 1 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 2 consultas
SystemOut U class test.CacheManager pidiendo elemento: 5
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 3 consultas
SystemOut U Key 2 ha tenido 4 consultas
SystemOut U Key 3 ha tenido 1 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 3 consultas

SystemOut U Busqueda de información para catálogo: 3
SystemOut U class test.CacheManager pidiendo elemento: 3
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 3 consultas
SystemOut U Key 2 ha tenido 4 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 3 consultas
SystemOut U No hay subCatálogos. Proceder a buscar artículos

En este punto, los mensajes a continuación, se refieren a la cache de artículos.

SystemOut U Búsqueda de artículos para catálogo: 3
SystemOut U class test.CacheManager pidiendo elemento: 1
SystemOut U Elemento 1 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 1 consultas
SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Elemento 2 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 1 consultas
SystemOut U Key 2 ha tenido 1 consultas
SystemOut U Creando visualizadores para departamentos
SystemOut U Direccionando hacia 'articulos.jsp'

En seguida, el cliente elige ver el catálogo 'Bicicletas', presente en la lista de departamentos original. En este casó, la cache está completa y se activa el algoritmo LRU para actualizar la cache.

SystemOut U Utilizando doFilter() desde class test.ServletTest
SystemOut U Filtro de Requerimientos: Process = 'irACatalogo'
SystemOut U Utilizando doProcess desde class test.RequestFilter
SystemOut U Utilizando performTask desde class test.ProcessRouter
SystemOut U Obteniendo lista de departamentos
SystemOut U class test.CacheManager pidiendo elemento: 1
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 4 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 3 consultas

SystemOut U class test.CacheManager pidiendo elemento: 2
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 3 consultas
SystemOut U class test.CacheManager pidiendo elemento: 5
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 4 consultas
SystemOut U Busqueda de información para catálogo: 5
SystemOut U class test.CacheManager pidiendo elemento: 5
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 4 ha tenido 1 consultas
SystemOut U Key 5 ha tenido 5 consultas

Nótese, que según las estadísticas, el elemento de key = 4 sólo ha sido consultado una vez, es el elemento que debe removerse.

SystemOut U class test.CacheManager pidiendo elemento: 6
SystemOut U Elemento 6 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Cantidad de elementos en cache ha sido superada. Aplicando LRU
SystemOut U Obteniendo elemento menos requerido: 4
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 5 consultas

Al remover un elemento de la cache, también se incorporaron mensajes para mostrar la estadística y visualizar qué elemento desapareció. En este caso el elemento de key = 4.

SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas

SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 5 consultas
SystemOut U Key 6 ha tenido 1 consultas
SystemOut U class test.CacheManager pidiendo elemento: 7
SystemOut U Elemento 7 no presente en cache. Procediendo a obtener desde otra fuente
SystemOut U Cantidad de elementos en cache ha sido superada. Aplicando LRU
SystemOut U Obteniendo elemento menos requerido: 6
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 5 consultas
SystemOut U Agregando estadísticas para LRU
SystemOut U Key 1 ha tenido 4 consultas
SystemOut U Key 2 ha tenido 5 consultas
SystemOut U Key 3 ha tenido 2 consultas
SystemOut U Key 5 ha tenido 5 consultas
SystemOut U Key 7 ha tenido 1 consultas
SystemOut U Creando visualizadores para departamentos
SystemOut U Direccionando hacia 'departamentos.jsp'

Esto fue el procedimiento básico de navegación a través de catálogos. Con sus variantes en el movimiento en elementos en la cache. A continuación una comparación entre gráficos para una cache de 3 elementos y 10 elementos.

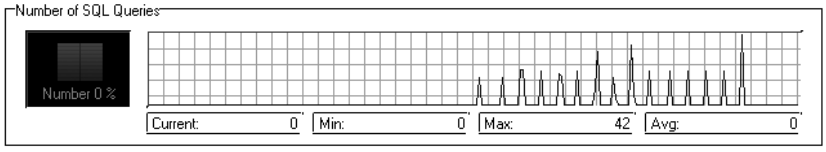

Figura 6.2: Gráfico cualitativo de acceso a la base de datos, para una cache de 3 elementos

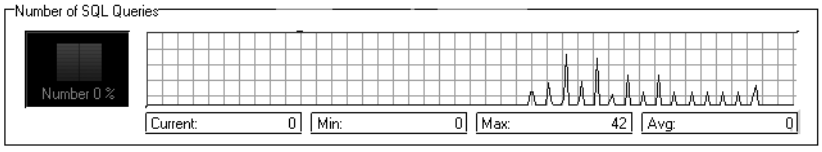

Figura 6.3: Gráfico cualitativo de acceso a la base de datos, para una cache de 10 elementos

Para los gráficos mostrados, la secuencia de navegación fue:

1. Inicio
2. Audio
3. Equipos de Música
4. Bicicletas
5. Bicicletas Aro 12
6. Audio
7. Equipos de Música
8. Audio
9. Equipos de Música
10. Audio
11. Bicicletas
12. Audio
13. Bicicletas
14. Audio
15. Equipos de Música

6.2 Conclusiones

El algoritmo de cache funciona exitosamente. Reduciendo el acceso a la base de datos. Adicionalmente, todo este funcionamiento no fue necesario desarrollarlo en la etapa de testing, pues lo hereda directamente de las clases creadas para el framework. Así como también el flujo entre los administradores. Sólo se requirió implementar aquellas cosas dependientes de la forma en que inicialmente se obtienen los datos, en este caso, implementar los métodos en el Administrador de Cache para obtener los elementos desde una base de datos MySQL. Fue necesario también, implementar las funcionalidades para lo cual un sitio interactúa con el cliente, en este caso, crear un método para acceder a los catálogos denominado particularmente como 'irACatalogo', junto con la creación de los visualizadores para las páginas JSP.

En esta versión del framework creado, se soporta el manejo de un flujo básico de navegación como es revisar catálogos, y la administración de la memoria a través de un algoritmo

de cache tipo LRU. Eventualmente mejoras a este framework será incorporar funcionalidades descritas en los primeros capítulos y los casos de uso CU.02 y CU.03. Otras ideas, como resultado de la etapa de pruebas, es reducir aún más los accesos a la base de datos, generando listas estáticas de tipos de elementos, etiquetas estándares y jerarquía de árboles de catálogos.

De esta manera, el objetivo inicial de crear un framework que permita funcionalidad básica para un sitio de comercio electrónico y que mediante su extensión permita nuevas funcionalidades fue alcanzado y medido.

Apéndice A

Javadoc de clases principales

Package b2c.framework.basicelement

Interface Summary	
B2CAdvancedElementInterface	Interfaz para B2CAdvancedElement.
B2CElementInterface	Interfaz que describe a un elemento básico.

Class Summary	
B2CAdvancedElement	Clase que representa un objeto con descripciones, características, precios, imágenes y subelementos.
B2CContent	Clase que represeta a un objeto con algún contenido.
B2CContentTyped	Clase que representa a un objeto con contenido y tipo
B2CElement	Clase que implementa un elemento básico del framework.
B2CObjectElementary	Clase que representa al objeto más elemental.

b2c.framework.basicelement

Interface B2CAdvancedElementInterface

All Known Implementing Classes:

B2CAdvancedElement

---

public interface **B2CAdvancedElementInterface**

Interfaz para B2CAdvancedElement. Un B2CAdvancedElement es un elemento que a su vez contiene subelementos e imágenes. La interfaz coneitne descripciones sobre funcionalidades para obtener y agregar estas cualidades adicionales del elemento

Version:

1.0

Author:

Javier Villalobos Arancibia

Method Summary	
void	**addComparator**(java.util.Comparator aComparator) Carga un nuevo comparador para ordenar los subelementos de este objeto
void	**addImage**(B2CImage aImage) Enlaza la información de una imagen al objeto
void	**addSubElement**(B2CElement aelement) Agrega un subelemento a este objeto
void	**applyRelationRules**() Aplica determinadas reglas al objeto relaciondas con su entorno
java.util.Comparator	**getComparator**() Obtiene el comparador utilizado para ordenar los subelementos de este objeto
B2CImage	**getImageByType**(B2CType aType) Obtiene una imagen dado su tipo
B2CImage[]	**getImages**() Obtiene todas las imagenes asociadas a este objeto
B2CElement	**getSubElement**(int aKey) Obtiene un subelemento dado su identificador
B2CElement[]	**getSubElements**() Obtiene todos los subelementos contenidos en este objeto ordenados de acuerdo al comparador

Method Detail

getComparator

`public java.util.Comparator` **`getComparator`**`()`

Obtiene el comparador utilizado para ordenar los subelementos de este objeto

Returns:

El comparador actualmente utilizado

---

getSubElement

```
public B2CElement getSubElement(int aKey)
                         throws B2CElementException
```

Obtiene un subelemento dado su identificador

Parameters:

`aKey` - Identificador del subelemento

Returns:

Un subelemento

Throws:

`B2CElementException` - El subelemento no se encuentra

---

getSubElements

```
public B2CElement[] getSubElements()
                             throws B2CElementException
```

Obtiene todos los subelementos contenidos en este objeto ordenados de acuerdo al comparador

Returns:

Arreglo de subelementos ordenados

Throws:

`B2CElementException` - Este objeto no contiene subelementos

---

getImageByType

```
public B2CImage getImageByType(B2CType aType)
                        throws B2CImageException
```

Obtiene una imagen dado su tipo

Parameters:

`aType` - Un tipo de Imagen

Returns:

Objeto con información de la imagen

Throws:

`B2CImageException` - Imagen no encontrada

---

getImages

```
public B2CImage[] getImages()
                  throws B2CImageException
```

Obtiene todas las imagenes asociadas a este objeto

Returns:

Arreglo de imágenes

Throws:

`B2CImageException` - Este objeto no tiene imágenes asociadas

---

addComparator

```
public void addComparator(java.util.Comparator aComparator)
```

Carga un nuevo comparador para ordenar los subelementos de este objeto

Parameters:

`aComparator` - Comparador para ordenamiento

---

addSubElement

```
public void addSubElement(B2CElement aelement)
```

Agrega un subelemento a este objeto

---

addImage

```
public void addImage(B2CImage aImage)
```

Enlaza la información de una imagen al objeto

Parameters:

`aImage` - Objeto con información de la Imagen

---

applyRelationRules

```
public void applyRelationRules()
                    throws B2CAdvancedElementException
```

Aplica determinadas reglas al objeto relaciondas con su entorno

Throws:

`B2CAdvancedElementException` - No se pudieron aplicar las reglas

---

b2c.framework.basicelement

Interface B2CElementInterface

All Known Implementing Classes:

B2CElement

---

public interface **B2CElementInterface**

Interfaz que describe a un elemento básico. Un elemento básico corresponde a un objeto que posee descripciones, características y precios. Es así que esta interfaz describe métodos para agregar y obtener estas cualidades.

Version:

1.0

Author:

Javier Villalobos Arancibia

Method Summary

`void`	**`addCharacteristic`**`(B2CCharacteristic aCharacteristic)` Agrega una característica al elemento
`void`	**`addDescription`**`(B2CDescription aDescription)` Agrega una descripción al elemento
`void`	**`addPrice`**`(B2CPrice aPrice)` Agrega un precio al elemento
`B2CCharacteristic`	**`getCharacteristicByType`**`(B2CType aType)` Obtiene una característica del del elemento referenciado por tipo
`B2CCharacteristic[]`	**`getCharacteristics`**`()` Obtiene todas las características del elemento
`B2CDescription[]`	**`getDescriptions`**`()` Obtiene todas las descripciones del elemento
`B2CDescription`	**`getDescriptionsByType`**`(B2CType aType)` Obtiene las descripciones del elemento referenciadas por tipo
`double`	**`getPriceByType`**`(B2CType aType)` Obtiene un precio de un elemento básico referenciado por tipo
`B2CPrice[]`	**`getPrices`**`()` Obtiene todos los precios del elemento

Method Detail

getPriceByType

```
public double getPriceByType(B2CType aType)
                      throws PriceException
```

Obtiene un precio de un elemento básico referenciado por tipo

Parameters:

`aType` - El tipo de precio

Returns:

El precio del elemento básico

Throws:

`PriceException` - No se pudo obtener el precio para el tipo especificado

---

getPrices

```
public B2CPrice[] getPrices()
                   throws PriceException
```

Obtiene todos los precios del elemento

Returns:

Arreglo de precios

Throws:

`PriceException` - No pudo obtenerse los precios del elemento

---

getDescriptionsByType

```
public B2CDescription getDescriptionsByType(B2CType aType)
                                     throws DescriptionException
```

Obtiene las descripciones del elemento referenciadas por tipo

Parameters:

`aType` - El tipo de descripción

Returns:

Una descripción del elemento

Throws:

`DescriptionException` - No se pudo obtener la descripción para el tipo especificado

---

getDescriptions

```
public B2CDescription[] getDescriptions()
                                  throws DescriptionException
```

Obtiene todas las descripciones del elemento

Returns:

Arreglo de descripciones

Throws:

`DescriptionException` - No pudo obtenerse las descripciones del elemento

---

getCharacteristicByType

```
public B2CCharacteristic getCharacteristicByType(B2CType aType)
                                           throws CharacteristicException
```

Obtiene una característica del del elemento referenciado por tipo

Parameters:

`aType` - El tipo de característica

Returns:

Una característica del elemento

Throws:

`CharacteristicException` - No se pudo obtener la característica para el tipo especificado

---

getCharacteristics

```
public B2CCharacteristic[] getCharacteristics()
                                       throws CharacteristicException
```

Obtiene todas las características del elemento

Returns:

Arreglo de características

Throws:

`CharacteristicException` - No se pudo obtener las características del elemento

---

addPrice

```
public void addPrice(B2CPrice aPrice)
```

Agrega un precio al elemento

Parameters:

`aPrice` - Un precio cualquiera

---

addDescription

`public void` **`addDescription`**`(B2CDescription aDescription)`

Agrega una descripción al elemento

Parameters:

`aDescription` - Un precio cualquiera

---

addCharacteristic

`public void` **`addCharacteristic`**`(B2CCharacteristic aCharacteristic)`

Agrega una característica al elemento

Parameters:

`aCharacteristic` - Una característica cualquiera

---

b2c.framework.basicelement

Class B2CAdvancedElement

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CElement
          └b2c.framework.basicelement.B2CAdvancedElement
```

All Implemented Interfaces:

B2CAdvancedElementInterface, B2CElementInterface, java.lang.Cloneable

Direct Known Subclasses:

B2CArticle, B2CPromotion

---

public class **B2CAdvancedElement**
extends B2CElement
implements B2CAdvancedElementInterface

Clase que representa un objeto con descripciones, características, precios, imágenes y subelementos. Posee un comparador propio para obtener los subelementos de forma ordenada. Este comparador puede establecer también.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

protected java.util.Comparator	**comparator** Comparador para ordenar subelementos.
protected java.util.TreeMap	**images** Mapa de imágenes
protected java.util.TreeMap	**subElements** Mapa de subelementos

Fields inherited from class b2c.framework.basicelement.B2CElement

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CAdvancedElement(int aKey) Único constructor permitido, donde se debe especificar un identificador.	

Method Summary

void	**addComparator**(java.util.Comparator aComparator) Carga un nuevo comparador para ordenar los subelementos de este objeto
void	**addImage**(B2CImage aImage) Enlaza la información de una imagen al objeto
void	**addSubElement**(B2CElement aelement) Agrega un subelemento a este objeto
void	**applyRelationRules**() Aplica determinadas reglas al objeto relaciondas con su entorno
java.lang.Object	**clone**() Crea y devuelve una copia de este objeto.
java.util.Comparator	**getComparator**() Obtiene el comparador utilizado para ordenar los subelementos de este objeto
B2CImage	**getImageByType**(B2CType aType) Obtiene una imagen dado su tipo

B2CImage[]	**getImages**() Obtiene todas las imagenes asociadas a este objeto
B2CElement	**getSubElement**(int aKey) Obtiene un subelemento dado su identificador
B2CElement[]	**getSubElements**() Obtiene todos los subelementos contenidos en este objeto ordenados de acuerdo al comparador

Methods inherited from class b2c.framework.basicelement.B2CElement

addCharacteristic, addDescription, addPrice, getCharacteristicByType, getCharacteristics, getDescriptions, getDescriptionsByType, getPriceByType, getPrices

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

comparator

```
protected java.util.Comparator comparator
```

Comparador para ordenar subelementos. Por defecto utiliza un comparador de identificadores

---

subElements

```
protected java.util.TreeMap subElements
```

Mapa de subelementos

---

images

```
protected java.util.TreeMap images
```

Mapa de imágenes

Constructor Detail

B2CAdvancedElement

```
public B2CAdvancedElement(int aKey)
```

Único constructor permitido, donde se debe especificar un identificador.

Method Detail

getComparator

```
public java.util.Comparator getComparator()
```

Description copied from interface: `B2CAdvancedElementInterface`

Obtiene el comparador utilizado para ordenar los subelementos de este objeto

Specified by:

`getComparator` in interface `B2CAdvancedElementInterface`

Returns:

El comparador actualmente utilizado

See Also:

`B2CAdvancedElementInterface.getComparator()`

---

getSubElement

```
public B2CElement getSubElement(int aKey)
                         throws B2CElementException
```

Description copied from interface: `B2CAdvancedElementInterface`

Obtiene un subelemento dado su identificador

Specified by:

`getSubElement` in interface `B2CAdvancedElementInterface`

Parameters:

`aKey` - Identificador del subelemento

Returns:

Un subelemento

Throws:

`B2CElementException` - El subelemento no se encuentra

See Also:

`B2CAdvancedElementInterface.getSubElement(int)`

---

getSubElements

```
public B2CElement[] getSubElements()
                            throws B2CElementException
```

Description copied from interface: B2CAdvancedElementInterface
Obtiene todos los subelementos contenidos en este objeto ordenados de acuerdo al comparador

Specified by:
getSubElements in interface B2CAdvancedElementInterface

Returns:
Arreglo de subelementos ordenados

Throws:
B2CElementException - Este objeto no contiene subelementos

See Also:
B2CAdvancedElementInterface.getSubElements()

---

getImageByType

```
public B2CImage getImageByType(B2CType aType)
                        throws B2CImageException
```

Description copied from interface: B2CAdvancedElementInterface
Obtiene una imagen dado su tipo

Specified by:
getImageByType in interface B2CAdvancedElementInterface

Parameters:
aType - Un tipo de Imagen

Returns:
Objeto con información de la imagen

Throws:
B2CImageException - Imagen no encontrada

See Also:
B2CAdvancedElementInterface.getImageByType(B2CType)

---

getImages

```
public B2CImage[] getImages()
                  throws B2CImageException
```

Description copied from interface: B2CAdvancedElementInterface
Obtiene todas las imagenes asociadas a este objeto
Specified by:
getImages in interface B2CAdvancedElementInterface
Returns:
Arreglo de imágenes
Throws:
B2CImageException - Este objeto no tiene imágenes asociadas
See Also:
B2CAdvancedElementInterface.getImages()

---

addComparator

```
public void addComparator(java.util.Comparator aComparator)
```

Description copied from interface: B2CAdvancedElementInterface
Carga un nuevo comparador para ordenar los subelementos de este objeto
Specified by:
addComparator in interface B2CAdvancedElementInterface
Parameters:
aComparator - Comparador para ordenamiento
See Also:
B2CAdvancedElementInterface.addComparator(Comparator)

---

addSubElement

```
public void addSubElement(B2CElement aelement)
```

Description copied from interface: B2CAdvancedElementInterface
Agrega un subelemento a este objeto
Specified by:
addSubElement in interface B2CAdvancedElementInterface
See Also:
B2CAdvancedElementInterface.addSubElement(B2CElement)

---

addImage

```
public void addImage(B2CImage aImage)
```

Description copied from interface: B2CAdvancedElementInterface
Enlaza la información de una imagen al objeto
Specified by:
addImage in interface B2CAdvancedElementInterface
Parameters:
aImage - Objeto con información de la Imagen
See Also:
B2CAdvancedElementInterface.addImage(B2CImage)

---

applyRelationRules

```
public void applyRelationRules()
                        throws B2CAdvancedElementException
```

Description copied from interface: B2CAdvancedElementInterface
Aplica determinadas reglas al objeto relaciondas con su entorno
Specified by:
applyRelationRules in interface B2CAdvancedElementInterface
Throws:
B2CAdvancedElementException - No se pudieron aplicar las reglas
See Also:
B2CAdvancedElementInterface.applyRelationRules()

---

clone

```
public java.lang.Object clone()
                       throws java.lang.CloneNotSupportedException
```

Crea y devuelve una copia de este objeto.
Overrides:
clone in class B2CElement
Returns:
Un clon de esta instancia
Throws:
java.lang.CloneNotSupportedException - Si el objeto de la clase no soporta la interfaz Cloneable. Subclases que sobreescriben el método clone() pueden también arrojar esta excepción para indicar que esa instancia no puede ser clonada

---

b2c.framework.basicelement

Class B2CContent

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
```

All Implemented Interfaces:

java.lang.Cloneable

Direct Known Subclasses:

B2CContentTyped, B2CType

---

public class **B2CContent**
extends B2CObjectElementary

Clase que represeta a un objeto con algún contenido. El contenido corresponde a una descripción de tipo String. Implementa Cloneable para poder obtener copias de si mismo

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

private java.lang.String	**content** Descripción o contenido de este elemento

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CContent()	

Method Summary

java.lang.Object	**clone**() Crea y devuelve una copia de este objeto.
java.lang.String	**getContent**() Obtiene el contenido

void	**setContent**(java.lang.String aContent) Asigna un contenido

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

content

```
private java.lang.String content
```

Descripción o contenido de este elemento

Constructor Detail

B2CContent

```
public B2CContent()
```

Method Detail

getContent

```
public java.lang.String getContent()
```

Obtiene el contenido

Returns:

un contenido

---

setContent

```
public void setContent(java.lang.String aContent)
```

Asigna un contenido

---

clone

```
public java.lang.Object clone()
                       throws java.lang.CloneNotSupportedException
```

Crea y devuelve una copia de este objeto.

Overrides:

clone in class B2CObjectElementary

Returns:

Un clon de esta instancia

Throws:

`java.lang.CloneNotSupportedException` - Si el objeto de la clase no soporta la interfaz Cloneable. Subclases que sobreescriben el método clone() pueden también arrojar esta excepción para indicar que esa instancia no puede ser clonada

---

b2c.framework.basicelement

Class B2CContentTyped

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.basicelement.B2CContentTyped
```

All Implemented Interfaces:

java.lang.Cloneable

Direct Known Subclasses:

B2CCharacteristic, B2CDescription, B2CImage, B2CPrice

---

public class **B2CContentTyped**
extends B2CContent

Clase que representa a un objeto con contenido y tipo

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

private B2CType	**type** Objeto contenedor de la información sobre el tipo

Fields inherited from class b2c.framework.basicelement.B2CContent

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CContentTyped()	

Method Summary

B2CType	**getType**() Obtiene el tipo
void	**setType**(B2CType aType) Asigna un tipo

Methods inherited from class b2c.framework.basicelement.B2CContent

clone, getContent, setContent

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

type

```
private B2CType type
```

Objeto contenedor de la información sobre el tipo

Constructor Detail

B2CContentTyped

```
public B2CContentTyped()
```

Method Detail

getType

`public B2CType getType()`

Obtiene el tipo

Returns:

El objeto con el tipo

---

setType

`public void setType(B2CType aType)`

Asigna un tipo

---

b2c.framework.basicelement

Class B2CElement

```
java.lang.Object
  └ b2c.framework.basicelement.B2CObjectElementary
      └ b2c.framework.basicelement.B2CElement
```

All Implemented Interfaces:

B2CElementInterface, java.lang.Cloneable

Direct Known Subclasses:

B2CAdvancedElement, B2CProduct

---

public class **B2CElement**
extends B2CObjectElementary
implements B2CElementInterface

Clase que implementa un elemento básico del framework. Un elemento básico posee descripciones, características y precios. Posee métodos para agregar y obtener estas cualidades. Además implementa Cloneable para generar copias de si mismo.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

private java.util.TreeMap	**characteristic** Mapa para características
private java.util.TreeMap	**description** Mapa para descripciones
private java.util.TreeMap	**price** Mapa para precios

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CElement(int aKey) Único constructor.

Method Summary

void	**addCharacteristic**(B2CCharacteristic aCharacteristic) Agrega una característica al elemento
void	**addDescription**(B2CDescription aDescription) Agrega una descripción al elemento
void	**addPrice**(B2CPrice aPrice) Agrega un precio al elemento
java.lang.Object	**clone**() Crea y devuelve una copia de este objeto.
B2CCharacteristic	**getCharacteristicByType**(B2CType aType) Obtiene una característica del del elemento referenciado por tipo
B2CCharacteristic[]	**getCharacteristics**() Obtiene todas las características del elemento
B2CDescription[]	**getDescriptions**() Obtiene todas las descripciones del elemento
B2CDescription	**getDescriptionsByType**(B2CType aType) Obtiene las descripciones del elemento referenciadas por tipo
double	**getPriceByType**(B2CType aType) Obtiene un precio de un elemento básico referenciado por tipo
B2CPrice[]	**getPrices**() Obtiene todos los precios del elemento

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

price

```
private java.util.TreeMap price
```

Mapa para precios

---

description

```
private java.util.TreeMap description
```

Mapa para descripciones

---

characteristic

```
private java.util.TreeMap characteristic
```

Mapa para características

Constructor Detail

B2CElement

```
public B2CElement(int aKey)
```

Único constructor. Se debe establecer un identificador

Method Detail

getPriceByType

```
public double getPriceByType(B2CType aType)
                      throws PriceException
```

Description copied from interface: B2CElementInterface
Obtiene un precio de un elemento básico referenciado por tipo
Specified by:
getPriceByType in interface B2CElementInterface
Parameters:
aType - El tipo de precio
Returns:
El precio del elemento básico
Throws:
PriceException - No se pudo obtener el precio para el tipo especificado
See Also:
B2CElementInterface.getPriceByType(B2CType)

---

getPrices

```
public B2CPrice[] getPrices()
                    throws PriceException
```

Description copied from interface: B2CElementInterface
Obtiene todos los precios del elemento
Specified by:
getPrices in interface B2CElementInterface
Returns:
Arreglo de precios
Throws:
PriceException - No pudo obtenerse los precios del elemento
See Also:
B2CElementInterface.getPrices()

---

getDescriptionsByType

```
public B2CDescription getDescriptionsByType(B2CType aType)
                                     throws DescriptionException
```

Description copied from interface: B2CElementInterface
Obtiene las descripciones del elemento referenciadas por tipo
Specified by:
getDescriptionsByType in interface B2CElementInterface
Parameters:
aType - El tipo de descripción
Returns:
Una descripción del elemento
Throws:
DescriptionException - No se pudo obtener la descripción para el tipo especificado
See Also:
B2CElementInterface.getDescriptionsByType(B2CType)

---

getDescriptions

```
public B2CDescription[] getDescriptions()
                                  throws DescriptionException
```

Description copied from interface: B2CElementInterface
Obtiene todas las descripciones del elemento
Specified by:
getDescriptions in interface B2CElementInterface
Returns:
Arreglo de descripciones
Throws:
DescriptionException - No pudo obtenerse las descripciones del elemento
See Also:
B2CElementInterface.getDescriptions()

---

getCharacteristicByType

```
public B2CCharacteristic getCharacteristicByType(B2CType aType)
                                          throws CharacteristicException
```

Description copied from interface: B2CElementInterface
Obtiene una característica del del elemento referenciado por tipo
Specified by:
getCharacteristicByType in interface B2CElementInterface
Parameters:
aType - El tipo de característica
Returns:
Una característica del elemento
Throws:
CharacteristicException - No se pudo obtener la característica para el tipo especificado
See Also:
B2CElementInterface.getCharacteristicByType(B2CType)

---

getCharacteristics

```
public B2CCharacteristic[] getCharacteristics()
                                       throws CharacteristicException
```

Description copied from interface: B2CElementInterface
Obtiene todas las características del elemento
Specified by:
getCharacteristics in interface B2CElementInterface
Returns:
Arreglo de características
Throws:
CharacteristicException - No se pudo obtener las características del elemento
See Also:
B2CElementInterface.getCharacteristics()

---

addPrice

```
public void addPrice(B2CPrice aPrice)
```

Description copied from interface: B2CElementInterface

Agrega un precio al elemento

Specified by:

addPrice in interface B2CElementInterface

Parameters:

aPrice - Un precio cualquiera

See Also:

B2CElementInterface.addPrice(B2CPrice)

---

addDescription

```
public void addDescription(B2CDescription aDescription)
```

Description copied from interface: B2CElementInterface

Agrega una descripción al elemento

Specified by:

addDescription in interface B2CElementInterface

Parameters:

aDescription - Un precio cualquiera

See Also:

B2CElementInterface.addDescription(B2CDescription)

---

addCharacteristic

```
public void addCharacteristic(B2CCharacteristic aCharacteristic)
```

Description copied from interface: B2CElementInterface

Agrega una característica al elemento

Specified by:

addCharacteristic in interface B2CElementInterface

Parameters:

aCharacteristic - Una característica cualquiera

See Also:

B2CElementInterface.addCharacteristic(B2CCharacteristic)

---

clone

```
public java.lang.Object clone()
                        throws java.lang.CloneNotSupportedException
```

Crea y devuelve una copia de este objeto.

Overrides:

`clone` in class `B2CObjectElementary`

Returns:

Un clon de esta instancia

Throws:

`java.lang.CloneNotSupportedException` - Si el objeto de la clase no soporta la interfaz Cloneable. Subclases que sobreescriben el método clone() pueden también arrojar esta excepción para indicar que esa instancia no puede ser clonada

---

b2c.framework.basicelement

Class B2CObjectElementary

```
java.lang.Object
  └ b2c.framework.basicelement.B2CObjectElementary
```

All Implemented Interfaces:

java.lang.Cloneable

Direct Known Subclasses:

B2CContent, B2CElement, B2CLRUObject

---

public class **B2CObjectElementary**
extends java.lang.Object
implements java.lang.Cloneable

Clase que representa al objeto más elemental. Posee un identificador o 'key' que lo individualiza de otros objetos o instancias. Además implementa Cloneable para permitir obtener copias y no ser afectada la instancia, en caso de hacer modificaciones temporales.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

int	**key** Identificador del objeto o instancia

Constructor Summary

B2CObjectElementary()	

Method Summary

java.lang.Object	**clone**() Crea y devuelve una copia de este objeto.

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

key

```
public int key
```

Identificador del objeto o instancia

Constructor Detail

B2CObjectElementary

```
public B2CObjectElementary()
```

Method Detail

clone

```
public java.lang.Object clone()
                    throws java.lang.CloneNotSupportedException
```

Crea y devuelve una copia de este objeto.

Returns:

Un clon de esta instancia

Throws:

`java.lang.CloneNotSupportedException` - Si el objeto de la clase no soporta la interfaz Cloneable. Subclases que sobreescriben el método clone() pueden también arrojar esta excepción para indicar que esa instancia no puede ser clonada

---

Package b2c.framework.catalogue

Class Summary	
B2CCatalogue	Clase que representa a un catálogo

b2c.framework.catalogue

Class B2CCatalogue

```
java.lang.Object
  └ b2c.framework.basicelement.B2CObjectElementary
      └ b2c.framework.basicelement.B2CElement
          └ b2c.framework.basicelement.B2CAdvancedElement
              └ b2c.framework.article.B2CPromotion
                  └ b2c.framework.catalogue.B2CCatalogue
```

All Implemented Interfaces:

B2CAdvancedElementInterface, B2CElementInterface, java.lang.Cloneable

---

public class **B2CCatalogue**
extends B2CPromotion

Clase que representa a un catálogo

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary	
private java.lang.Class	**articleClass** Propiedad de Artículo para discrimnar artículos de promociones
private java.util.Comparator	**catalogueComparator** Comparador para catálogos.
private java.lang.Class	**promotionClass** Propiedad de Promoción para discriminar promociones de artículos
private java.util.TreeMap	**subcatalogues** Mapa de subcatálogos

Fields inherited from class b2c.framework.basicelement.B2CAdvancedElement
comparator, images, subElements
Fields inherited from class b2c.framework.basicelement.B2CElement
Fields inherited from class b2c.framework.basicelement.B2CObjectElementary
key

Constructor Summary

B2CCatalogue(int aKey) Constructor único para catálogos.	

Method Summary

void	**addArticle**(B2CArticle article)
void	**addCatalogueComparator**(java.util.Comparator aComparator) Establece un nuevo comparador para catálogos.
void	**addPromotion**(B2CPromotion aPromotion)
void	**addSubCatalogue**(B2CCatalogue catalogue)
java.lang.Object	**clone**() Crea y devuelve una copia de este objeto.
B2CArticle	**getArticle**(int aKey)
B2CArticle[]	**getArticles**()
java.util.Comparator	**getCatalogueComparator**() Obtiene el comparador utilizado para catálogos
B2CPromotion	**getPromotion**(int aKey)
B2CPromotion[]	**getPromotions**()
B2CCatalogue	**getSubCatalogue**(int aKey)
B2CCatalogue[]	**getSubCatalogues**()

Methods inherited from class b2c.framework.basicelement.B2CAdvancedElement
addComparator, addImage, addSubElement, applyRelationRules, getComparator, getImageByType, getImages, getSubElement, getSubElements
Methods inherited from class b2c.framework.basicelement.B2CElement
addCharacteristic, addDescription, addPrice, getCharacteristicByType, getCharacteristics, getDescriptions, getDescriptionsByType, getPriceByType, getPrices
Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

promotionClass

```
private java.lang.Class promotionClass
```

Propiedad de Promoción para discriminar promociones de artículos

---

articleClass

```
private java.lang.Class articleClass
```

Propiedad de Artículo para discrimnar artículos de promociones

---

subcatalogues

```
private java.util.TreeMap subcatalogues
```

Mapa de subcatálogos

---

catalogueComparator

```
private java.util.Comparator catalogueComparator
```

Comparador para catálogos. Por defecto utiliza un comparador en base a keys de catálogo

Constructor Detail

B2CCatalogue

```
public B2CCatalogue(int aKey)
```

Constructor único para catálogos. Se debe establecer un identificador

Method Detail

addCatalogueComparator

```
public void addCatalogueComparator(java.util.Comparator aComparator)
```

Establece un nuevo comparador para catálogos. El comparador debe comparar objetos del tipo B2CCatalogue

Parameters:

`aComparator` - Un comparador de objetos B2CCatalogue

---

getCatalogueComparator

```
public java.util.Comparator getCatalogueComparator()
```

Obtiene el comparador utilizado para catálogos

Returns:

El comparador de catálogos

---

addPromotion

```
public void addPromotion(B2CPromotion aPromotion)
```

---

getPromotion

```
public B2CPromotion getPromotion(int aKey)
                          throws B2CPromotionException
```

Throws:

`B2CPromotionException`

---

getPromotions

```
public B2CPromotion[] getPromotions()
                              throws B2CPromotionException
```

Throws:
B2CPromotionException

---

addArticle

```
public void addArticle(B2CArticle article)
```

Overrides:
addArticle in class B2CPromotion

---

getArticle

```
public B2CArticle getArticle(int aKey)
                      throws B2CArticleException
```

Overrides:
getArticle in class B2CPromotion
Throws:
B2CArticleException

---

getArticles

```
public B2CArticle[] getArticles()
                         throws B2CArticleException
```

Overrides:
getArticles in class B2CPromotion
Throws:
B2CArticleException

---

addSubCatalogue

```
public void addSubCatalogue(B2CCatalogue catalogue)
```

---

getSubCatalogue

```
public B2CCatalogue getSubCatalogue(int aKey)
                             throws B2CCatalogueException
```

Throws:
B2CCatalogueException

---

getSubCatalogues

```
public B2CCatalogue[] getSubCatalogues()
                               throws B2CCatalogueException
```

Throws:
B2CCatalogueException

---

clone

```
public java.lang.Object clone()
                        throws java.lang.CloneNotSupportedException
```

Description copied from class: B2CAdvancedElement
Crea y devuelve una copia de este objeto.
Overrides:
clone in class B2CAdvancedElement
Returns:
Un clon de esta instancia
Throws:
java.lang.CloneNotSupportedException - Si el objeto de la clase no soporta la interfaz Cloneable. Subclases que sobreescriben el método clone() pueden también arrojar esta excepción para indicar que esa instancia no puede ser clonada

---

Package b2c.framework.article

Interface Summary	
B2CArticleManagerInterface	Interfaz para el Administrador de Artículos.

Class Summary	
B2CArticle	Clase que representa a un artículo
B2CArticleManager	Administrador de Artículos.
B2CProduct	Clase que representa a un producto
B2CPromotion	Calse que representa a una promoción

b2c.framework.article

Interface B2CArticleManagerInterface

All Known Implementing Classes:
B2CArticleManager

---

public interface **B2CArticleManagerInterface**

Interfaz para el Administrador de Artículos. Establece métodos y reglas para el manejo de catálogos, artículos, promociones y productos. Adicionalmente establece métodos para limpiar la cache de estos elementos

Version:
1.0
Author:
Javier Villalobos Arancibia

Method Summary	
void	**applyRuleArticles**(B2CArticle article) Aplica reglas a un artículo.
void	**applyRuleCatalogues**(B2CCatalogue catalogue) Aplica reglas a un catálogo.
void	**applyRuleProducts**(B2CProduct product) Aplica reglas a un producto.

void	**applyRulePromotions**(B2CPromotion promotion) Aplica reglas a una promoción.
void	**clearArticleCache**() Limpia la cache de Artículos
void	**clearCatalogueCache**() Limpia la cache de Catálogos
void	**clearProductCache**() Limpia la cache de Productos
void	**clearPromotionCache**() Limpia la cache de Promociones
B2CArticle	**getArticle**(int key) Obtiene un artículo.
B2CCatalogue	**getCatalogue**(int key) Obtiene un catálogo.
B2CProduct	**getProduct**(int key) Obtiene un producto.
B2CPromotion	**getPromotion**(int key) Obtiene una promoción.

Method Detail

applyRuleArticles

```
public void applyRuleArticles(B2CArticle article)
                       throws B2CArticleException
```

Aplica reglas a un artículo. Por defecto, las reglas son generar descripciones vacías para el artículo

Parameters:

`article` - Articulo a ser modificado según las reglas

Throws:

`B2CArticleException` - Excepción al manipular el artículo

---

applyRuleProducts

```
public void applyRuleProducts(B2CProduct product)
                       throws B2CProductException
```

Aplica reglas a un producto. Por defecto, las reglas son generar descripciones vacías para el producto

Parameters:

`product` - Producto a ser modificado según reglas

Throws:

`B2CProductException` - Excepción al manipular el producto

---

applyRuleCatalogues

```
public void applyRuleCatalogues(B2CCatalogue catalogue)
                         throws B2CCatalogueException
```

Aplica reglas a un catálogo. Por defecto, las reglas son generar descripciones vacías para el catálogo

Parameters:

`catalogue` - Catálogo a ser modificado según las reglas

Throws:

`B2CCatalogueException` - Excepción al manupular el producto

---

applyRulePromotions

```
public void applyRulePromotions(B2CPromotion promotion)
                         throws B2CPromotionException
```

Aplica reglas a una promoción. Por defecto, las reglas son generar descripciones vacías para la promoción

Parameters:

`promotion` - Promoción a ser modificada según las reglas

Throws:

`B2CPromotionException` - Excepción al manipular la promoción

---

getArticle

```
public B2CArticle getArticle(int key)
                     throws B2CArticleException,
                            B2CCacheException
```

Obtiene un artículo. El procedimiento por defecto es buscar el artículo en el administrador de Cache.

Parameters:

`key` - Identificador de artículo

Returns:

Un artículo

Throws:

`B2CArticleException` - Artículo no existe en la cache

`B2CCacheException` - Excepción al acceder a la cache´

---

getProduct

```
public B2CProduct getProduct(int key)
                     throws B2CProductException,
                            B2CCacheException
```

Obtiene un producto. El procedimiento por defecto es buscar el producto en el administrador de Cache.

Parameters:

`key` - Identificador de producto

Returns:

Un producto

Throws:

`B2CProductException` - Producto no existe en la cache

`B2CCacheException` - Excepción al acceder a la cache´

---

getCatalogue

```
public B2CCatalogue getCatalogue(int key)
                          throws B2CCatalogueException,
                                 B2CCacheException
```

Obtiene un catálogo. El procedimiento por defecto es buscar el catálogo en el administrador de Cache.

Parameters:

`key` - Identificador de catálogo

Returns:

Un catálogo

Throws:

`B2CCatalogueException` - Catálogo no existe en la cache

`B2CCacheException` - Excepción al acceder a la cache´

---

getPromotion

```
public B2CPromotion getPromotion(int key)
                          throws B2CPromotionException,
                                 B2CCacheException
```

Obtiene una promoción. El procedimiento por defecto es buscar la promoción en el administrador de Cache.

Parameters:

`key` - Identificador de promoción

Returns:

Una promoción

Throws:

`B2CPromotionException` - Promoción no existe en la cache

`B2CCacheException` - Excepción al acceder a la cache´

---

clearArticleCache

```
public void clearArticleCache()
                        throws B2CCacheException
```

Limpia la cache de Artículos

Throws:

`B2CCacheException` - Excepción al acceder a la cache

---

clearProductCache

```
public void clearProductCache()
                        throws B2CCacheException
```

Limpia la cache de Productos

Throws:

B2CCacheException - Excepción al acceder a la cache

---

clearCatalogueCache

```
public void clearCatalogueCache()
                          throws B2CCacheException
```

Limpia la cache de Catálogos

Throws:

B2CCacheException - Excepción al acceder a la cache

---

clearPromotionCache

```
public void clearPromotionCache()
                          throws B2CCacheException
```

Limpia la cache de Promociones

Throws:

B2CCacheException - Excepción al acceder a la cache

---

b2c.framework.article

Class B2CArticle

```
java.lang.Object
  └ b2c.framework.basicelement.B2CObjectElementary
      └ b2c.framework.basicelement.B2CElement
          └ b2c.framework.basicelement.B2CAdvancedElement
              └ b2c.framework.article.B2CArticle
```

All Implemented Interfaces:

B2CAdvancedElementInterface, B2CElementInterface, java.lang.Cloneable

---

public class **B2CArticle**
extends B2CAdvancedElement

Clase que representa a un artículo

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CAdvancedElement

`comparator`, `images`, `subElements`

Fields inherited from class b2c.framework.basicelement.B2CElement

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

`key`

Constructor Summary

`B2CArticle``(int aKey)` Constructor único.	

Method Summary

`void`	**`addProduct`**`(B2CProduct aProduct)` Agrega un producto al artículo
`B2CProduct`	**`getProduct`**`(int aKey)` Obtiene un producto de este artículo, dado su identificador

B2CProduct[]	**getProducts**() Obtiene todos los productos contenidos en este artículo, ordenados según el comparador

Methods inherited from class b2c.framework.basicelement.B2CAdvancedElement

addComparator, addImage, addSubElement, applyRelationRules, clone, getComparator, getImageByType, getImages, getSubElement, getSubElements

Methods inherited from class b2c.framework.basicelement.B2CElement

addCharacteristic, addDescription, addPrice, getCharacteristicByType, getCharacteristics, getDescriptions, getDescriptionsByType, getPriceByType, getPrices

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

B2CArticle

```
public B2CArticle(int aKey)
```

Constructor único. Debe especificar un identificador

Method Detail

getProduct

```
public B2CProduct getProduct(int aKey)
                      throws B2CProductException
```

Obtiene un producto de este artículo, dado su identificador

Parameters:

aKey - Identificador del producto

Returns:

Un producto

Throws:

B2CProductException - No se encontró el producto

---

getProducts

```
public B2CProduct[] getProducts()
                           throws B2CProductException
```

Obtiene todos los productos contenidos en este artículo, ordenados según el comparador

Returns:

Arreglo de productos

Throws:

`B2CProductException` - No hay productos

---

addProduct

```
public void addProduct(B2CProduct aProduct)
```

Agrega un producto al artículo

Parameters:

`aProduct` - Un producto

---

b2c.framework.article

Class B2CArticleManager

```
java.lang.Object
  └ b2c.framework.article.B2CArticleManager
```

All Implemented Interfaces:

B2CArticleManagerInterface

---

public class **B2CArticleManager**
extends java.lang.Object
implements B2CArticleManagerInterface

Administrador de Artículos. Contiene la implementación por defecto de reglas aplicacbles a catálogos, promociones, artículos y productos. Estas reglas generan descripciones, características, precios e imágenes vacías para los elementos. El tipo empleado para estas cualidades posee una key = -1, y descripción "default", por lo tanto, estas cualidades vacías pueden ser referenciadas por este tipo "default".

Este administrador contiene, además, la implementación básica para obtener catálogos,

promociones, artículos y productos. El procedimiento utilizado es la búsqueda de estos elementos en la cache. Para ello, utiliza la referencia establecida hacia el Administrador de Cache.

También es posible acceder a la cache de los cuatro elementos básicos ya nombrados, y limpiar su contenido. Esto es útil, pues la información puede cambiar y es necesario recrear su contenido.

Version:
1.0
Author:
Javier Villalobos Arancibia
See Also:
B2CArticleManagerInterface

Field Summary

private B2CCacheManager	**cacheManager** Referencia hacia el Administrador de Cache

Constructor Summary

B2CArticleManager(B2CCacheManager cacheManager) Constructor único.

Method Summary

void	**applyRuleArticles**(B2CArticle article) Aplica reglas a un artículo.
void	**applyRuleCatalogues**(B2CCatalogue catalogue) Aplica reglas a un catálogo.
void	**applyRuleProducts**(B2CProduct product) Aplica reglas a un producto.
void	**applyRulePromotions**(B2CPromotion promotion) Aplica reglas a una promoción.
void	**clearArticleCache**() Limpia la cache de Artículos
void	**clearCatalogueCache**() Limpia la cache de Catálogos
void	**clearProductCache**() Limpia la cache de Productos
void	**clearPromotionCache**()

	Limpia la cache de Promociones
B2CArticle	**getArticle**(int key) Obtiene un artículo.
B2CCacheManager	**getCacheManager**() Obtiene la referencia hacia el Administrador de Cache
B2CCatalogue	**getCatalogue**(int key) Obtiene un catálogo.
B2CProduct	**getProduct**(int key) Obtiene un producto.
B2CPromotion	**getPromotion**(int key) Obtiene una promoción.
void	**setCacheManager**(B2CCacheManager cacheManager) Establece la referencia hacia el Administrador de Cache
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

cacheManager

```
private B2CCacheManager cacheManager
```

Referencia hacia el Administrador de Cache

Constructor Detail

B2CArticleManager

```
public B2CArticleManager(B2CCacheManager cacheManager)
```

Constructor único. Crea una nueva intancia del Administrador de Artículos, indicando el lazo hacia el Administrador de Cache

Parameters:

cacheManager - Referencia hacia el Administrador de Cache

Method Detail

applyRuleArticles

```
public void applyRuleArticles(B2CArticle article)
                       throws B2CArticleException
```

Description copied from interface: B2CArticleManagerInterface
Aplica reglas a un artículo. Por defecto, las reglas son generar descripciones vacías para el artículo

Specified by:
applyRuleArticles in interface B2CArticleManagerInterface

Parameters:
article - Articulo a ser modificado según las reglas

Throws:
B2CArticleException - Excepción al manipular el artículo

See Also:
B2CArticleManagerInterface.applyRuleArticles(B2CArticle)

---

applyRuleProducts

```
public void applyRuleProducts(B2CProduct product)
                       throws B2CProductException
```

Description copied from interface: B2CArticleManagerInterface
Aplica reglas a un producto. Por defecto, las reglas son generar descripciones vacías para el producto

Specified by:
applyRuleProducts in interface B2CArticleManagerInterface

Parameters:
product - Producto a ser modificado según reglas

Throws:
B2CProductException - Excepción al manipular el producto

See Also:
B2CArticleManagerInterface.applyRuleProducts(B2CProduct)

---

applyRuleCatalogues

```
public void applyRuleCatalogues(B2CCatalogue catalogue)
                         throws B2CCatalogueException
```

Description copied from interface: B2CArticleManagerInterface
Aplica reglas a un catálogo. Por defecto, las reglas son generar descripciones vacías para el catálogo

Specified by:
applyRuleCatalogues in interface B2CArticleManagerInterface

Parameters:
catalogue - Catálogo a ser modificado según las reglas

Throws:
B2CCatalogueException - Excepción al manupular el producto

See Also:
B2CArticleManagerInterface.applyRuleCatalogues(B2CCatalogue)

---

applyRulePromotions

```
public void applyRulePromotions(B2CPromotion promotion)
                         throws B2CPromotionException
```

Description copied from interface: B2CArticleManagerInterface
Aplica reglas a una promoción. Por defecto, las reglas son generar descripciones vacías para la promoción

Specified by:
applyRulePromotions in interface B2CArticleManagerInterface

Parameters:
promotion - Promoción a ser modificada según las reglas

Throws:
B2CPromotionException - Excepción al manipular la promoción

See Also:
B2CArticleManagerInterface.applyRulePromotions(B2CPromotion)

---

getArticle

```
public B2CArticle getArticle(int key)
                      throws B2CArticleException,
                             B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Obtiene un artículo. El procedimiento por defecto es buscar el artículo en el administrador de Cache.

Specified by:
getArticle in interface B2CArticleManagerInterface

Parameters:
key - Identificador de artículo

Returns:
Un artículo

Throws:
B2CCacheException - Excepción al acceder a la cache´
B2CArticleException - Artículo no existe en la cache

See Also:
B2CArticleManagerInterface.getArticle(int)

---

getProduct

```
public B2CProduct getProduct(int key)
                      throws B2CProductException,
                             B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Obtiene un producto. El procedimiento por defecto es buscar el producto en el administrador de Cache.

Specified by:
getProduct in interface B2CArticleManagerInterface

Parameters:
key - Identificador de producto

Returns:
Un producto

Throws:
B2CCacheException - Excepción al acceder a la cache´
B2CProductException - Producto no existe en la cache

See Also:
B2CArticleManagerInterface.getProduct(int)

---

getCatalogue

```
public B2CCatalogue getCatalogue(int key)
                          throws B2CCatalogueException,
                                 B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Obtiene un catálogo. El procedimiento por defecto es buscar el catálogo en el administrador de Cache.

Specified by:
getCatalogue in interface B2CArticleManagerInterface

Parameters:
key - Identificador de catálogo

Returns:
Un catálogo

Throws:
B2CCatalogueException - Catálogo no existe en la cache
B2CCacheException - Excepción al acceder a la cache´

See Also:
B2CArticleManagerInterface.getCatalogue(int)

---

getPromotion

```
public B2CPromotion getPromotion(int key)
                          throws B2CPromotionException,
                                 B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Obtiene una promoción. El procedimiento por defecto es buscar la promoción en el administrador de Cache.

Specified by:
getPromotion in interface B2CArticleManagerInterface

Parameters:
key - Identificador de promoción

Returns:
Una promoción

Throws:
B2CPromotionException - Promoción no existe en la cache
B2CCacheException - Excepción al acceder a la cache´

See Also:
B2CArticleManagerInterface.getPromotion(int)

---

clearArticleCache

```
public void clearArticleCache()
                    throws B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Limpia la cache de Artículos
Specified by:
clearArticleCache in interface B2CArticleManagerInterface
Throws:
B2CCacheException - Excepción al acceder a la cache
See Also:
B2CArticleManagerInterface.clearArticleCache()

---

clearProductCache

```
public void clearProductCache()
                    throws B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Limpia la cache de Productos
Specified by:
clearProductCache in interface B2CArticleManagerInterface
Throws:
B2CCacheException - Excepción al acceder a la cache
See Also:
B2CArticleManagerInterface.clearProductCache()

---

clearCatalogueCache

```
public void clearCatalogueCache()
                      throws B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Limpia la cache de Catálogos
Specified by:
clearCatalogueCache in interface B2CArticleManagerInterface
Throws:
B2CCacheException - Excepción al acceder a la cache
See Also:
B2CArticleManagerInterface.clearCatalogueCache()

---

clearPromotionCache

```
public void clearPromotionCache()
                        throws B2CCacheException
```

Description copied from interface: B2CArticleManagerInterface
Limpia la cache de Promociones
Specified by:
clearPromotionCache in interface B2CArticleManagerInterface
Throws:
B2CCacheException - Excepción al acceder a la cache
See Also:
B2CArticleManagerInterface.clearPromotionCache()

---

getCacheManager

```
public B2CCacheManager getCacheManager()
```

Obtiene la referencia hacia el Administrador de Cache
Returns:
Devuelve la referencia hacia el administrador de cache

---

setCacheManager

```
public void setCacheManager(B2CCacheManager cacheManager)
```

Establece la referencia hacia el Administrador de Cache
Parameters:
cacheManager - La referencia hacia el Administrador de Cache

---

b2c.framework.article

Class B2CProduct

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CElement
          └b2c.framework.article.B2CProduct
```

All Implemented Interfaces:

B2CElementInterface, java.lang.Cloneable

---

public class **B2CProduct**
extends B2CElement

Clase que representa a un producto

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CElement

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CProduct(int aKey)
Único constructor.

Methods inherited from class b2c.framework.basicelement.B2CElement

addCharacteristic, addDescription, addPrice, clone, getCharacteristicByType, getCharacteristics, getDescriptions, getDescriptionsByType, getPriceByType, getPrices

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

B2CProduct

`public` **`B2CProduct`**`(int aKey)`

Único constructor. Se debe establcer un identificdor

---

b2c.framework.article

Class B2CPromotion

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CElement
          └b2c.framework.basicelement.B2CAdvancedElement
              └b2c.framework.article.B2CPromotion
```

All Implemented Interfaces:

B2CAdvancedElementInterface, B2CElementInterface, java.lang.Cloneable

Direct Known Subclasses:

B2CCatalogue

---

public class **B2CPromotion**
extends B2CAdvancedElement

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CAdvancedElement
`comparator`, `images`, `subElements`
Fields inherited from class b2c.framework.basicelement.B2CElement
Fields inherited from class b2c.framework.basicelement.B2CObjectElementary
`key`

Constructor Summary

`B2CPromotion``(int aKey)` Constructor for B2CPromotion	

Method Summary

void	**addArticle**(B2CArticle article)
B2CArticle	**getArticle**(int aKey)
B2CArticle[]	**getArticles**()

Methods inherited from class b2c.framework.basicelement.B2CAdvancedElement

addComparator, addImage, addSubElement, applyRelationRules, clone, getComparator, getImageByType, getImages, getSubElement, getSubElements

Methods inherited from class b2c.framework.basicelement.B2CElement

addCharacteristic, addDescription, addPrice, getCharacteristicByType, getCharacteristics, getDescriptions, getDescriptionsByType, getPriceByType, getPrices

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

B2CPromotion

```
public B2CPromotion(int aKey)
```

Constructor for B2CPromotion

Method Detail

getArticle

```
public B2CArticle getArticle(int aKey)
                      throws B2CArticleException
```

Throws:
B2CArticleException

---

getArticles

```
public B2CArticle[] getArticles()
                        throws B2CArticleException
```

Throws:
B2CArticleException

addArticle

```
public void addArticle(B2CArticle article)
```

Package b2c.framework.servlet

Class Summary	
B2CProcessRouter	Clase que implementa el Encausador de Procesos.
B2CRequestFilter	Clase que implementa el Filtro de Requerimientos.
B2CServlet	Clase principal Servlet.
B2CVisualManager	Clase que implementa al Administrador de Vitrina.

---

b2c.framework.servlet

Class B2CProcessRouter

```
java.lang.Object
  └b2c.framework.servlet.B2CProcessRouter
```

---

public abstract class **B2CProcessRouter**
extends java.lang.Object

Clase que implementa el Encausador de Procesos. Esta clase discrimina mediante una etiqueta que identifica al proceso a ser ejecutado, llamando a un procedimiento llamado como la etiqueta lo indica. Para ello, se deben implementar estos métodos. Adicionalmente, cada uno de estos métodos son responsable de manipular los objetos que contienen la información desde y hacia el cliente. Finalmente está disponible el método: goToVisual(HttpServletRequest req, HttpServletResponse resp, String URL) para redireccionar la salida del sistema. Cada método debe devolver una URL hacia el JSP correspondiente llevando el resultado del proceso

Version:
1.0
Author:

Field Summary

private B2CArticleManager	**articleManager** Referencia hacia el Administrador de Artículos
private B2CVisualManager	**visualManager** Referencia hacia el Administrador de Vitrina

Constructor Summary

B2CProcessRouter(B2CVisualManager visualManager, B2CArticleManager articleManager) Constructor único que enlaza al Administrador de Vitrina

Method Summary

B2CArticleManager	**getArticleManager**() Obtiene la referencia hacia el Administrador de Artículos
B2CVisualManager	**getVisualManager**() Obtiene la referencia hacie el Administrador de Vitrina
void	**goToVisual**(java.lang.String jspPath, HttpServletRequest req, HttpServletResponse resp) Redirecciona la salida hacia el administrador de vitrina, con la URL indicada
void	**performTask**(java.lang.String process, HttpServletRequest req, HttpServletResponse resp) Método encargado de ejecutar métodos implementados bajo la etiqueta 'process'.
void	**setArticleManager**(B2CArticleManager articleManager) Establece la referencia hacia el Administrador de Artículos
void	**setVisualManager**(B2CVisualManager visualManager) Obtiene la referencia hacia el Administrador de Vitrina

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

visualManager

private B2CVisualManager **visualManager**
Referencia hacia el Administrador de Vitrina

---

articleManager

```
private B2CArticleManager articleManager
```

Referencia hacia el Administrador de Artículos

Constructor Detail

B2CProcessRouter

```
public B2CProcessRouter(B2CVisualManager visualManager,
                        B2CArticleManager articleManager)
```

Constructor único que enlaza al Administrador de Vitrina

Parameters:

`visualManager` - Referencia hacia el Administrador de Vitrina

Method Detail

performTask

```
public void performTask(java.lang.String process,
                        HttpServletRequest req,
                        HttpServletResponse resp)
                 throws java.lang.Exception
```

Método encargado de ejecutar métodos implementados bajo la etiqueta 'process'. Estos métodos deben serl tipo: 'process(HttpServletRequest req, HttpServletResponse resp)'

Parameters:

`process` - Etiqueta que identifica a un método a implementar y que ejecuta una tarea específica. El método debe llamarse de forma idéntica a lo que la etiqueta establece

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`java.lang.Exception` - Cualquier excepción acontencida durante el proceso

---

goToVisual

```
public void goToVisual(java.lang.String jspPath,
                       HttpServletRequest req,
                       HttpServletResponse resp)
                throws java.io.IOException,
                       ServletException,
                       java.lang.Exception
```

Redirecciona la salida hacia el administrador de vitrina, con la URL indicada

Parameters:

`jspPath` - ruta con la ubicación del JSP de interés, desde el contexto del sitio web

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`java.io.IOException` - Excepción en la búsqueda del JSP

`ServletException` - Excepción en el Servlet Principal

`java.lang.Exception`

---

getVisualManager

```
public B2CVisualManager getVisualManager()
```

Obtiene la referencia hacie el Administrador de Vitrina

Returns:

La referencia hacia el administrador de vitrina

---

getArticleManager

```
public B2CArticleManager getArticleManager()
```

Obtiene la referencia hacia el Administrador de Artículos

Returns:

la referencia hacia el administrador de artículos

---

setVisualManager

```
public void setVisualManager(B2CVisualManager visualManager)
```

Obtiene la referencia hacia el Administrador de Vitrina

Parameters:

`visualManager` - La referencia hacia el administrador de vitrina

---

setArticleManager

`public void` **`setArticleManager`**`(B2CArticleManager articleManager)`

Establece la referencia hacia el Administrador de Artículos

Parameters:

`articleManager` - La referencia hacia el administrador de artículos

---

b2c.framework.servlet

Class B2CRequestFilter

```
java.lang.Object
  └ b2c.framework.servlet.B2CRequestFilter
```

---

public abstract class **B2CRequestFilter**
extends java.lang.Object

Clase que implementa el Filtro de Requerimientos. Este filtro se encarga de analizar la información que llega desde el cliente (HttpServletRequest), realizar filtraje personalizado mediante un método toFilter() que debe ser implementado. Finalmente puede utilizarse el método goToProcess() para delegar el control de los procesos a la clase especializada por el Encausador de Procesos. Posee un constructor único quien recibe las referencias hacia el Encausador de Procesos y Servlet Principal.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

private B2CProcessRouter	**processRouter** Contiene la referencia hacia el objeto que implementa el Encausador de Procesos
private B2CServlet	**servletReference** Referencia al servlet principal

Constructor Summary

B2CRequestFilter(B2CProcessRouter processRouter, B2CServlet servletReference) Constructor único que establece el enlace con el Encausador	

de Procesos	

Method Summary

void	**doProcess**(java.lang.String process, HttpServletRequest req, HttpServletResponse resp) Método que redirecciona hacia el Encausador de Procesos.
B2CProcessRouter	**getProcessRouter**() Obtiene la referencia hacia el Encausador de Procesos
B2CServlet	**getServletReference**() Obtiene la referencia hacia el Servlet Principal
void	**setProcessRouter**(B2CProcessRouter processRouter) Establece la referencia hacia el Encausador de Procesos
void	**setServletReference**(B2CServlet servletReference) Establece la referencia hacia el Servlet Principal
abstract void	**toFilter**(HttpServletRequest req, HttpServletResponse resp) Método no implementado que realiza el filtro de información proveniente desde el cliente.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

processRouter

```
private B2CProcessRouter processRouter
```

Contiene la referencia hacia el objeto que implementa el Encausador de Procesos

---

servletReference

```
private B2CServlet servletReference
```

Referencia al servlet principal

Constructor Detail

B2CRequestFilter

```
public B2CRequestFilter(B2CProcessRouter processRouter,
                        B2CServlet servletReference)
```

Constructor único que establece el enlace con el Encausador de Procesos

Parameters:

`processRouter` - Referencia al encausador de procesos

`servletReference` - Referencia hacia el Servlet Principal

Method Detail

toFilter

```
public abstract void toFilter(HttpServletRequest req,
                              HttpServletResponse resp)
```

Método no implementado que realiza el filtro de información proveniente desde el cliente. Puede fijar la etiqueta 'process' para ser utilizada en el método goToProcess(String process, HttpServletRequest req, HttpServletResponse resp)

Parameters:

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

---

doProcess

```
public void doProcess(java.lang.String process,
                      HttpServletRequest req,
                      HttpServletResponse resp)
               throws java.lang.Exception
```

Método que redirecciona hacia el Encausador de Procesos. Eventualmente en el Encausador de procesos, se discrimanrá mediante una etiqueta qué tareas deben realizarse en el sistema

Parameters:

`process` - Etiqueta que referencia a un determinado proceso a ser ejecutado.

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`java.lang.Exception` - Alguna excepción en el procesamiento de la tarea

---

getProcessRouter

`public B2CProcessRouter getProcessRouter()`

Obtiene la referencia hacia el Encausador de Procesos

Returns:

La referencia hacia el encausador de procesos

---

getServletReference

`public B2CServlet getServletReference()`

Obtiene la referencia hacia el Servlet Principal

Returns:

Devuelve la referencia hacia el Servlet

---

setProcessRouter

`public void setProcessRouter(B2CProcessRouter processRouter)`

Establece la referencia hacia el Encausador de Procesos

Parameters:

`processRouter` - Referencia hacia el encausador de procesos

---

setServletReference

`public void setServletReference(B2CServlet servletReference)`

Establece la referencia hacia el Servlet Principal

Parameters:

`servletReference` - La referencia hacia el Servlet

---

b2c.framework.servlet

Class B2CServlet

```
java.lang.Object
  └HttpServlet
      └b2c.framework.servlet.B2CServlet
```

---

public abstract class **B2CServlet**
extends HttpServlet

Clase principal Servlet. Maneja procedimiento cliente - servidor via HTTP. Posee dos propiedades para establecer la relación con un Administrador de Vitrina y un Filtro de Requerimientos. Para ello, es recomendable implementar el método *init()* y ahí establecer dicha relación, además de establecer el tamaño de los elementos de cache dado por los métodos de prefijo *setSize*. Esta clase sólo es una interfaz entre el cliente y el sistema, su única función es encaminar el flujo de la interacción.

Version:
1.0
Author:
Javier Villalobos Arancibia
See Also:
Serialized Form

Field Summary

private B2CRequestFilter	**requestFilter** Objeto que enlaza la referencia hacia el Filtro de Requerimientos
private B2CVisualManager	**visualManager** Objeto que enlaza la referencia hacia el administrador de vitrina

Constructor Summary

B2CServlet()	

Method Summary	
void	**doFilter**(HttpServletRequest req, HttpServletResponse resp) Realiza llamado al Filtro de Requerimientos
void	**doGet**(HttpServletRequest req, HttpServletResponse resp) Manipula el método HTTP GET
void	**doPost**(HttpServletRequest req, HttpServletResponse resp) Manipula el método HTTP POST
B2CRequestFilter	**getRequestFilter**() Obtiene la referencia hacia el Filtro de Requerimientos.
B2CVisualManager	**getVisualManager**() Obtiene la referencia hacia el Administrador de Vitrina
void	**goToPage**(java.lang.String jspPath, HttpServletRequest req, HttpServletResponse resp) Redirecciona el flujo hacia un JSP.
void	**setRequestFilter**(B2CRequestFilter requestFilter) Establece el objeto que implementa al Filtro de Requerimientos.
void	**setSizeArticles**(int size) Establece el tamaño de la cache de Artículos
void	**setSizeCatalogues**(int size) Establece el tamaño de la cache de Catálogos
void	**setSizeProducts**(int size) Establece el tamaño de la cache de Productos
void	**setSizePromotions**(int size) Establece el tamaño de la cache de Promociones
void	**setVisualManager**(B2CVisualManager visualManager) Establece el objeto que implementa al Administrador de Vitrina.
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

visualManager

```
private B2CVisualManager visualManager
```

Objeto que enlaza la referencia hacia el administrador de vitrina

---

requestFilter

```
private B2CRequestFilter requestFilter
```

Objeto que enlaza la referencia hacia el Filtro de Requerimientos

Constructor Detail

B2CServlet

```
public B2CServlet()
```

Method Detail

doFilter

```
public void doFilter(HttpServletRequest req,
                     HttpServletResponse resp)
              throws ServletException
```

Realiza llamado al Filtro de Requerimientos

Parameters:

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`ServletException` - Excepción generalizada del Servlet

---

setSizeCatalogues

```
public void setSizeCatalogues(int size)
```

Establece el tamaño de la cache de Catálogos

Parameters:

`size` - Tamaño de la cache de catálogos (cantidad de catálogos)

---

setSizeArticles

```
public void setSizeArticles(int size)
```

Establece el tamaño de la cache de Artículos

Parameters:

`size` - Tamaño de la cache de artículos (cantidad de artículos)

---

setSizeProducts

```
public void setSizeProducts(int size)
```

Establece el tamaño de la cache de Productos

Parameters:

`size` - Tamaño de la cache de productos (cantidad de productos)

---

setSizePromotions

```
public void setSizePromotions(int size)
```

Establece el tamaño de la cache de Promociones

Parameters:

`size` - Tamaño de la cache de promociones (cantidad de promociones)

---

goToPage

```
public void goToPage(java.lang.String jspPath,
                     HttpServletRequest req,
                     HttpServletResponse resp)
              throws ServletException,
                     java.io.IOException
```

Redirecciona el flujo hacia un JSP. Una vez establecida la información a enviar al cliente, se direcciona el fujo hacia un JSP

Parameters:

`jspPath` - Ubicación y nombre del archivo JSP, relativo al contexto del sitio web

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`ServletException` - Excepción relacionado con el Servlet

`java.io.IOException` - Excepción en la búsqueda del JSP

---

doGet

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
           throws ServletException,
                  java.io.IOException
```

Manipula el método HTTP GET

Parameters:

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`ServletException` - Excepción relacionado con el Servlet

`java.io.IOException` - Excepción en la búsqueda del JSP

---

doPost

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse resp)
            throws ServletException,
                   java.io.IOException
```

Manipula el método HTTP POST

Parameters:

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`ServletException` - Excepción relacionado con el Servlet

`java.io.IOException` - Excepción en la búsqueda del JSP

---

getVisualManager

```
public B2CVisualManager getVisualManager()
```

Obtiene la referencia hacia el Administrador de Vitrina

Returns:

La referencia al adminsitrador de vitrina

---

setVisualManager

`public void` **`setVisualManager`**`(B2CVisualManager visualManager)`

Establece el objeto que implementa al Administrador de Vitrina. Para esto, se debe instanciar la clase que implementa el Administrador de Vitrina con el constructor predeterminado

---

getRequestFilter

`public B2CRequestFilter` **`getRequestFilter`**`()`

Obtiene la referencia hacia el Filtro de Requerimientos.

Returns:

La referencia al filtro de requerimientos

---

setRequestFilter

`public void` **`setRequestFilter`**`(B2CRequestFilter requestFilter)`

Establece el objeto que implementa al Filtro de Requerimientos. Para esto, se debe instanciar la clase que implementa el Filtro de Requerimientos con el constructor predeterminado

---

b2c.framework.servlet

Class B2CVisualManager

```
java.lang.Object
  └ b2c.framework.servlet.B2CVisualManager
```

---

public abstract class **B2CVisualManager**
extends java.lang.Object

Clase que implementa al Administrador de Vitrina. Posee un constructor único en el cual se establece el enlace con el Servlet Principal. Utiliza el método *goOut()* para redireccionar la salida hacia el cliente a través del Servlet Principal. En general esta clase, se encargará de armar la información visual que será enviada al cliente.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

protected B2CServlet	**servletReference** Referencia hacia el Servlet Principal

Constructor Summary

B2CVisualManager(B2CServlet servletReference) Constructor único para establecer el enlace con el Servlet Principal

Method Summary

B2CServlet	**getServletReference**() Obtiene la referencia hacia el Servlet Principal
void	**goOut**(java.lang.String jspPath, HttpServletRequest req, HttpServletResponse resp) Método que hace el redireccionamiento a un JSP o salida del Servlet.
void	**setServletReference**(B2CServlet servletReference) Establece la referencia hacia el Servlet Principal

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

servletReference

protected B2CServlet **servletReference**
Referencia hacia el Servlet Principal

Constructor Detail

B2CVisualManager

public **B2CVisualManager**(B2CServlet servletReference)
Constructor único para establecer el enlace con el Servlet Principal

Parameters:
servletReference - Referencia al servlet principal

Method Detail

goOut

```
public void goOut(java.lang.String jspPath,
                  HttpServletRequest req,
                  HttpServletResponse resp)
           throws ServletException,
                  java.io.IOException
```

Método que hace el redireccionamiento a un JSP o salida del Servlet. Este método llama a goToPage(jspPath, req, resp) en el servlet principal

Parameters:

`jspPath` - Ruta hacia un JSP, relativo al contexto del sitio web

`req` - Objeto con contenido del requerimiento desde el cliente

`resp` - Objeto con información sobre el cliente para retorno de la llamada

Throws:

`ServletException` - Excepción en el Servlet

`java.io.IOException` - Excepción en la búsqueda del JSP

---

getServletReference

```
public B2CServlet getServletReference()
```

Obtiene la referencia hacia el Servlet Principal

Returns:

Referencia hacia el Servlet

---

setServletReference

```
public void setServletReference(B2CServlet servletReference)
```

Establece la referencia hacia el Servlet Principal

Parameters:

`servletReference` - La referencia hacia el servlet

Package b2c.framework.util.resource

Class Summary	
B2CResource	Clase de tipo singleton para la obtención de variable de inicialización.

---

b2c.framework.util.resource

Class B2CResource

```
java.lang.Object
  └ b2c.framework.util.resource.B2CResource
```

---

public class **B2CResource**
extends java.lang.Object

Clase de tipo singleton para la obtención de variable de inicialización.

Version:
1.0
Author:
Javier Villalobos Arancibia

Field Summary	
private java.util.PropertyResourceBundle	**bundle** Referencia hacia el objeto PropertyResourceBundle.
private static B2CResource	**instanceB2CResource** Instancia de tipo singleton para ResourceB2C.

Constructor Summary	
B2CResource(java.lang.String nameFile) Crea una nueva instancia de ResourceB2C	

Method Summary

static B2CResource	**getInstance**() Genera y/u obtiene la instancia única de ResourceB2C.
static B2CResource	**getInstance**(java.lang.String nameFile) Genera y/u obtiene la instancia única de ResourceB2C.
java.lang.String	**getProperty**(java.lang.String key) Lee valor en el recurso (bundle) correspondiente a la 'key' entregada.
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

instanceB2CResource

```
private static B2CResource instanceB2CResource
```

Instancia de tipo singleton para ResourceB2C.

---

bundle

```
private java.util.PropertyResourceBundle bundle
```

Referencia hacia el objeto PropertyResourceBundle.

Constructor Detail

B2CResource

```
public B2CResource(java.lang.String nameFile)
            throws B2CResourceException
```

Crea una nueva instancia de ResourceB2C

Throws:

`ResourceB2CException` - No es posible construir el boundle
`B2CResourceException`

Method Detail

getInstance

```
public static B2CResource getInstance()
                              throws B2CResourceException
```

Genera y/u obtiene la instancia única de ResourceB2C.

Returns:

Instancia de ResourceB2C

Throws:

`ResourceB2CException` - Excepción al generar el resourceBundle.
`B2CResourceException`

---

getInstance

```
public static B2CResource getInstance(java.lang.String nameFile)
                              throws B2CResourceException
```

Genera y/u obtiene la instancia única de ResourceB2C.

Returns:

Instancia de ResourceB2C

Throws:

`ResourceB2CException` - Excepción al generar el resourceBundle.
`B2CResourceException`

---

getProperty

```
public java.lang.String getProperty(java.lang.String key)
                             throws NotFoundPropertyException
```

Lee valor en el recurso (bundle) correspondiente a la 'key' entregada.

Parameters:

`key` - Clave de referencia en el recurso (bundle).

Returns:

Valor del recurso leído.

Throws:

`NotFoundPropertyException` - Excepción en la búsqueda del valor paramétrico.

Package b2c.framework.descriptionelement

Class Summary	
B2CCharacteristic	Clase que representa una característica cualquiera, tiene un contenido, un tipo y una cantidad
B2CDescription	Clase que representa a una descripción, con contenido y tipo
B2CImage	Clase que representa una imagen
B2CPrice	Clase que representa a un precio
B2CType	Clase que describe un tipo cualquiera.

b2c.framework.descriptionelement

Class B2CCharacteristic

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.basicelement.B2CContentTyped
              └b2c.framework.descriptionelement.B2CCharacteristic
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **B2CCharacteristic**
extends B2CContentTyped

Clase que representa una característica cualquiera, tiene un contenido, un tipo y una cantidad

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary	
private double	**quantity** Propiedad que contiene una cantidad
Fields inherited from class b2c.framework.basicelement.B2CContentTyped	
Fields inherited from class b2c.framework.basicelement.B2CContent	
Fields inherited from class b2c.framework.basicelement.B2CObjectElementary	
key	

Constructor Summary	
B2CCharacteristic()	

Method Summary	
double	**getQuantity**() Obtiene la cantidad asociada
void	**setQuantity**(double aQuantity) Asigna una cantidad a la característica
Methods inherited from class b2c.framework.basicelement.B2CContentTyped	
getType, setType	
Methods inherited from class b2c.framework.basicelement.B2CContent	
clone, getContent, setContent	
Methods inherited from class java.lang.Object	
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

quantity

```
private double quantity
```

Propiedad que contiene una cantidad

Constructor Detail

B2CCharacteristic

```
public B2CCharacteristic()
```

Method Detail

getQuantity

```
public double getQuantity()
```

Obtiene la cantidad asociada

Returns:

Una cantidad

---

setQuantity

```
public void setQuantity(double aQuantity)
```

Asigna una cantidad a la característica

---

b2c.framework.descriptionelement

Class B2CPrice

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.basicelement.B2CContentTyped
              └b2c.framework.descriptionelement.B2CPrice
```

All Implemented Interfaces:

java.lang.Cloneable

public class **B2CPrice**
extends B2CContentTyped

Clase que representa a un precio

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CContentTyped

Fields inherited from class b2c.framework.basicelement.B2CContent

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

`key`

Constructor Summary

`B2CPrice``()`	

Method Summary

`double`	**`getPrice`**`()` Obtiene el valor del precio
`void`	**`setValue`**`(double aValue)` Establece el valor del precio

Methods inherited from class b2c.framework.basicelement.B2CContentTyped

`getType`, `setType`

Methods inherited from class b2c.framework.basicelement.B2CContent

`clone`, `getContent`, `setContent`

Methods inherited from class java.lang.Object

`equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

B2CPrice

```
public B2CPrice()
```

Method Detail

getPrice

```
public double getPrice()
                throws java.lang.NumberFormatException
```

Obtiene el valor del precio

Returns:

Un valor monetario

Throws:

`java.lang.NumberFormatException` - El valor contenido en el precio no puede ser transformado a un valor de tipo double

---

setValue

```
public void setValue(double aValue)
```

Establece el valor del precio

Parameters:

`aValue` - Un valor numérico cualquiera

---

b2c.framework.descriptionelement

Class B2CDescription

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.basicelement.B2CContentTyped
              └b2c.framework.descriptionelement.B2CDescription
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **B2CDescription**
extends B2CContentTyped

Clase que representa a una descripción, con contenido y tipo

Version:
1.0
Author:
Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CContentTyped

Fields inherited from class b2c.framework.basicelement.B2CContent

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

`key`

Constructor Summary

`B2CDescription()`

Methods inherited from class b2c.framework.basicelement.B2CContentTyped

`getType`, `setType`

Methods inherited from class b2c.framework.basicelement.B2CContent

`clone`, `getContent`, `setContent`

Methods inherited from class java.lang.Object

`equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

B2CDescription

```
public B2CDescription()
```

---

b2c.framework.descriptionelement

Class B2CImage

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.basicelement.B2CContentTyped
              └b2c.framework.descriptionelement.B2CImage
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **B2CImage**
extends B2CContentTyped

Clase que representa una imagen

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CContentTyped
Fields inherited from class b2c.framework.basicelement.B2CContent
Fields inherited from class b2c.framework.basicelement.B2CObjectElementary
key

Constructor Summary

B2CImage()
Methods inherited from class b2c.framework.basicelement.B2CContentTyped
getType, setType

Methods inherited from class b2c.framework.basicelement.B2CContent
clone, getContent, setContent
Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

B2CImage

```
public B2CImage()
```

---

b2c.framework.descriptionelement

Class B2CType

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.basicelement.B2CContent
          └b2c.framework.descriptionelement.B2CType
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **B2CType**
extends B2CContent

Clase que describe un tipo cualquiera. Esta clase es utilizada para agrupar o clasificar descripciones mediante un tipo. Cada tipo contiene un identificador y una descripción o contenido.

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

Fields inherited from class b2c.framework.basicelement.B2CContent

Fields inherited from class b2c.framework.basicelement.B2CObjectElementary

key

Constructor Summary

B2CType()

Methods inherited from class b2c.framework.basicelement.B2CContent

clone, getContent, setContent

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

B2CType

public **B2CType**()

Package b2c.framework.cache

Interface Summary	
B2CArticleCacheInterface	Interfaz para cache de artículos
B2CCacheManagerInterface	Interfaz para el Administrador de Cache
B2CCatalogueCacheInterface	Interfaz para cache de catálogos
B2CProductCacheInterface	Interfaz para cache de productos
B2CPromotionCacheInterface	Interfaz para cache de promociones

Class Summary	
B2CArticleCache	Clase que implementa la cache de Artículos
B2CCacheManager	Administrador de Cache.
B2CCatalogueCache	Clase que implementa la cache de catálogos
B2CElementCache	Clase que implementa a la cache propiamente tal. Haciendo referencia genéricamente a un elemento posible de tratar en una cache
B2CLRUElement	Clase que implementa un elemento en la cache cuyo ciclo de vida está sujeto al método LRU
B2CLRUObject	Clase que implementa el seguimiento y estadísticas de un elemento en la cache, dado el método LRU
B2CProductCache	Clase que implementa la cache de productos
B2CPromotionCache	Clase que implementa la cache de promociones

b2c.framework.cache

Interface B2CArticleCacheInterface

All Known Subinterfaces:
B2CCacheManagerInterface
All Known Implementing Classes:
B2CCacheManager

---

public interface **B2CArticleCacheInterface**

Method Summary

void	**clearArticleCache**()
B2CArticle	**getArticle**(java.lang.Integer key)
B2CArticle	**getArticleFromCache**(int aKey)

Method Detail

getArticle

```
public B2CArticle getArticle(java.lang.Integer key)
                      throws B2CArticleException,
                             B2CCacheException
```

Throws:
B2CArticleException
B2CCacheException

---

getArticleFromCache

```
public B2CArticle getArticleFromCache(int aKey)
                                throws B2CArticleException,
                                       B2CCacheException
```

Throws:
B2CArticleException
B2CCacheException

---

clearArticleCache

```
public void clearArticleCache()
                        throws B2CCacheException
```

Throws:
B2CCacheException

---

b2c.framework.cache

Interface B2CCacheManagerInterface

All Superinterfaces:
B2CArticleCacheInterface, B2CCatalogueCacheInterface, B2CProductCacheInterface, B2CPromotionCacheInterface

All Known Implementing Classes:
B2CCacheManager

---

public interface **B2CCacheManagerInterface**
extends B2CProductCacheInterface, B2CArticleCacheInterface, B2CCatalogueCacheInterface, B2CPromotionCacheInterface

---

Methods inherited from interface b2c.framework.cache.B2CProductCacheInterface
clearProductCache, getProduct, getProductFromCache
Methods inherited from interface b2c.framework.cache.B2CArticleCacheInterface
clearArticleCache, getArticle, getArticleFromCache
Methods inherited from interface b2c.framework.cache.B2CCatalogueCacheInterface
clearCatalogueCache, getCatalogue, getCatalogueFromCache
Methods inherited from interface b2c.framework.cache.B2CPromotionCacheInterface
clearPromotionCache, getPromotion, getPromotionFromCache

---

b2c.framework.cache

Interface B2CCatalogueCacheInterface

All Known Subinterfaces:
B2CCacheManagerInterface
All Known Implementing Classes:
B2CCacheManager

---

public interface **B2CCatalogueCacheInterface**

Interfaz para cache de catálogos

Method Summary

void	**clearCatalogueCache**()
B2CCatalogue	**getCatalogue**(java.lang.Integer key)
B2CCatalogue	**getCatalogueFromCache**(int aKey)

Method Detail

getCatalogue

```
public B2CCatalogue getCatalogue(java.lang.Integer key)
                          throws B2CCatalogueException,
                                 B2CCacheException
```

Throws:
B2CCatalogueException
B2CCacheException

---

getCatalogueFromCache

```
public B2CCatalogue getCatalogueFromCache(int aKey)
                                   throws B2CCatalogueException,
                                          B2CCacheException
```

Throws:
B2CCatalogueException
B2CCacheException

---

clearCatalogueCache

```
public void clearCatalogueCache()
                        throws B2CCacheException
```

Throws:
B2CCacheException

---

b2c.framework.cache

Interface B2CProductCacheInterface

All Known Subinterfaces:
B2CCacheManagerInterface
All Known Implementing Classes:
B2CCacheManager

---

public interface **B2CProductCacheInterface**

Method Summary

void	**clearProductCache**()
B2CProduct	**getProduct**(java.lang.Integer key)
B2CProduct	**getProductFromCache**(int aKey)

Method Detail

getProduct

```
public B2CProduct getProduct(java.lang.Integer key)
                      throws B2CProductException,
                             B2CCacheException
```

Throws:
B2CProductException
B2CCacheException

---

getProductFromCache

```
public B2CProduct getProductFromCache(int aKey)
                                throws B2CProductException,
                                       B2CCacheException
```

Throws:
B2CProductException
B2CCacheException

---

clearProductCache

```
public void clearProductCache()
                     throws B2CCacheException
```

Throws:
B2CCacheException

---

b2c.framework.cache

Interface B2CPromotionCacheInterface

All Known Subinterfaces:
B2CCacheManagerInterface
All Known Implementing Classes:
B2CCacheManager

---

public interface **B2CPromotionCacheInterface**

Method Summary	
void	**clearPromotionCache**()
B2CPromotion	**getPromotion**(java.lang.Integer key)
B2CPromotion	**getPromotionFromCache**(int aKey)

Method Detail

getPromotion

```
public B2CPromotion getPromotion(java.lang.Integer key)
                          throws B2CPromotionException,
                                 B2CCacheException
```

Throws:
B2CPromotionException
B2CCacheException

---

getPromotionFromCache

```
public B2CPromotion getPromotionFromCache(int aKey)
                                   throws B2CPromotionException,
                                          B2CCacheException
```

Throws:
B2CPromotionException
B2CCacheException

---

clearPromotionCache

```
public void clearPromotionCache()
                         throws B2CCacheException
```

Throws:
B2CCacheException

---

b2c.framework.cache

Class B2CArticleCache

```
java.lang.Object
  └b2c.framework.cache.B2CElementCache
      └b2c.framework.cache.B2CArticleCache
```

---

public class **B2CArticleCache**
extends B2CElementCache

Field Summary

protected static B2CArticleCache	**instaceArticleCache**

Fields inherited from class b2c.framework.cache.B2CElementCache

b2cLruElement, elementCacheMap, maxSize, nameMethod, sizeWarning

Constructor Summary

B2CArticleCache(int size)	

Method Summary

static B2CArticleCache	**getInstance**()
static B2CArticleCache	**getInstance**(int size)

Methods inherited from class b2c.framework.cache.B2CElementCache

clearCache, getElement

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instaceArticleCache

```
protected static B2CArticleCache instaceArticleCache
```

Constructor Detail

B2CArticleCache

```
public B2CArticleCache(int size)
```

b2c.framework.cache

Class B2CCacheManager

```
java.lang.Object
  └ b2c.framework.cache.B2CCacheManager
```

All Implemented Interfaces:

B2CArticleCacheInterface, B2CCacheManagerInterface, B2CCatalogueCacheInterface, B2CProductCacheInterface, B2CPromotionCacheInterface

---

public abstract class **B2CCacheManager**
extends java.lang.Object
implements B2CCacheManagerInterface

Administrador de Cache. Utiliza métodos definidos para la obtención de elementos de la cache. Adicionalemente hereda la obligación de implementar métodos para obtener inicialmente los elementos fuera de la cache

Version:

1.0

Author:

Javier Villalobos Arancibia

Field Summary

protected java.lang.Object	**cacheManagerClass** Referencia hacia la implementación personalizada de un Administrador de Cache.

Constructor Summary

B2CCacheManager() Constructor del Administrador de Cache.	

Method Summary

void	**clearArticleCache**()
void	**clearCatalogueCache**()

void	**clearProductCache**()
void	**clearPromotionCache**()
B2CArticle	**getArticleFromCache**(int aKey)
B2CCatalogue	**getCatalogueFromCache**(int aKey)
B2CProduct	**getProductFromCache**(int aKey)
B2CPromotion	**getPromotionFromCache**(int aKey)

Method Detail

getInstance

```
public static B2CArticleCache getInstance()
                                  throws B2CCacheException
```

Throws:
B2CCacheException

---

getInstance

```
public static B2CArticleCache getInstance(int size)
```

---

b2c.framework.cache

Class B2CCatalogueCache

```
java.lang.Object
  └b2c.framework.cache.B2CElementCache
      └b2c.framework.cache.B2CCatalogueCache
```

---

public class **B2CCatalogueCache**
extends B2CElementCache

Field Summary

protected static B2CCatalogueCache	**instaceCatalogueCache**

Fields inherited from class b2c.framework.cache.B2CElementCache

b2cLruElement, elementCacheMap, maxSize, nameMethod, sizeWarning

Constructor Summary

B2CCatalogueCache(int size)	

Method Summary

static B2CCatalogueCache	**getInstance**()
static B2CCatalogueCache	**getInstance**(int size)

Methods inherited from class b2c.framework.cache.B2CElementCache

clearCache, getElement

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instaceCatalogueCache

protected static B2CCatalogueCache **instaceCatalogueCache**

Constructor Detail

B2CCatalogueCache

public **B2CCatalogueCache**(int size)

Method Detail

getInstance

```
public static B2CCatalogueCache getInstance()
                                 throws B2CCacheException
```

Throws:
B2CCacheException

---

getInstance

```
public static B2CCatalogueCache getInstance(int size)
```

---

b2c.framework.cache

Class B2CElementCache

```
java.lang.Object
  └ b2c.framework.cache.B2CElementCache
```

Direct Known Subclasses:
B2CArticleCache, B2CCatalogueCache, B2CProductCache, B2CPromotionCache

---

public class **B2CElementCache**
extends java.lang.Object

Field Summary

protected B2CLRUElement	**b2cLruElement**
protected java.util.TreeMap	**elementCacheMap**
protected int	**maxSize**
protected java.lang.String	**nameMethod**
protected int	**sizeWarning**

Constructor Summary

B2CElementCache(int size)	

Method Summary

void	**clearCache**()
B2CElement	**getElement**(java.lang.Object managerClass, int aKey)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

elementCacheMap

protected java.util.TreeMap **elementCacheMap**

---

maxSize

protected int **maxSize**

---

sizeWarning

protected int **sizeWarning**

---

nameMethod

protected java.lang.String **nameMethod**

---

b2cLruElement

protected B2CLRUElement **b2cLruElement**

Constructor Detail

B2CElementCache

```
public B2CElementCache(int size)
```

Method Detail

clearCache

```
public void clearCache()
```

---

getElement

```
public B2CElement getElement(java.lang.Object managerClass,
                             int aKey)
                      throws B2CAdvancedElementException,
                             B2CCacheException
```

Throws:
B2CAdvancedElementException
B2CCacheException

---

b2c.framework.cache

Class B2CLRUElement

```
java.lang.Object
  └ b2c.framework.cache.B2CLRUElement
```

---

public class **B2CLRUElement**
extends java.lang.Object

Field Summary

protected java.util.TreeMap	**KeyMap**

Constructor Summary

B2CLRUElement()	

Method Summary

void	**addHint**(int aKey)
int	**getKeyLeastMatched**()
int	**getLruElement**()
void	**removeHint**(int aKey)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

KeyMap

protected java.util.TreeMap **KeyMap**

Constructor Detail

B2CLRUElement

public **B2CLRUElement**()

Method Detail

getKeyLeastMatched

public int **getKeyLeastMatched**()

---

addHint

public void **addHint**(int aKey)

---

removeHint

```
public void removeHint(int aKey)
```

---

getLruElement

```
public int getLruElement()
```

---

b2c.framework.cache

Class B2CLRUObject

```
java.lang.Object
  └b2c.framework.basicelement.B2CObjectElementary
      └b2c.framework.cache.B2CLRUObject
```

All Implemented Interfaces:

java.lang.Cloneable

---

public class **B2CLRUObject**
extends B2CObjectElementary

Field Summary	
int	**matchCount**
Fields inherited from class b2c.framework.basicelement.B2CObjectElementary	
key	

Constructor Summary	
B2CLRUObject()	
Methods inherited from class b2c.framework.basicelement.B2CObjectElementary	
clone	
Methods inherited from class java.lang.Object	
equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Field Detail

matchCount

```
public int matchCount
```

Constructor Detail

B2CLRUObject

```
public B2CLRUObject()
```

---

b2c.framework.cache

Class B2CProductCache

```
java.lang.Object
  └b2c.framework.cache.B2CElementCache
      └b2c.framework.cache.B2CProductCache
```

---

public class **B2CProductCache**
extends B2CElementCache

Field Summary

protected static B2CProductCache	**instaceProductCache**

Fields inherited from class b2c.framework.cache.B2CElementCache

b2cLruElement, elementCacheMap, maxSize, nameMethod, sizeWarning

Constructor Summary

B2CProductCache(int size)	

Method Summary

static B2CProductCache	**getInstance**()
static B2CProductCache	**getInstance**(int size)

Methods inherited from class b2c.framework.cache.B2CElementCache
clearCache, getElement
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instaceProductCache

```
protected static B2CProductCache instaceProductCache
```

Constructor Detail

B2CProductCache

```
public B2CProductCache(int size)
```

Method Detail

getInstance

```
public static B2CProductCache getInstance()
                                 throws B2CCacheException
```

Throws:
B2CCacheException

---

getInstance

```
public static B2CProductCache getInstance(int size)
```

---

b2c.framework.cache

Class B2CPromotionCache

```
java.lang.Object
  └b2c.framework.cache.B2CElementCache
      └b2c.framework.cache.B2CPromotionCache
```

---

public class **B2CPromotionCache**
extends B2CElementCache

Field Summary

protected static B2CPromotionCache	**instacePromotionCache**

Fields inherited from class b2c.framework.cache.B2CElementCache

b2cLruElement, elementCacheMap, maxSize, nameMethod, sizeWarning

Constructor Summary

B2CPromotionCache(int size)	

Method Summary

static B2CPromotionCache	**getInstance**()
static B2CPromotionCache	**getInstance**(int size)

Methods inherited from class b2c.framework.cache.B2CElementCache

clearCache, getElement

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instacePromotionCache

```
protected static B2CPromotionCache instacePromotionCache
```

Constructor Detail

B2CPromotionCache

```
public B2CPromotionCache(int size)
```

Method Detail

getInstance

```
public static B2CPromotionCache getInstance()
                                    throws B2CCacheException
```

Throws:
B2CCacheException

---

getInstance

```
public static B2CPromotionCache getInstance(int size)
```

Bibliografía

[CL2002] CRAIG LARMAN *Applying UML and Patterns, 2nd edition*. Prentice Hall PTR, 2002

[JBR99] JACOBSON, I., BOOCH, G., AND RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999

[NOAY1999] NABIL R.ADAM, OKTAY DOGRAMACI, ARYYA GANGOPADHYAY, YELENA YESHA. *Electronic Commerce. Technical, Business and Legal Issues*. Prentice Hall PTR, 1999.

[RUP1998] RATIONAL® *Rational Unified Process Best Practices for Software Development Teams*. Rational Software White Paper, TP026B, Rev 11/01.

[SUN2003] THE SOURCE FOR JAVA TECHNOLOGY® *JAVA SERVLET TECHNOLOGY. The Power Behind the Server*. http://java.sun.com/products/servlet/index.html

[HTTP1999] COPYRIGHT (C) THE INTERNET SOCIETY (1999). ALL RIGHTS RESERVED. *Hypertext Transfer Protocol – HTTP/1.1*. Request for Comments: 2616.

[JSP98] JASON HUNTER WITH WILLIAM CRAWFORD. *Java ^{TM} Servlet Programming*. Copyright ©1998 O'Reilly & Associates, Inc. All rights reserved.