

PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

Dra. Maricela Bravo
mari_clau_18@hotmail.com

JUSTIFICACIÓN

- Java es el lenguaje de programación que más impacto ha tenido en los últimos años, especialmente en el mundo de desarrollo para la Web.
- La expansión de Java va en aumento no sólo en el desarrollo de aplicaciones Web, sino en el desarrollo de nuevas tecnologías como son: servicios Web y la programación para dispositivos electrónicos.

1.1 CARACTERÍSTICAS DE JAVA

- **Lenguaje totalmente orientado a objetos**
 - Encapsulación, herencia, polimorfismo, etc.
- **Dispone de un amplio conjunto de librerías**
 - Oracle pone a disposición un amplio conjunto de clases para la creación de interfaces gráficas, gestión de redes, multitarea, acceso a datos, etc.

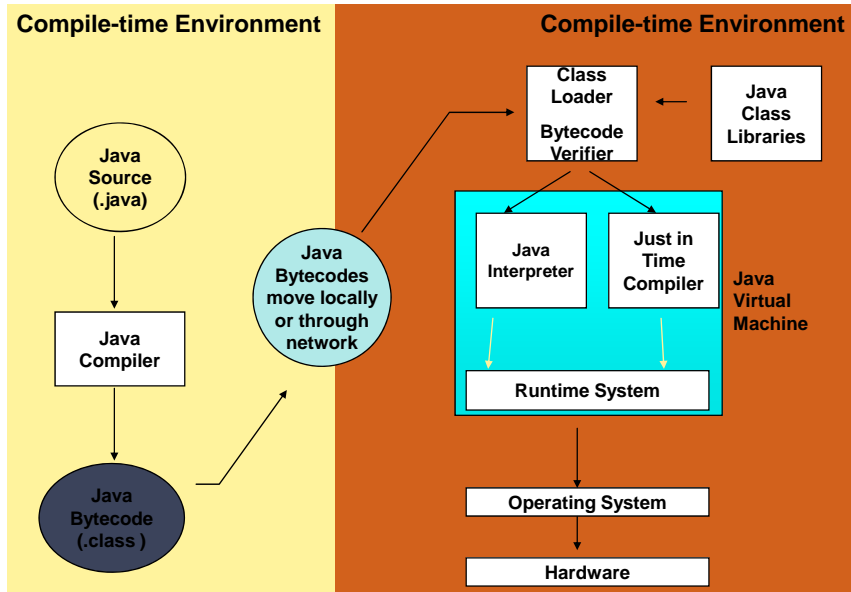
3

1.1 CARACTERÍSTICAS DE JAVA

- **Aplicaciones multiplataforma**
 - Una vez que se ha compilado el programa, éste puede ser ejecutado en diferentes sistemas operativos sin necesidad de volver a compilar el programa.
- **Ejecución segura de aplicaciones**
 - El lenguaje carece de instrucciones que pueden provocar accesos descontrolados a la memoria (apuntadores en C++), la máquina virtual de Java impone ciertas restricciones a las aplicaciones para garantizar una ejecución segura.

4

CÓMO FUNCIONA...!



CÓMO FUNCIONA...!

- Java es independiente de plataforma:
 - Solamente depende de la Máquina Virtual de Java (JVM)
 - El código fuente se compila a *bytecode*, el cual es interpretado por la JVM residente en la máquina,
 - JIT (just in time) los compiladores intentan incrementar la velocidad de ejecución.

VENTAJAS DE JAVA

- Portable – Escribe una vez, ejecuta en todas partes
- Seguridad
- Administración robusta de memoria
- Diseñado para la programación de redes
- Multi-hilos (múltiples tareas simultáneas)
- Dinámico y extensible
 - Clases almacenadas en archivos separados
 - Cargadas solamete cuando es necesario

7

EDICIONES DE JAVA

- Java 2 Estándar Edition (J2SE)
 - Paquetes para el tratamiento de cadenas, colecciones, acceso a datos, creación de entornos gráficos y applets.
- Java 2 Enterprise Edition (J2EE)
 - Paquetes y tecnologías necesarias para la creación de aplicaciones empresariales multicapa, por ejemplo aplicaciones Web.
- Java 2 Micro Edition (J2ME)
 - Paquetes para la creación de aplicaciones para dispositivos electrónicos como son: PDA's, teléfonos móviles, agendas electrónicas, etc.

8

JDK - JAVA DEVELOPMENT KIT

- javac – Compilador de java
- java - Interprete de java
- jdb - Debugger de java
- appletviewer –Herramienta para ejecutar applets

- javap - to print the Java bytecodes
- javaprof - Java profiler
- javadoc - documentation generator

9

ENTORNOS DE DESARROLLO IDE

- NetBeans Sun Microsystem
<http://www.netbeans.org>
- Jbuilder Borland
<http://www.borland.com>
- Jdeveloper Oracle
<http://www.oracle.com>
- Eclipse Eclipse Foundation
<http://www.eclipse.org>

10

ESTRUCTURA DE UN PROGRAMA EN JAVA

```
public class miClase
{
    public static void main(String[] args)
    {
        System.out.println("Primera clase");
    }
}
```

11

CONCEPTOS BÁSICOS DE PROGRAMACIÓN EN JAVA

- Objetos
 - Es una caja negra que expone una serie de operaciones (métodos) que pueden ser utilizados por otros programas.
- Clases
 - Contienen la definición de objetos, dónde se codifican los métodos que van a exponer los objetos de esa clase.
- Métodos
 - Definen el comportamiento de los objetos de una clase, estos campos pueden hacer uso de atributos para almacenar información sobre el objeto.
- Atributos

12

TAREA 2

- Cómo se configuran las variables de entorno
 - CLASSPATH
 - JAVAHOME
 - PATH
- En Windows y Linux, en línea de comandos, archivos por lotes y con herramientas del sistema operativo.

13

ELEMENTOS DEL LENGUAJE DE PROGRAMACIÓN JAVA



2.1 TIPOS DE DATOS PRIMITIVOS

- Como tipos primitivos entendemos aquellos tipos de información más usuales y básicos. Son los habituales de otros lenguajes de programación.
- **Boolean:** No es un valor numérico, solo admite los valores true o false.
- **Char:** Cada caracter ocupa 16 bits.
- **Enteros:** Difieren en las precisiones y pueden ser positivos o negativos.
 - Byte: 1 byte.
 - Short: 2 bytes.
 - Int: 4 bytes.
 - Long: 8 bytes.
- **Reales en punto flotante:** igual que los enteros también difieren en las precisiones y pueden ser positivos o negativos.
 - Float: 4 bytes.
 - Double: 8 bytes.

15

TIPOS DE DATOS PRIMITIVOS Y VARIABLES

- boolean, char, byte, short, int, long, float, double etc.
- Estos tipos de datos básicos o primitivos no son objetos.
- Esto significa que no se puede utilizar el operador new para crear una instancia de objeto para una variable.
- Declaración de variables con tipos primitivos:

```
float initVal;  
int retVal, index = 2;  
double gamma = 1.2;  
boolean valueOk = false;
```

16

2.2 VARIABLES

- Una variable es un espacio físico de memoria donde un programa puede almacenar un datos para su posterior utilización.
- Tipos de datos de una variable
 - Tipos primitivos
 - Tipo objeto

17

INICIALIZACIÓN DE VARIABLES

- Si no se asigna ningún valor previamente al uso de la variable, el compilador causará error.
- Java asigna variables primitivas a cero o falso en el caso de tipos booleanos.
- Todas las referencias a objetos apuntan a null inicialmente.
- Un arreglo es un objeto que se inicia en null al declararse, y sus elementos se inicializan a cero durante la creación.

18

DECLARACIONES

```
int a ; // declaración de una variable 'a'
        //inicializada a 0 (valor por defecto).

int b = 8; // declaración de una variable 'b' inicializada a 8.

NombreClase referencia; // declaración de una variable 'referencia'
                        //preparada //para llevar un objeto de la clase 'NombreClase'.

NombreClase referencia2; // lo mismo que en la variable anterior.

Referencia = new NombreClase; // se crea un nuevo objeto de la
clase
                        //"NombreClase",y es asignado a la variable
                        //"Referencia"

Referencia2 = referencia; // Ahora también 'referencia2' tiene el
mismo
                        // objeto a su cargo que 'referencia'
```

19

ASIGNACIONES

- Todas las asignaciones en Java son de derecha a izquierda

```
int a = 1, b = 2, c = 5
```

```
a = b = c
```

```
System.out.print(
```

```
    "a= " + a + "b= " + b + "c= " + c)
```

- ¿Cuál es el valor de a, b y c?
- De derecha a izquierda: a = (b = c);

20

2.3 OPERADORES ARITMÉTICOS

- * / % + - operadores básicos
- * / % tienen mayor prioridad que + o -

```
double myVal = a + b % d - c * d / b;
```

- Es lo mismo que:

```
double myVal = (a + (b % d)) -  
                ((c * d) / b);
```

21

2.3 OPERADORES DE ASIGNACIÓN

Expresión	Equivalente
A += B;	A = A + B;
A -= B;	A = A - B;
A *= B;	A = A * B;
A /= B;	A = A / B;
A %= B;	A = A % B;

EJEMPLO

```
int a = 2, b = 3;  
a += b;  
b*=5;  
a=++b;  
b*+=--a+5;
```

23

2.3 OPERADORES RELACIONALES

==	Igual
!=	Diferente
>=	Mayor o igual que
<=	Menor o igual que
>	Mayor que
<	Menor que

24

EJERCICIO. DETERMINAR SI LAS SIGUIENTES RELACIONES SON TRUE O FALSE

- `7 <= 5`
- `2 > 6`
- `3 != 5`
- `a == a + 1`
- `b ++ != b++`

25

2.4 EL RECOLECTOR DE BASURA

- Cuando se utilizan los métodos constructores para inicializar variables de instancia de objetos, éstos ocupan recursos del sistema, como por ejemplo memoria.
- Se necesita una forma disciplinada de devolver estos recursos al sistema cuando ya no son necesarios.
- Para evitar la fuga de recursos Java realiza automáticamente la recolección de basura.

26

2.4 EL RECOLECTOR DE BASURA

- El recolector de basura ayuda a regresar al sistema la memoria ocupada por los objetos que ya no se utilizan.
- Cuando ya no existen referencias a un objeto, éste queda marcado para la recolección de basura.
- La memoria de ese objeto podrá reclamarse al momento en que se ejecute el recolector de basura.

27

2.5 ESTRUCTURAS DE CONTROL

- Una instrucción sencilla es un comando terminado por ;
name = "Fred";
- Un bloque de instrucciones es un conjunto de comandos encerrados por llaves

```
{  
    name1 = "Fred";  
    name2 = "Bill";  
}
```
- Los bloques de instrucciones pueden contener otros bloques de instrucciones.

28

FLUJO DE CONTROL

- Java ejecuta una instrucción después de la otra en el orden en que fueron escritos.
- Muchas instrucciones de Java son flujos de control

Alternativas: if, if else, switch

Ciclos: for, while, do while

Escapes: break, continue, return

29

IF – LA INSTRUCCIÓN CONDICIONAL

- La instrucción condicional evalúa una expresión y si el resultado de la evaluación es verdadero, entonces se ejecuta la acción

```
if ( x < 10 ) x = 10;
```

- Si el valor de x es menor que 10, hacer que x sea igual a 10

- También se puede escribir como

```
if ( x < 10 )
```

```
  x = 10;
```

- O alternativamente:

```
if ( x < 10 ) { x = 10; }
```

30

IF... ELSE

- La instrucción condicional if ... else evalúa una expresión y realiza una acción si la evaluación resulta verdadera, o realiza otra acción si la evaluación resulta falsa.

```
if (x != oldx) {  
    System.out.print("x was changed");  
}  
else {  
    System.out.print("x is unchanged");  
}
```

31

IF ... ELSE ANIDADO

```
if ( myVal > 100 ) {  
    if ( remainderOn == true) {  
        myVal = mVal % 100;  
    }  
    else {  
        myVal = myVal / 100.0;  
    }  
}  
else  
{  
    System.out.print("myVal is in range");  
}
```

32

ELSE IF

- Útil para escoger entre alternativas:

```
if ( n == 1 ) {  
    // execute code block #1  
}  
else if ( n == 2 ) {  
    // execute code block #2  
}  
else {  
    // si todas las evaluaciones anteriores han  
    fallado, execute code block #3  
}
```

33

LA SENTENCIA SWITCH

```
switch ( n ) {  
    case 1:  
        // execute code block #1  
        break;  
    case 2:  
        // execute code block #2  
        break;  
    default:  
        // if all previous tests fail then  
        //execute code block #4  
        break;  
}
```

34

EL CICLO FOR

- Repetición n veces

```
for ( i = 0; i < n; n++ ) {  
    // este código se ejecutará n veces  
    // i desde 0 hasta n-1  
}
```

- For anidado:

```
for ( j = 0; j < 10; j++ ) {  
    for ( i = 0; i < 20; i++ ){  
        // este código se ejecutará 200 veces  
    }  
}
```

35

CICLO WHILE

```
while(response == 1) {  
    System.out.print( "ID =" + userID[n]);  
    n++;  
    response = readInt( "Enter ");  
}
```

36

CICLO DO {... } WHILE

```
do {  
    System.out.print( "ID =" + userID[n] );  
    n++;  
    response = readInt( "Enter " );  
}while (response == 1);
```

37

BREAK

- La instrucción break causa un exit desde el ciclo que lo llama.

```
for ( int i = 0; i < maxID, i++ ) {  
    if ( userID[i] == targetID ) {  
        index = i;  
        break;  
    }  
} // la ejecución del programa salta  
aquí después del break
```

38

CONTINUE

- Puede ser utilizado solamente con los ciclos while, do o for.
- La instrucción continue ocasiona que el ciclo continue en la siguiente iteración inmediatamente.

```
for ( int i = 0; i < maxID; i++ )  
{  
    if ( userID[i] != -1 ) continue;  
    System.out.print( "UserID " + i + " :" +  
        userID);  
}
```

39

2.6 ARREGLOS

- Un arreglo es una lista de elementos similares
- Un arreglo tiene:
 - nombre
 - tipo
 - tamaño
- Estos deben ser declarados cuando el arreglo es creado.
- El tamaño del arreglo no puede ser cambiado durante la ejecución del programa.

40

myArray =

3	6	3	1	6	3	4	1
---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

myArray tiene espacio para 8 elementos

- Los elementos son accedidos por su índice
- en Java, los índices de los arreglos comienzan con 0.

41

DECLARACIÓN DE ARREGLOS

int myArray[];

declara *myArray* como un arreglo de enteros

myArray = new int[8];

reserva 8 espacios de enteros en memoria, etiquetados de *myArray[0]* - *myArray[7]*

int myArray[] = new int[8];

combina las dos instrucciones anteriores.

42

ASIGNACIÓN DE VALORES

- Se hace referencia a los elementos del arreglo mediante su índice.

```
myArray[0] = 3;  
myArray[1] = 6;  
myArray[2] = 3; ...
```

- Se puede inicializar en un solo paso.

```
int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};
```

43

ITERACIÓN CON ARREGLOS

- Los ciclos *for* loops son los más usuales para trabajar con arreglos:

```
for (int i = 0; i < myArray.length; i++)  
{  
    myArray[i] = getsomevalue();  
}
```

44

ARREGLOS DE OBJETOS

- Hasta ahora solo hemos visto arreglos de tipos de datos primitivos.
 - integers
 - doubles, floats, characters...
- Frecuentemente necesitamos arreglos de objetos
 - Estudiantes, Libros, Créditos
- Se necesitan seguir tres pasos.

45

DECLARACIÓN DEL ARREGLO

1. Declarar el arreglo

```
private Student studentList[];
```

- Esto declara studentList

2 .Crear el arreglo

```
studentList = new Student[10];
```

- reserva 10 espacios en memoria que pueden almacenar referencias a los objetos de student

3. Crear el objeto de estudiante y añadirlo al arreglo

```
studentList[0] = new Student("Cathy",  
"Computing");
```

46