Introducción a Java

Agustín J. González

ELO329

¿Por qué Java?

- Es probablemente uno de los buenos lenguajes que podemos aprender hoy por lo siguiente:
  - Java es pequeño: Hay versiones que corren en pequeños dispositivos portátiles.
  - Java es Orientado al Objeto
  - Java es “compatible” con Internet: Corre en navegadores, sus programas pueden ser transmitidos y correr en “cualquier” computador.
  - Es de propósito general
  - Es independiente de la plataforma
  - Tiene mucha bibliotecas: para manejo gráfico, acceso a Internet, interfaces gráficas de usuarios (GUIs) etc..
- Surge por la necesidad de construir software para la electrónica de consumo (VCRs, TV, teléfonos,....) La dificultad de C y C++ fue el transporte a otros procesadores. Objetivo: crear programas pequeños, rápidos, transportables.

Tipos

- Tipos Numéricos:
  - Similares a C++ pero sin tipos “unsigned”.
  - Tipo Tamaño
  - byte 1 byte
  - boolean 1 byte
  - char 2 byte Unicode
  - short 2 byte
  - int 4 byte
  - long 8 byte
  - float 4 byte
  - double 8 byte
- Caracteres: Unicode incluyen ASCII (9 bit mas significatvos =0) y caracteres de otros lenguajes (Chino, Japonés..). Caracteres no ASCII se denotan por ‘\Uxxxx’.
- Tipo booleano: No hay conversión a int. Expresiones como
- if (x=0) ... Con x entero, no es un error en C++ pero si en Java.

Tipos Arreglos

- Tenemos el tipo T[] (arreglo de T). Los arreglos siempre se ubican en el heap.
  Ej: int [] numbers = new int [n];

  Obtenido a tiempo de ejecución
- Su tamaño no puede ser redefinido.
- No existe la forma: int numbers [6]; // sólo en C / C++
- El largo puede ser obtenido con:
  largo = numbers.length; // análogo a v.size() de vectores
  // es una variable de sólo lectura
- La semántica de acceso es referencia:
  int [] winning_numbers;
  winning_numbers = numbers;
  numbers [0]=13; // cambia ambas variables!!
- Si tratamos de acceder a un índice no válido, tenemos un error de ejecución.
- Java no soporta el uso del nombre como puntero al primer elemento. Esto puede conducir a errores de rango.

Tipos Vectores

- El arreglo en Java no crece baja demanda.
- La biblioteca java.util implementa la clase Vector la cual resuelve este problema.
- En C++ necesitábamos especificar el tipo vector <T>, en Java todos los tipos (excepto números y boolean) heredan de ***Object***. En particular arreglos y strings descienden de Object. Es así como en Java hay sólo una clase Vector que almacena Object.

  Ej: Vector v = new Vector();
  v.addElement (new Integer(10)); // hace crecer el vector en 1
  Integer a = (Integer) v.elementAt(0); // como Inetger a = v[0];
  v.setElementAt("Hello", 0); // => como v [0]="hello";

  Nota: en Java no se puede sobrecargar operadores !! Por ello se usan funciones para acceder a los elementos del vector.

Tipos Strings

- Java tiene un tipo String en su biblioteca.
- El operador + es el operador concatenación (como en C++).
- Si algún operando no es String, éste es convertido a string usando el método toString().

```
int errorNum;
Socket s;
....
String message="Error " + errorNum + ": " + s;
```

- Los string son inmutables!!! No pueden ser cambiados.

```
String saludo = "Hola ";
saludos [4]='!';     // Error !!
saludo = "Qué tal!!";   // OK.
```

- Mirar la documentación por operaciones sobre strings.
- Para string que sean editables (datos llegando de un stream, socket, etc) usamos la clase: StringBuffer.
- Métodos: equals() por ==, compareTo() por < <= > >=.

Clases

- Definición de clases
  Una clase típica es:
- class Employee {
  public Employee (String n, double s){
  _name = n;
  _salary = s;
  }
  public double salary() {
  return _salary;
  }
  public void raiseSalary(double percent) {
  _salary *= 1+percent/100;
  }
  private String _name;
  private double _salary;
  }

Clases: Objetos

- En java no se puede expresar la diferencia entre una función de acceso (accesor) y de cambio (mutador), las cuales en C++ se caracterizan por “const”.
- Objetos: Todos los objetos en Java se ubican en el heap.
  Employee harry = new Employee(“Harry Hacker”, 35000);
- En Java debemos invocar new para crear un nuevo objeto. Los nombres de variables, en lugar de objetos son en realidad referencias a objetos que creamos con new.
- En Java no necesitamos preocuparnos por manejo de memoria. Un tarea de “recolección de basura” es efectuada por la máquina virtual en forma periódica. Ésta recolecta los espacios no referenciados.(distinto a C++)
- En Java todas las variables locales deben ser inicializadas antes de su uso. De lo contrario el compilador envía un error.

Clases: Herencia

- La sintaxis para herencia es similar a la de C++.
- Class Manager **extends** Employee {
  // operaciones y campos de datos adicionales
  }
- En Java todas las clases derivan de una clase origen **Object**. No es preciso hacer esta declaración en forma explícita.
- Esto hace posible definir clases genéricas tales como Vector sin usar templates.
- Es posible asignar objetos de una clases deriva a una variable de la clase base (como en C++).
  Manager carl = new Manager(“Carl Craker”, 110000);
  Employee boss = carl;
- Podemos verificar si una variable es una referencia a un objeto derivado.
  If (boss **instanceof** Manager) {
  Manager m = (Manager) boss;
  ...
  }

Clases: Herencia

- En Java todas las funciones son invocadas en forma dinámica (dynamic binding o enlace dinámico). En C++ lo indicábamos en forma explícita con virtual. En Java si no queremos ligado o enlace dinámico, declaramos la función como final.

```
Class Employee {
  public final double salary()  { return _salary;}
...
}
```

- También podemos declarar una clase como final, así ninguna clase puede derivar de ésta. Si la clase es final, todas sus operaciones también lo son.

```
Final class StringBuffer {...}
```

- En Java una clase abstracta (abstract) es indicada con esta palabra reservada. Recordar que éstas no pueden ser instanciadas, sólo se usan como clases bases.

```
abstract class shape {...}
```

Clases: Constructores

- En Java no hay lista de iniciación como en C++.

```
Class Employee     {   // Java
  public Employee (String n, double s) {
    _name = n;
    _salary= s;
}
...
}
```

En C++

```
Employee::Employee(string n, double s)    // C++
:  _name(n),  _salary(s){
}
```

- Objetos en Java nunca contienen otros objetos, solo referencias a otros objetos. Est porque todo lo que existe son referencias a objetos creados con new.
- Todos los campos son dejados en 0 o null por defecto.
- Podemos llamar al constructor de la clase base, para ello usamos la palabra reservada **super**.

Clases: Constructores

- Class Manager extends Employee { // Java
  public Manager (String n, double s) {
  super (n,s); // debe ser lo PRIMERO en el constructor
  ...
  }
  }
- En Java no tenemos argumentos por defecto. Debemos crear tantos constructores como sea necesario y podemos llamar un constructor desde otro.
  Class Employee {
  public Employee (String n, double s) {
  _name = n;
  _salary = s;
  }
  public Employee() {
  this(“”, 0);
  }
  ..
  }

Clases: Constructores

- Orden seguido en la iniciación de variables:
  - Todos los campos son llevados a cero, falso o null.
  - Se llama a otro constructor si el constructor comienza con this.
  - El constructor de la case base es invocado, con los parámetros de sentencia super o sin parámetros si no hay sentencia super.
  - Los campos datos son inicializados si tienen valores asignados, y en el orden en que aparecen.
  - El cuerpo del constructor es ejecutado.

Clases: Funciones

- En Java todas las funciones deben ser operaciones de alguna clase.
- Si necesitamos crear operaciones no ligadas a algún objeto, las declaramos como estáticas (static).
- Parámetros:
  - Todos los parámetros son pasados por **valor**. No hay parámetros por referencia. => no podemos crear una función swap entre dos números como en C++.
  - Class Employee {

    ...

    public void swapSalary (double s) {

    double temp=_salary;

    _salary = s;

    s= temp; // no tiene efecto fuera de la función.

    }
  - Los objetos también son pasados por **valor**.
  - Class Employee {

    ...

    public void swapSalary (Employee b) {

    double temp=_salary;

    _salary = b._salary;

    b._salary= tem; // SI tiene efecto fuera de la función.

    b = new Employee(_name, _salary); // No tiene efecto fuera...

    }

Clases:Llamados a la clase base

- Se utiliza la palabra reservada super.
- Class ScalableText extends Text {
  public void scale (Point center, double s) {
  super.scale(center,s);
  _size *=s;
  }
  ..
  }
- En C++ no se puede usar esta sintaxis porque en C++ se puede heredar de más de una clase. En Java esto no se puede.
- Paquetes:
  - En Java el código está distribuido en diferentes paquetes.
  - Si necesitamos usar una clase particular podemos declarar:
  - java.util.Vector v = new java.util.Vector();
  - O podemos importar el paquete
  - import java.util.Vector;
  - y luego usar la forma corta: Vector v = new Vector();
  - Si queremos importar todas las clases de un paquete usamos:
  - import java.util.*;

Entrada y Salida

- Veremos archivos en detalle más adelante. Por ahora una introducción breve.
- Hay muchas clases para entrada y salida (~30)
- El archivo System.out es de la clase PrintStream.
- Éste se puede asociar a un archivo usando:
  PrintStream ps = new PrintStream(new FileOutputStream(“output.dat”));
- Luego usamos print y println para la salida.
  System.out.println(“Hello”);
  ps.print(n);
- El objeto System.in es del tipo InputStream. Esta clase sólo puede leer bytes individuales. Para leer string creamos un DataInputStream.
  DataInputStream stdin = new DataInputStream(System.in);
- Igualmente se puede abrir un archivo de disco.
  DataInputStream in = new DataInputStream(new
  FileInputStream(“input.dat”));
- Una operación interesante de DataInputStrean es readLine.

```
Int age;
try {
        System.out. Println(“Please enter your age: “);
        String s = stdin.readLine();
        age = Integer.parseInt(s);
} catch (exception e )  { age = 0;}
```

Trabajando con Java

- Creación programa: Con editor crear programa *.java (Lowercase.java)
  - Hacer uso de documentación accesible desde aldebaran.elo.utfsm.cl
- Compilación: vía el comando el línea
  $ javac Lowercase.java
- Ejecución:
  $java Lowercase
- Hay ambientes de trabajo más amigables para hacer estas tareas.