

Introducción al Lenguaje Java

ELO329: Diseño y Programación Orientados a
Objetos

Mi Primer Programa en Java (1/3)

- Como en C y C++, en Java todo programa parte por la “función” main. Como en Java no existen funciones “al aire” como en los lenguajes no orientados a objetos, main debe ser un método de alguna clase.
- Corolarios:
 - Todo programa Java debe tener al menos una clase.
 - Todo programa Java debe tener el método main definido en alguna clase.
- Ver FirstSample.java

Nombre de archivo => FirstSample.java

```
public class FirstSample {  
    public static void main(String[ ] args) {  
        System.out.println("Hello, Sansanos!");  
    }  
}
```

Mi Primer Programa en Java

(2/3)

```
public class FirstSample{  
    public static void main(String[ ] args){  
        System.out.println("Hello, Sansanos!");  
    }  
}
```

- Se compila así: `$javac FirstSample.java`
- En Java el nombre de un archivo fuente y su clase visible desde fuera deben llamarse igual (archivo agrega extensión `.java`).
- Se usa así para identificar el archivo que contiene la clase buscada. ¿Qué pasa si los fuentes están en otra carpeta? Ya viene ...
- En tiempo de ejecución en general debemos invocar los métodos de un objeto usando el nombre o referencia al objeto. (main es una excepción)
 - Ej. `Punto p = new Punto(); // para instanciar un punto`
 - `p.equals(otroPunto); // para invocar un método.`
- Un **método** también puede ser **estático**, lo acompaña el calificar `static`, éste puede ser invocado directamente usando el nombre de la clase. Los métodos estáticos pueden entenderse como métodos de la clase en general y no de un objeto específico; por ejemplo, funciones matemáticas como `abs()`, `valor absoluto`, están en clase `Math`.

Mi Primer Programa en Java (3/3)

```
public class FirstSample{  
    public static void main(String[ ] args){  
        System.out.println("Hello, Sansanos!");  
    }  
}
```

- Al ejecutar: `$ java FirstSample`, la máquina virtual java busca el archivo `FirstSample.class` e invoca `FirstSample.main(<aquí pone los argumentos de la línea de comandos>)`
- Así como una clase tiene métodos estáticos (son lo más parecido a una función tipo C), también puede tener atributos estáticos. Este es el caso del atributo de nombre `out` de la clase `System`.
- `System.out` es un objeto al cual podemos invocar el método `println(String s)`.
- Ver documentación java. Ver tecnología en Java SE.

Sobre los nombres de archivos fuentes (.java)

- Javac accede y compila el archivo dado como argumento. Se generarán tantos archivos .class como clases hayan sido definidas en él.
- Para cada clase A instanciada en el programa, javac busca el A.class correspondiente. Si no existe, busca y compila el archivo A.java que debe estar en algún directorio listado en la variable de ambiente CLASSPATH (por omisión .). Si A.class existe pero es más antiguo que A.java, vuelve a compilar A.java.
- Al ejecutar un programa P, java buscar P.class e inicia todo desde su método main. Si el programa hace referencia a otras clases, sus correspondientes .class deben estar en algún directorio listado en CLASSPATH.

Instalación

- En este curso usaremos la Edición Estándar (SE). Otras versiones son: Enterprise Edition (Java EE) y la Micro Edition (Java ME).
- Baje el Java Development Kit (JDK) desde <http://java.oracle.com>
- En Windows verificar que se haya incorporado la ruta de la carpeta del compilador y de la máquina virtual al PATH del sistema. Es algo del tipo:
 - Control Panel -> System -> Environment. Avanzar hasta las variables de usuario y buscar la variable PATH. Agregar el directorio `jdk\bin` de su instalación al comienzo del PATH. Ej. `PATH=c:\jdk\bin; <siguen las otras rutas>`.
- Instalación en UNIX:
 - Incorporar la ruta de los ejecutables de su instalación java al path del sistema. Esto se hace editando el archivo `.bashrc` o `.bashrc_profile`.
 - Por ejemplo, agregando: `export PATH=/usr/local/jdk/bin:$PATH`

Ejecución de programas Java

- Para ejecutar programas en Windows, lance una consola (ejecutando cmd) y corra en ésta los comandos de compilación y ejecución.
- En Linux lance una consola y ejecute los comandos para compilar y ejecutar su programa.
- Obviamente, esto se puede hacer más simple usando un ambiente de desarrollo integrado (IDE: Integrated Development Environment), como los gratuitos Jgrasp, Eclipse, NetBeans, etc.

Aspectos básicos: Tipos primitivos (8, no son objetos, todo lo demás sí es)

- Booleano
 - boolean
 - Puede tomar los valores true y false
- Enteros (4)
 - int 4 bytes Ej: 24, 0xFA (base 16), 015 (base 8)
 - short 2 bytes
 - long 8 bytes Ej: 400L
 - byte 1 byte
- Punto flotante (2)
 - float 4 bytes Ej: 3.14F (6-7 dígitos signif.)
 - double 8 bytes Ej: 3.14D (15 dígitos signif.)

Tipos primitivos (cont.)

- Carácter: char
 - Usa una codificación conocida como Unicode
 - Usa dos bytes (distinto de ASCII que usa 7 bits)
 - Diseñado para internacionalización
 - Comillas simples: 'a', 'A', '!', '1', ... (los Strings usan “”)
 - Forma hexadecimal '\u0008' (Unicode backspace)
 - El byte menos significativo corresponde al “ASCII” de 8 bits.
 - Los hay no visibles : Ej:

• '\b'	backspace
• '\t'	tab
• '\n'	linefeed
• '\r'	return

• '\"'	double quote
• '\''	single quote
• '\\'	el mismo backslash!

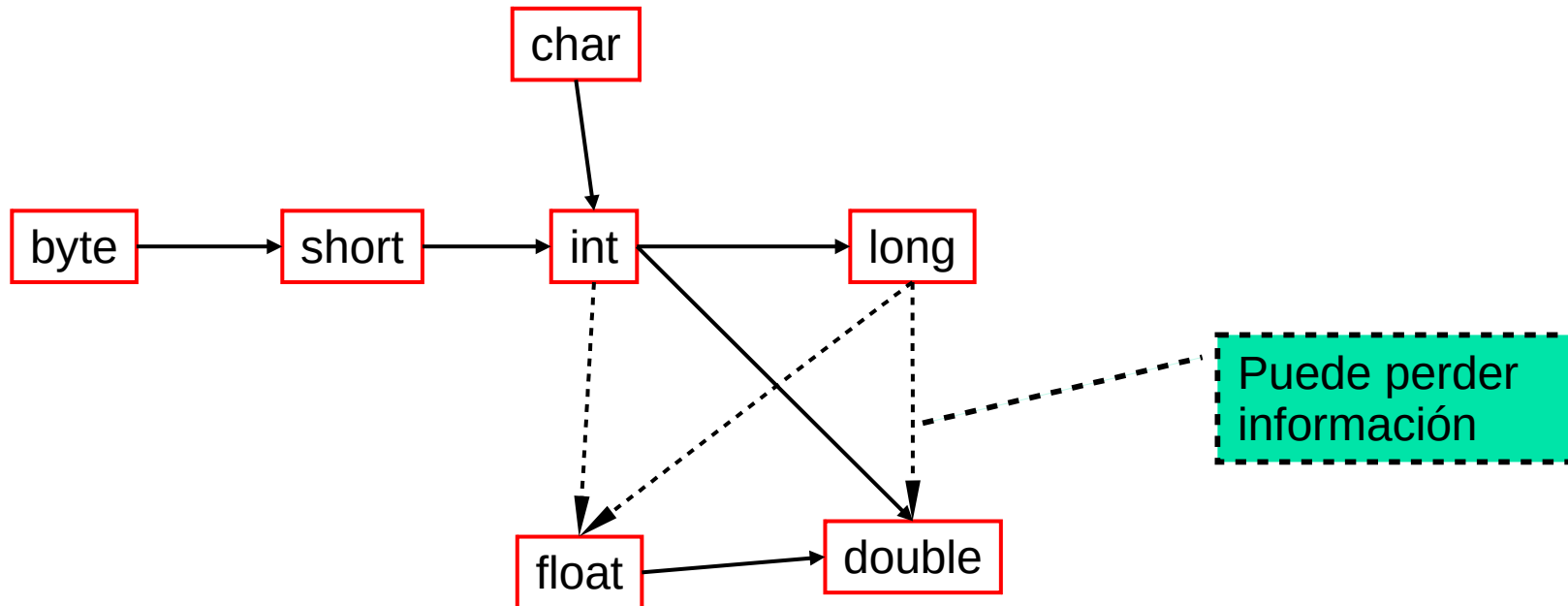
Constantes

- Se usa la palabra reservada **final**
- Ej: public **final** float CM_PER_INCH=2.54;
- Si deseamos crear sólo una instancia de esta constante para todos los objetos de una clase, usamos:

```
public class Constante {  
    public static final float CM_PER_INCH=2.54;  
    ...  
}
```

- Para acceder al valor: Constante.CM_PER_INCH

Cambios de tipo automáticos



Operadores listados por su precedencia

A la derecha su orden de evaluación

Ej. Para +,
`a=3+4+5;` se interpreta como
`a=(3+4)+5`
 En esta caso se aplicó asociatividad de izquierda a derecha.

Para asignación
`a=b=c=4;` se interpreta como
`a=(b=(c=4));`
 En este caso se aplicó asociatividad de derecha a izquierda.

[] . () (invocación)	-->
! ~ ++ -- + - (<tipo o clase>) new	<--
* / %	-->
+ -	-->
<< >> >>>	-->
< <= > >= instanceof	-->
== !=	-->
&	-->
^	-->
	-->
&&	-->
	-->
? :	<--
= += -= *= /= %= &= = ^= <<= >>= >>>=	<--

String

- Java tiene una clase pre-definida llamada String.
- Todos los string son objetos y su comportamiento está dado por la clase String. Ver en documentación las operaciones para Strings.
- La clase String tiene un manejo especial en el lenguaje:
 - El operador + concatena strings. Si uno de los operandos no es string, Java lo convierte a string y luego lo concatena.
Ej: `int trece=13;`
`String estacion = "Canal"+trece; // asigna "Canal13"`
- Igual que para cualquier par de objetos, para comparar dos strings, usar el método equals. De otra manera comparamos referencias.
- El identificador de todo objeto es una referencia al objeto ("dirección"), no el objeto mismo.



Entrada y Salida

- La salida de texto por consola es simple haciendo uso del objeto `System.out`. Es decir atributo `out` de la clase `System`.
- Hasta la versión 1.4 la entrada de texto era bastante engorrosa. Esto se simplifica en V1.5 (o Java versión 5) con la clase `Scanner`.
- Formas gráficas de entrada y salida se verán después.
- Las clases principales a estudiar son:
 - `Java.io.PrintStream` (desde Java 1.0), y
 - `Java.util.Scanner` (desde Java 1.5)

Salida de datos simple a consola

- Desde la versión 1.0 de Java existe la clase `java.io.PrintStream`. **System.out** es instancia de ésta.
- Define métodos para la salida de stream vía un buffer. Esto es, los caracteres son puestos en memoria temporalmente antes de salir a consola.
- Algunos métodos son:
 - `print(Object o)`: invoca método `toString()` de `o` e imprime resultado.
 - `print(String s)`: imprime string `s`.
 - `print(tipo_básico b)`: imprime el valor de `b`
 - `println(String s)`: Imprime `s` seguido de `newline`.

Entrada de datos simples por consola

- El objeto especial para efectuar entrada de datos es `System.in`; sin embargo, éste **no ofrece** métodos cómodos (es instancia de `InputStream`).
- Para facilitar la entrada de datos se creó, a partir de la versión 1.5, la clase **Scanner**, en paquete `java.util`, la cual trabaja como envoltorio o recubriendo (`wrapper`) de la clase `InputStream`.
- `Scanner` tiene varios métodos convenientes para la entrada de datos.
- Ver ejemplo: `InputExample.java`

Métodos de `Java.util.Scanner`

- Ver documentación en `manuales.elo.utfsm.cl`
- Revisar métodos:
 - `hasNext()`: hay más datos en entrada?
 - `next()`: retorna próximo token.
 - `hasNextType()`: `Type` es un tipo básico, verdadero si hay tal dato a continuación. `Type` puede ser `Boolean`, `Byte`, `Double`, `Float`, `Int`, `Long` o `Short`.
 - `nextType()`: retorna el dato del tipo `Type` a continuación.
 - Ver también: `hasNextLine()`, `nextLine()`; `findInLine(String s)`;

Entrada de datos simples vía interfaz gráfica

- Otra forma de ingresar datos es vía la clase `JOptionPane`, en particular vía uno de sus métodos:
`JOptionPane.showInputDialog(promptString);`
este llamado retorna el string ingresado por el usuario.
- Ver ejemplo: `InputTest.java`

Sentencias (esto lo deben estudiar por su cuenta) nosotros saltamos lámina #23.

- La condición: if
- if(exp) statement1;
else statement2;
- if (a>b) x = a;
else x = b;
- *else* es opcional como en:
if (x[i] > max) max = x[i];

Sentencias - Lazos

- La repetición: while, el do ... while, y for
- Lazo “while”
while(exp) statement1;
while(exp) { statements; }
- while (a>b) a = x[i++];
while (x < 0) {
 x = z.getX(...);
 y = y + x;
}
- “while” permite evitar el viaje al bloque interno

Sentencias - Lazos

- Lazo “do”
do statement; while(exp);
do { statements; } while(exp);
- do a = x[i++]; while(a>z);
- do {
 x = z.getX(...);
 y = y + x;
} while (x > 0);
- “do” implica al menos un viaje al bloque interno.

Sentencias - Lazos

- for
for(exp1; exp2; exp3) { s; }
- equivalente a:
exp1;
while (exp2)
 { s; exp3; }
- for(int k=0; k<n; k++) { s; } // en este caso k no está definida después.
equivale a:
int k=0;
while(k<n) { s; k++; } // aquí k sí existe aún después del loop.
- Podemos poner cualquier expresión en las partes del lazo for, pero es buena práctica sólo inicializar, probar condición de término y actualizar la variable de control.
- Patrón estándar para n iteraciones!

Sentencias - Lazos

- **for mejorado** diseñado para el manejo de arreglos y Colecciones
- Las colecciones son clases ya definidas en Java que permiten agrupar varios objetos como en listas, colas, etc.
- La versión mejorada del for permite hacer lazos más compactos y fáciles de leer. Su estructura se desprende el siguiente ejemplo:

```
class EnhancedForDemo {  
    public static void main(String[] args){  
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};  
        for (int item : numbers) {  
            System.out.println("Count is: " + item);  
        }  
    }  
}
```

Sentencias - switch

- Condicional múltiple
- `switch(exp1) {`
 - `case x1: s1; break; // si no ponemos break, sigue s2.`
 - `case x2: s2; break;`
 - `default: s3;`
 - `}`
- Ejemplo:
 - `switch(x) {`
 - `case 1: y = a; break;`
 - `case 2: y = b; break;`
 - `default: y = c;`
 - `}`

Break y continue

- La sentencia **break** permite salir de un lazo de repetición sin terminarlo (además de su uso en switch). Continúa en instrucción siguiente al lazo.
- **Break** también puede ser usada en conjunto con un rótulo para salir fuera de cualquier bloque. El rótulo se pone inmediatamente antes del bloque en cuestión. Así podemos salir de varios loops anidados.

rotulo:

```
while(..){  
    ... for (..) {  
        ... break rotulo; //  
    }  
}
```

} // esta idea se puede aplicar a cualquier bloque definido con {...}

- La sentencia **continue** transfiere el control de flujo al encabezado del lazo más interno; es decir, a su condición de continuidad. Análogamente podemos usar continue con un rótulo como break.

Clases para tipos de datos primitivos

- Crean objetos para los tipos estándares.
 - Éstas son: Boolean, Byte, Short, Integer, Long, Character, Float, Double
 - Se encuentran en paquete java.lang
- Un método importante en estas clases permite transformar un string que contiene números en un tipo básico (como atoi() en C).

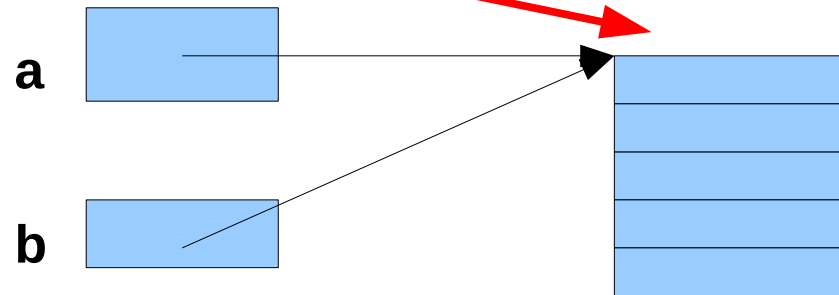
Ej: `int a = Integer.parseInt("3425");`
hace que **a** tome el valor 3425.
Se usó en ejemplo `InputTest.java`

Arreglos en Java

- Los arreglos almacenan una colección de valores de igual tipo, su acceso es vía un índice entero ($0 \leq \text{índice} < \text{max}$)
- Declaración de una variable arreglo de enteros:
 - `int [] a;` // hasta aquí sólo tenemos un identificador o nombre
 - `a = new int[100];` // recién ahora tenemos los datos
- Todos los arreglos son objetos de tamaño fijo, definidos en tiempo de ejecución.
- Además de sus datos, todos los arreglo tienen el atributo constante, **length**, el cual entrega el largo del arreglo.
- `int [] a = {3,5,7,11,17};` // creación con primeros valores.
- `int [] a = new int [20];` // sólo creación con valor cero
- `for (int i=0; i< a.length; i++)` // otra forma de inicializarlo
`a[i] = i;`

Arreglos son objetos

- `Int [] a = new int [5];`
- `Int [] b=a;`



- Ojo los cambios en **b**, afectarán **a** y viceversa.
- `a[2]=3; // hará que b[2] sea 3.`

Arreglos Multidimensionales

- `int [][] matriz = {{1,2,3},{4,5,6},{7,8,9}};`
- Acceso `int a = matriz[1,2];` // será el 6 porque parten de 0
- Los arreglos multidimensionales son en realidad arreglos de arreglos. Como curiosidad, podrían no ser cuadrados.
- `int [][] triangular = new int [5][];`
- `for (int n=0; n<triangular.length; n++) {
 triangular[n]=new int[n+1];
 for (int j=0; j < triangular[n].length; j++)
 triangular[n][j] = n+j;`

