Lenguaje Java

Nicolás Serrano

Febrero 2020

Command prompt

- Es la ventana de comandos del anterior sistema operativo: MS-DOS
- Se pueden definir y ver las variables de entorno con el comando SET
- PATH es la variable de entorno que define la ruta en la que se buscan los comandos
  `Path=C:\Java\jdk\bin;C:\WINDOWS\system32;C:\WINDOWS`
- SET %variable% muestra el contenido de la variable

Variables

- Variables de tipos primitivos
- Variables de referencia
  - Arrays
  - Objetos
- Variables miembros
  - Static
  - de objetos
- Variables locales
  - de un método
  - o de un bloque {}

Variables: tipos primitivos

- **boolean** 1 byte. Valores true y false
- **char** 2 bytes. Unicode . Comprende el código ASCII.
- **byte** 1 byte. Entero entre -128 y 127
- **short** 2 bytes. Entero entre -32768 y 32767
- **int** 4 bytes. Entero entre -2.147.483.648 y 2.147.483.647
- **long** 8 bytes. Entero entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
- **float** 4 bytes. Real (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
- **double** 8 bytes. Real (15 cifras decimales equivalentes). De -1.7976931348623 2E308 a -4.940656458412 47E-324 y de 4.940656458412 47E-324 a 1.7976931348623 2E308

Ejemplo de tipos enteros

```
class num {
    public static void main(String[] args)  {
      short num_s = 1;
      int num_i = 1;
      long num_l = 1;
      for (int i=1;i<70 ;i++ ) {
          num_s *= 2;
          num_i *= 2;
          num_l *= 2;
          System.out.println("i: " + i
            + " \t\tnum_s: “ + num_s
            + " \t\tnum_i: " + num_i
            + " \t\tnum_l: " + num_l);
      }
      System.out.println("End");
   }
}
```

Declaración de variables

Ejemplos de declaración e inicialización de variables:

```
int x;                  // Declaración de la variable primitiva x. Se inicializa a 0
int y = 5;              // Declaración de la variable primitiva y. Se inicializa a 5

MyClass unaRef;                 // Declaración de una referencia a un objeto MyClass.
                                // Se inicializa a null
unaRef = new MyClass();         // La referencia "apunta" al nuevo objeto creado
                                // Se ha utilizado el constructor por defecto
MyClass segundaRef = unaRef;    // Declaración de una referencia a un objeto MyClass.
                                // Se inicializa al mismo valor que unaRef
int [] vector;                  // Declaración de un array. Se inicializa a null
vector = new int[10];           // Vector de 10 enteros, inicializados a 0
double [] v ={1.0, 2.65, 3.1}; // Declaración e inicialización de un vector de 3
                                // elementos con los valores entre llaves
MyClass[] lista=new MyClass[5];// Se crea un vector de 5 referencias a objetos
                                // Las 5 referencias son inicializadas a null
lista[0] = unaRef;              // Se asigna a lista[0] el mismo valor que unaRef
lista[1] = new MyClass();       // Se asigna a lista[1] la referencia al nuevo objeto
                                // El resto (lista[2]…lista[4] siguen con valor null
```

Operadores

- **Operadores aritméticos**
  - \+ op1 + op2 Suma op1 y op2
  - \- op1 - op2 Resta op2 de op1
  - * op1 * op2 Multiplica op1 por op2
  - / op1 / op2 Divide op1 por op2
  - % op1 % op2 Calcula el resto de dividir op1 por op2
- **Operador de concatenación de cadenas de caracteres +**
- **Operadores unarios +, -**
- **Operador de comparación terciario: expression ? op1 : op2**
- **Operadores incrementales ++, --**
- **Operadores relacionales**
  - \> op1 > op2 op1 es mayor que op2
  - >= op1 >= op2 op1 es mayor o igual que op2
  - < op1 < op2 op1 es menor que op2
  - <= op1 <= op2 op1 es menor o igual que op2
  - == op1 == op2 op1 y op2 son iguales
  - != op1 != op2 op1 y op2 son diferentes

Operadores

- **Operadores de asignación**

=	op1 = op2	
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

- **Operadores lógicos**

&&	(and)	op1 && op2	true si op1 y op2 son true
||	(or)	op1 || op2	true si op1 u op2 son true
!	(not)	!op	true si op es false y false si op es true

Estructuras de programación

- Declaraciones
- Sentencias y comentarios // … y /* … */
- Funciones
  - Valor de retorno
- Clases
- Bifurcación
  - If
  - switch
- Bucles
  - for
  - while
  - do while

Estilo de programación (1)

Java Code Conventions

- Declaraciones

```
int level; // indentation level
int size;  // size of table
```

- Funciones

```
void myMethod() {
    int int1 = 0;     // beginning of method block

    if (condition) {
        int int2 = 0; // beginning of "if" block
        ...
    }
}
```

Estilo de programación (2)

- Clases

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}

    ...
}
```

Estilo de programación (3)

- If-else

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

Estilo de programación (4)

- for

```
for (initialization; condition; update) {
    statements;
}
```

- while

```
while (condition) {
    statements;
}
```

- do-while

```
do {
    statements;
} while (condition);
```

Estilo de programación (5)

- Switch

```
switch (value) {
case ABC:
    statements;
    /* falls through */

case DEF:
    statements;
    break;

case XYZ:
    statements;
    break;

default:
    statements;
    break;
}
```

Estilo de programación (6)

- Try-catch

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

Java
Clases, OOP, Package, Interface

Nicolás Serrano

Febrero 2014

Clases

- Clase: estructura de datos y funciones
  - En Java todo se define con clases
  - Clases del sistema y de librerías
  - Clases definidas por el usuario
- Objeto: instancia de una clase
  - permite múltiples instancias de una clase
    - int i, j, k;
    - Circulo a, b, c;

Ejemplo de clases (Circulo.java)

```
public class Circulo {
    static int numCirculos = 0;
    public static final double PI=3.14159;
    public double x;
    public double y;
    public double r;

    public Circulo(double x, double y, double r) {
        this.x=x;
        this.y=y;
        this.r=r;
        numCirculos++;
    }
    public Circulo(double r) {
       this(0.0, 0.0, r);
    }
    public Circulo(Circulo c) {
        this(c.x, c.y, c.r);
    }
    public Circulo() {
        this(0.0, 0.0, 1.0);
    }
    public double perimetro() {
        return 2.0 * PI * r;
    }
    public double area() {
        return PI * r * r;
    }
    // método de objeto para comparar círculos
    public Circulo elMayor(Circulo c) {
        if (this.r>=c.r) {
            return this;
        } else {
            return c;
        }
    }
    // método de clase para comparar círculos
    public static Circulo elMayor(Circulo c, Circulo
   d) {
        if (c.r>=d.r) {
            return c;
        } else {
             return d;
        }
    }
    public String toString() {
        return "x: " + x + " y: " + y + " radio: " +
   r;
    }
} // fin de la clase Circulo
```

Utilización de clases

```
class pruebaCirculos {
    public static void main(String[] args) {
        System.out.println(new Circulo());
        System.out.println(new Circulo(5.0));
        System.out.println(new Circulo(3,5,2));
        Circulo c1 = new Circulo(7);
        Circulo v[] = new Circulo[10];
        System.out.println(c1);
    }
}
```

Componentes de una clase

- Variables
  - De tipos primitivos
  - Otras clases
  - Palabra reservada “this”
- Métodos
  - Son las funciones de la clase que actúan sobre las variables de la clase
  - Constructores
- Ejemplo: Clase Complex.java

OOP

- Conceptos de OOP (Programación Orientada a Objetos)
  - Herencia
    - Una clase puede derivar de otra
    - Hereda métodos y variables
      - Ejemplo: **java.util.Stack**
  - Métodos
    - Permiten sobrecarga y redefinición en métodos heredados
    - Se pueden acceder a los métodos de la super clase con el nombre “super”
  - Variables
    - Pueden ser tipos primitivos u otras clases
    - Diferencia de tipos primitivos (float) y clases (Float)
      java.lang.Float

Variables static

- Las variables son propias de cada objeto o instancia de la clase.
- Las variables **static** pertenecen a la clase
- Se denominan static o de clase
- Se comportan como variables globales (se desaconseja su uso, salvo propósitos específicos: contador, etc)
- Ejemplo:

```
System.in
System.out
```

Métodos static

- Los métodos se aplican a cada objeto o instancia de la clase.
- Las métodos **static** pertenecen a la clase
- Se denominan static o de clase
- Ejemplos:

```
public static void main(String[] args)

Clase Math:
public static double sin(double a)
```

Packages (paquetes)

- Las clases pertenecen habitualmente a un paquete (package)
- Una clase de define de un paquete con la sentencia

```
packake name;
```

- El nombre puede estar compuesto:

```
java.awt.event
```

- El nombre se corresponde con la jerarquía de directorios
- Para utilizar una clase de un paquete es necesario importar la clase
  - Con la excepción de java.lang

Paquetes (2)

- Se utilizan para:
  - Agrupar clases
  - Evitar conflictos de nombres
  - Seguridad de acceso a variables y nombres
- Se importan con:

```
import packageName.*;
import packageName.className;
```

- Documentación del jdk
  - https://docs.oracle.com/javase/8/docs/api/

Acceso a variables y métodos

- Las palabras clave: public, protected y private definen el acceso
  - Public: pueden ser accedidas por cualquier otra clase
  - Protected: pueden ser accedidas por clases del paquete
  - Private: sólo puede acceder la propia clase

Interfaces

- Es un conjunto de declaraciones (en las clases hay definiciones) y constantes
- Define cual es el interface de la clase que lo implementa con el exterior.
- Una clase implementa interfaces, definiendo las declaraciones realizadas en el interface
- Utilidad:
  - Permite que una clase implemente varios interfaces
  - Se puede programar teniendo en cuenta el interface sin necesidad de saber quién lo vaya a implementar
  - Permite ocultar el funcionamiento de la clase que lo implementa

Interfaces - ejemplo

- Enumeration es un interface con los métodos:

```
boolean    hasMoreElements()
nextElement()

for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
     System.out.println(e.nextElement());
}
```

- Las clases implementan el interface mediante la palabra **implements**

Lenguaje Java

Nicolás Serrano

Febrero 2020