Primer Certamen

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Primera parte, **sin apuntes** (30 minutos; 30 puntos):

1.- Responda brevemente y entregue esta hoja con su nombre.

a)

Considere las siguientes clases definidas

```
public abstract class Person{
    public void sayHello() {
        System.out.println("Hello Person!");
    }
    public abstract void sayBye();
}

public class Student extends Person{
    public void sayHello() {
        System.out.println("Hello Student!");
    }
    public void sayBye() {
        System.out.println("Bye Student");
    }
}
```

Suponiendo que el siguiente bloque de código se encuentra definido dentro de un método main

```
...
Person p = new Person();
Student s = (Student) p;
s.sayHello();
s.sayBye();
...
```

Indique cuál es el resultado de compilar y ejecutar este programa. Justifique.

Este programa lanzará un error, pues se intenta crear una instancia de una clase abstracta, lo cual no es posible.

b) Explique con sus propias palabras la diferencia entre un atributo de clase con el modificador **static** y un atributo de clase con el modificador **final**. ¿Qué efecto produciría en un atributo de clase que este sea definido como final y static **al mismo tiempo**?

Un atributo definido como static es un atributo que se encuentra almacenado en memoria estática (y por tanto disponible durante toda la ejecución del programa) y es compartido por todas las instancias de una clase. El estar almacenado en memoria estática no le impide poder modificar su valor.
Un atributo definido como final es un atributo cuyo valor almacenado no puede ser modificado. Sin embargo, será almacenado como cualquier otro atributo de clase y, por tanto, quedará ligado a una instancia de clase.
Definir un atributo como final y static al mismo tiempo, producirá que el atributo se comparta entre todas las instancias de la clase y que su valor almacenado no pueda ser modificado. Es decir, sería una constante de clase.

c) Considere el siguiente código:

```
interface OperacionDeString {
   String operacion(String a, String b);
}
class Concatenador implements OperacionDeString  {
   public String operacion( String a, String b) {
     return a+b;
   }
}
class Extractor implements OperacionDeString {
   public String operacion (String a, String b) {
     return a.replace(b,""); // elimina de a todo substring b.
    }
}
public class Main {
  public static void main (String [] args) {
    Concatenador concatena = new Concatenador();    // [1]
    Extractor extraiga = new Extractor();                  // [2]
    System.out.println("Hola concatenado con Jóvenes =" + concatena.operacion("Hola","Jóvenes"));
    System.out.println("ola de mar="+extraiga.operacion("Hola","H"));
}
```

Reescriba el método main de la clase Main usando expresiones lambda en lugar de instancias de las clases Concatenador y Extractor de líneas 1 y 2.

```
public class Main {
  public static void main (String [] args) {
    OperacionDeString concatena = (a,b) → a+b;
    OperacionDeString extraiga = (a,b) → a.replace(b,"");
    System.out.println("Hola concatenado con Jóvenes =" + concatena.operacion("Hola","Jóvenes"));
    System.out.println("ola de mar="+extraiga.operacion("Hola","H"));
}
```

d) Considere:

```
class A {
   private int i;
   private Date dob;
   //  métodos
  }
   class B extends A{
    private String profesion;
    //  métodos
    }
```

Complete las clases A y B para conseguir copia profunda de instancias de B

```
class A implements Cloneable {
   private int i;
   private Date dob;
   public A clone () {
      A a = (A) super.clone();
      a.dob = dob.clone();
      return a;
   }
   // otros métodos
}

class B extends A implements Cloneable {
    private String profesion;
    public B clone() {
      B b = (B) super.clone();
      return b;
    }
    //  otros métodos
    }
```

e) ¿Qué es un event Handler?, y dé un ejemplo de su uso para el caso de un botón JavaFX

Un Event Handler es una clase que permite ejecutar acciones producidas por la detección de un evento. Para definir esta clase es necesario implementar una interfaz de tipo EventHandler.
El uso de un manejador de eventos permite al desarrollador proveer interfaces que respondan a eventos gatillados por la interacción del usuario con la interfaz. Un ejemplo de esto es un botón en una interfaz JavaFX:

```
@Override
public void start (Stage stage) {
Button button = new Button();
button.setText("Click Me!");
button.setOnAction (event -> System.out.println("Hello World!"));

StackPane root = new StackPane();
root.getChildren().add(button);

stage.setScene(new Scene(root, 300, 250));
stage.show();
}
```

f) Mencione y explique 2 desafíos para los desarrolladores en el contexto del desarrollo de aplicaciones

- Múltiples tamaños de pantalla y resoluciones: Las aplicaciones deben poder visualizarse correctamente en distintos tamaños y configuraciones de pantalla
- Desempeño: Las aplicaciones deben ser responsivas y con transiciones suaves
- Seguridad: Mantener seguro tanto el código fuente así como la información de los usuarios
- Compatibilidad: Debe ser capaz de correr en plataformas antiguas

Diseño y programación orientada a objetos - ELO-329

Primer semestre 2021 - Campus Casa Central - Campus San Joaquín

---

Información sobre la entrega

Debe subir a aula un solo archivo comprimido con el código que haya realizado. El nombre del archivo debe tener el siguiente formato: *ROL_NOMBRE_APELLIDO.zip*.

Certamen 1

Pregunta de Desarrollo

- Descargue el código de ayuda desde el siguiente repositorio gitlab: https://gitlab.com/francisco.cabezas/elo-329-c1-example

Dada la contingencia, se hace estrictamente necesario que hayan servicios que puedan permitir comercio electrónico.
Todo comercio electrónico dispone de un inventario de productos, cuyos atributos van desde el *SKU* o *id*, hasta el precio. Además, los usuarios del sitio pueden realizar búsqueda de productos y agregar sus productos favoritos a una lista personal, más bien conocida como *carrito de compras*.
El desafío del día de hoy será construir un servicio que permita realizar operaciones básicas de compra. Para ello, se ha dispuesto de un código inicial que permite leer un archivo de entrada en formato *csv*. Este archivo tiene 1000 registros, en donde cada fila tiene los siguientes atributos:

id (String)	timestamp (String)	category (String)	product (String)

brand (String)	size (String)	price (int)	availability (String)

En donde cada ítem tiene entre paréntesis el tipo de dato que contiene. Actualmente, el código de ejemplo puede leer el archivo, y guardar cada fila como un `String` dentro de un `ArrayList`.
Se pide lo siguiente:

a) **(25 puntos)** Implemente la funcionalidad de búsqueda que permita buscar según lo que se exprese en los parámetros de entrada, de la siguiente forma:

Buscador por marca

```
> java InventoryManager -s -brand Parle
```

El ejemplo corresponde a una búsqueda (`-s`) por marca (`-brand`), para lo cual deberá buscar aquellos productos de la marca ingresada en el tercer atributo. Lo anterior deberá imprimir por pantalla los atributos *id*, *product*, *brand* y *price*, por ejemplo:

```
> Se encontró 1 registro:
> Item Id: 8687c5317c6629f89fe62e0233563246, product: Parle Melody Chocolaty
Toffee  (195.5 g), brand: Parle, price: 50
```

En el caso de no encontrar registros, imprimir:

```
> No se han encontrado registros
```

Punto importante: Solo debe devolver aquellos productos con disponibilidad, es decir availability = TRUE

Respuesta

Es importante entender que podemos modelar cada fila como un Producto con sus atributos, en tanto la salida de la lectura del archivo será una lista de productos. Ello permite que las operaciones sobre el producto sean más simples. También podemos encapsular la impresión del mensaje en pantalla con formato como un servicio del producto.

Para poder separar cada atributo (el cual está separado por punto y coma en el archivo), podemos usar la función split como sigue:

```
String[] row = fileRow.split(";");
```

Esto nos devolverá un arreglo con cada parémtro de la fila. Entonces, podemos crear un constructor para la clase Producto y pasarle una fila, dado eso, sería el constructor el responsable de inicializar el producto:

```
public Product (String fileRow) {
  String[] row = fileRow.split(";");
  this.id = row[0];
  this.timestamp = row[1];
  this.category = row[2];
  this.product = row[3];
  this.brand = row[4];
  this.size = row[5];
  this.availability = row[7];

  try {
    this.price = Integer.parseInt(row[6]);
  } catch (NumberFormatException e) {
    this.price = 0;
  }
}
```

Por omisión el valor de cada atributo será un string, pero en el caso del precio, necesitamos hacer in cast. Si el cast resulta erróneo, se activará la excepción, siendo plausible asumir que el precio es cero en ese caso.

(10 puntos, si lo hace de otra forma pero cumple con crear una clase con datos, también vale el puntaje)

También podemos definir la impresión del mensaje formateado como un servicio del producto, a través de un método, como sigue:

```
public void printFormatted () {
    System.out.println("Item Id: " + id + ", product: " + product +
        ", brand: " + brand + ", price: " + price + ", availability:
        " + availability);
}
```

Y la disponibilidad también:

```
public boolean getAvailability () {
  if (availability.equals("TRUE")) {
    return true;
  } else {
    return false;
  }
}
```

El cambio del tipo de la lista lo podría haber hecho en cualquier lugar, pero la sugerencia es hacerlo en quien es responsable de poblar la estructura de datos con datos, valga la redundancia. Dado eso, la recomendación es refactorizar el método getRowData() de la clase FileManager:

```
public ArrayList<Product> getRowData() throws FileNotFoundException {
  Scanner fileReader = openFileScanner();
  while (fileReader.hasNextLine()) {
    String data = fileReader.nextLine();
    rowData.add(new Product(data));
  }
  fileReader.close();

  return rowData;
}
```

(5 puntos, si lo hace en otro punto del programa, también vale el puntaje)

Si lograba hacer lo anterior, el resto era bastante más sencillo, puesto que para generar la búsqueda, puede generar una clase Search que reciba como input la lista de productos, como sigue:

```
public Search (ArrayList<Product> rowData) {
  this.rowData = rowData;
}
```

Dado lo anterior, implementar la búsqueda se hace mucho más fácil:

```
public void performSearch (String brand) {
  for (Product product : rowData) {
    // Recuerde que availability debe ser igual a TRUE
    if (product.getBrand().equals(brand) && product.getAvailability()) {
      System.out.println("Se encontro 1 registro:");
      product.printFormatted();
      return;
    }
  }
  System.out.println("No se han encontrado registros");
}
```

(10 puntos)

b) **(25 puntos)** Implemente funcionalidad para ordenar según precio (de Mayor a Menor):

```
> java InventoryManager -o
```

Los ejemplos corresponden a un ordenamiento (-o) según precios. La salida debería ser la siguiente:

```
> Item Id: 0633d9fd9a3271730fae687f105c7a3a, product: B Natural Dakshin Pink
Guava  (750 ml), brand: Parle, price: 95, availability: TRUE

> Item Id: 8687c5317c6629f89fe62e0233563246, product: Parle Melody Chocolaty
Toffee  (195.5g), brand: Parle, price: 50, availability: TRUE

...
```

Respuesta

Teniendo definida la lista de productos de la pregunta anterior, entonces podemos definir una clase Sort que reciba esta lista de productos como parámetro y permita ordenar:

```
public class Sort {
   private ArrayList<Product> rowData;

   public Sort (ArrayList<Product> rowData) {
      this.rowData = rowData;
   }

   public void performSort () {
      Collections.sort(rowData, new Comparator<Product>(){
         @Override
         public int compare(Product p1, Product p2) {
            return p2.getPrice() - p1.getPrice();
         }

      });
   }
}
```

Para ordenar, nos podemos apoyar en la clase Collections, la cual tiene un método sort que permite ordenar una lista. Pero dado que nuestro objeto es customizado, se hace necesario implementar una sobrecarga del método, en donde pasamos la interfaz Comparator, y realizamos la comparación según el enunciado, en donde debemos ordenar desde el mayor precio (p2) al menor precio (p1).

(25 puntos, si ordena usando otro algoritmo, cumpliendo con el requerimiento del enunciado, también valen los puntos)

Luego para imprimir, podemos hacer:

```
Sort sort = new Sort(rowData);
sort.performSort();

for (Product product : rowData) {
   product.printFormatted();
}
```

c) **(20 puntos)** Implemente una vista usando JavaFX, en la cual exista un input de texto que actúe como buscador de marca, y un botón *Buscar*. Los resultados los devolverá por consola, en el mismo formato que a).

Respuesta

Dado que solo deseamos imprimir en pantalla, nos podemos apoyar totalmente en lo que hicimos en la pregunta a):

```
public class SearchBar extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        FileManager fileManager = new FileManager("data_example_csv.csv");
        ArrayList<Product> rowData = fileManager.getRowData();
        Search search = new Search(rowData);

        GridPane grid = createGrid();
        final TextField searchField = new TextField();

        // Input de busqueda
        createTextField(grid, searchField);

        // Boton buscar
        createButton(grid, searchField, search);

        primaryStage.setTitle("Barra de busqueda");
        primaryStage.setScene(new Scene(grid, 300, 275));
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }

    private GridPane createGrid () {
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(5);
        grid.setHgap(5);

        return grid;
    }

    private void createTextField (GridPane grid, TextField searchField) {
        searchField.setPromptText("Ingrese marca");
        searchField.setPrefColumnCount(10);

        GridPane.setConstraints(searchField, 0, 0);
        grid.getChildren().add(searchField);
    }

    private void createButton (GridPane grid, TextField searchField, Search search) {
        Button searchButton = new Button("Buscar");
        setButtonAction(searchButton, searchField, search);

        GridPane.setConstraints(searchButton, 1, 0);
        grid.getChildren().add(searchButton);
    }
```

```
    private void setButtonAction (Button searchButton, TextField searchField, Search s
        searchButton.setOnAction(actionEvent -> {
            String searchText = searchField.getText();
            search.performSearch(searchText);
        });
    }
}
```

(20 puntos, si utiliza otros componentes, pero cumpliendo con implementar un buscador, también valen los puntos)

Se dará puntaje también por orden y buen uso de modelamiento orientado a objetos

Hint

En IntelliJ también puede usar argumentos para arrancar su programa, vea la siguiente documentación: https://users.drew.edu/bburd/JavaForDummies6/CommandLineIntelliJIDEA.pdf