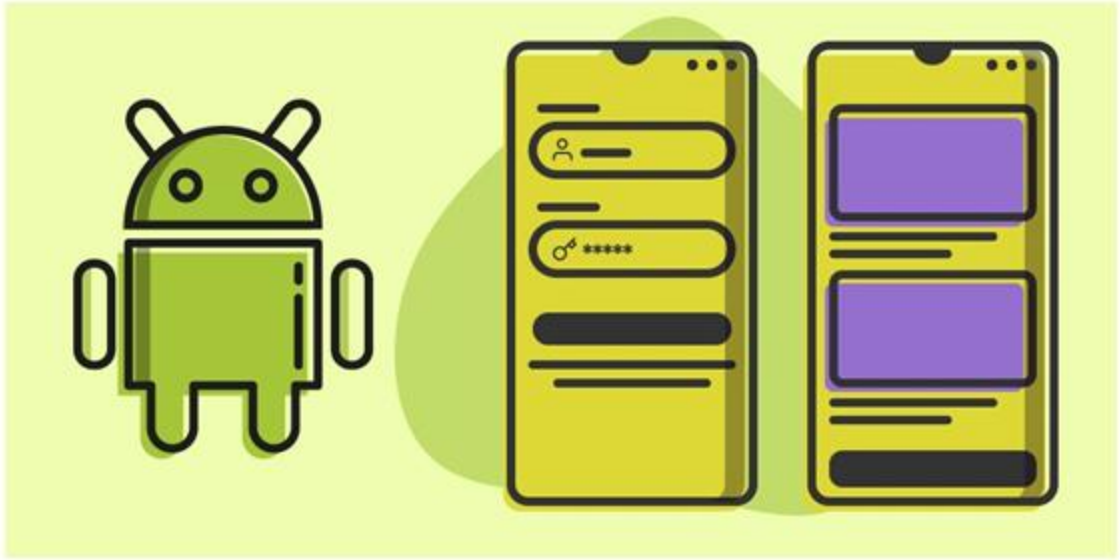
FACULTAD DE INGENIERÍA
ESCUELA DE COMPUTACIÓN



Desarrollo de Software para Móviles

Introducción a Java

Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems y actualmente es propiedad de Oracle.

Su sintaxis es muy parecida a la del lenguaje C o C++, e incorpora como propias algunas características que en otros lenguajes son extensiones: gestión de hilos, ejecución remota, etc.

El código Java, una vez compilado, puede llevarse sin modificación a cualquier máquina, y ejecutarlo. Esto se debe a que el código se ejecuta sobre una máquina virtual (Virtual Machine), dicha maquina virtual se encarga de interpretar el código (ficheros compilados) y convertirlo a código particular de lo pueda entender la CPU.

Características de Java

Podemos mencionar algunas características de Java:

1. Lenguaje Simple: “Se lo conoce como lenguaje simple porque viene de la misma estructura de c y c++; ya que c++ fue un referente para la creación de java por eso utiliza determinadas características de c++ y se han eliminado otras.”
2. Orientado a Objetos.
3. Multihilos: Java tiene una facilidad de cumplir varias funciones al mismo tiempo, gracias a su función de multihilos ya que por cada hilo que el programa tenga se ejecutaran en tiempo real muchas funciones al mismo tiempo.

Conceptos técnicos antes de programar en Java

JRE

El JRE (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma.

JDK

Es el acrónimo de "Java Development Kit", es decir Kit de desarrollo de Java. Se puede definir como un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar aplicaciones Java.

Conceptos técnicos antes de programar en Java

API

Las siglas API tienen su origen en Application Program Interface y consiste en un conjunto de librerías de código java compilas que son ofrecidas a los desarrolladores. La API puede utilizarse según el ámbito de su desarrollo:

1. **Java ME** (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
2. **Java SE** (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
3. **Java EE** (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Tipos de datos numéricos

Tipo	Tamaño en bits	Rango de almacenamiento
byte	8	-128 a 127
short	16	-32,768 a 32,767
int	32	-2,147,483,648 a 2,147,483,647
long	64	-9,223,372,036,854,775,808L a 9,223,372,036,854,775,807L

Tipos de datos numéricos en punto flotante

Tipo	Tamaño en bits	Rango de almacenamiento
float	32	+/- 3.4E+38F (6-7 dígitos importantes)
double	64	+/- 1.8E+308 (15 dígitos importantes)

Tipos de datos boleanos y caracteres

Tipo	Tamaño en bits	Rango de almacenamiento
char	16	Conjunto de caracteres Unicode ISO
boolean	1	verdadero o falso

Tipos de datos estructurados

Cadenas de caracteres

Las cadenas de carácter son una instancia de la clase String y se delimitan entre comillas dobles, en lugar de simple como los caracteres individuales. El siguiente es un ejemplo de la utilización de una cadena de caracteres: String nombreMateria = "Desarrollo de Software para móviles";

Vectores o arrays

Los vectores son colecciones de datos de un mismo tipo, también son conocidos como arrays o arreglos. Un vector es una estructura de datos en la que cada elemento le corresponde una posición identificada por uno o más índices numéricos enteros. Los elementos de un vector se empiezan a numerar en la posición cero y permiten gestionar desde una sola variable múltiples datos del mismo tipo.

Nota: en algunas ocasiones es habitual llamar matrices a los vectores, sin embargo nos referimos a vectores que trabajan con dos dimensiones.

Identificadores

Un identificador es un nombre que identifica a una variable, a un método o función miembro, a una clase. Todos los lenguajes tienen ciertas reglas para componer los identificadores:

Todos los identificadores han de comenzar con una letra, el carácter subrayado (_) o el carácter dollar ($).

- Puede incluir, pero no comenzar por un número
- No puede incluir el carácter espacio en blanco
- Distingue entre letras mayúsculas y minúsculas
- No se pueden utilizar las palabras reservadas como identificadores

Operadores relacionales

Operador	Uso	Devuelve verdadero si
>	ope1 > ope2	ope1 es mayor que ope2
>=	ope1 >= ope2	ope1 es mayor o igual que ope2
<	ope1 < ope2	ope1 es menor que ope2
<=	ope1 <= ope2	ope1 es menor o igual que ope2
= =	ope1 = = ope2	ope1 y ope2 son iguales
! =	ope1 != ope2	ope1 y ope2 son distintos

Operadores matemáticos

Operador	Uso	Descripción
+	ope1 + ope2	Suma ope1 y ope2 (*)
-	ope1 - ope2	Resta ope1 de ope2
*	ope1 * ope2	Multiplica ope1 y ope2
/	ope1 / ope2	Divide ope1 por ope2
%	ope1 % ope2	Obtiene el módulo de dividir ope1 por ope2
++	ope++	Incrementa ope en 1; evalúa el valor antes de incrementar
++	++ope	Incrementa ope en 1; evalúa el valor después de incrementar
--	ope--	Decrementa ope en 1; evalúa el valor antes de incrementar
--	--ope	Decrementa ope en 1; evalúa el valor después de incrementar

Operadores lógicos

Operador	Uso	Devuelve verdadero si
&&	ope1 && ope2	ope1 y ope2 son verdaderos
||	ope1 || ope2	uno de los dos es verdadero
!	! ope	ope es falso (niega ope)

¿Qué es una clase en Java?

Una clase es una plantilla (características y comportamiento) que define la forma de un objeto. Especifica los datos y el código que operará en esos datos. Java usa una especificación de clase para construir objetos. Los objetos son instancias de una clase.

Por lo tanto, **una clase es esencialmente un conjunto de planes que especifican cómo construir un objeto.**

```
class NombreClase{
    //Declarar variables de instancia
    tipo var1;
    tipo var2;
    //..

    //Declarar métodos

    tipo metodo1(parámetros){
        //Cuerpo del método
    }

    tipo metodo2(parámetros){
        //Cuerpo del método
    }

}
```

Campo, método y constructor

Toda clase u objeto se compone internamente de constructores, campos y/o métodos.

- ***Un campo*** es un elemento que contiene información relativa a la clase,
- ***un método*** es un elemento que permite manipular la información de los campos.
- ***Un constructor*** es un elemento que permite reservar memoria para almacenar los campos y métodos de la clase, a la hora de crear un objeto de la misma.

Herencia

Con la herencia podemos definir una clase a partir de otra que ya exista, de forma que la nueva clase tendrá todas las variables y métodos de la clase a partir de la que se crea, más las variables y métodos nuevos que necesite.

Nota: A la clase base a partir de la cual se crea la nueva clase se le llama superclase.

Por ejemplo, podríamos tener una clase genérica Animal, y heredamos de ella para formar clases más específicas, como Pato, Elefante, o León.

Estas clases tendrían todo lo de la clase padre Animal, y además cada una podría tener sus propios elementos adicionales.

Polimorfismo

Polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Dicho de otro modo el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa.

En JAVA el término polimorfismo también suele definirse como “Sobrecarga de parámetros”

```
class Animal {
  public void makeSound() {
    System.out.println("Grr...");
  }
}
class Cat extends Animal {
  public void makeSound() {
    System.out.println("Meow");
  }
}
class Dog extends Animal {
  public void makeSound() {
    System.out.println("Woof");
  }
}
```

Modificadores de acceso de ámbito

Tanto las clases como sus elementos (constructores, campos y métodos) pueden verse modificados por lo que se suelen llamar modificadores de acceso, que indican hasta dónde es accesible el elemento que modifican. Tenemos tres tipos de modificadores:

- **Privado:** el elemento es accesible únicamente dentro de la clase en la que se encuentra.
- **Protegido:** el elemento es accesible desde la clase en la que se encuentra, y además desde las subclases que hereden de dicha clase.
- **Público:** el elemento es accesible desde cualquier clase.

Modificadores de acceso especiales

- **abstract :** elemento base para la herencia (los objetos subtipo deberán definir este elemento). Se utiliza para definir clases abstractas, y métodos abstractos dentro de dichas clases, para que los implementen las subclases que hereden de ella.

- **static :** elemento compartido por todos los objetos de la misma clase. Con este modificador, no se crea una copia del elemento en cada objeto que se cree de la clase, sino que todos comparten una sola copia en memoria del elemento, que se crea sin necesidad de crear un objeto de la clase que lo contiene.

- **final** : objeto final, no modificable (se utiliza para definir constantes) ni heredable (en caso de aplicarlo a clases).

- **synchronized** : para elementos a los que no se puede acceder al mismo tiempo desde distintos hilos de ejecución.

Modificadores de acceso especiales

```
// Clase abstracta para heredar de ella
public abstract class Ejemplo
{
  // Constante estática de valor 10
  public static final TAM = 10;

  // Método abstracto a implementar
  public abstract void metodo();

  public synchronized void otroMetodo()
  {
    ... // Aquí dentro sólo puede haber un hilo a la vez
  }
}
```

Paquetes

Las clases en Java se organizan (o pueden organizarse) en paquetes, de forma que cada paquete contenga un conjunto de clases.

También puede haber subpaquetes especializados dentro de un paquete o subpaquete, formando así una jerarquía de paquetes, que después se plasma en el disco duro en una estructura de directorios y subdirectorios igual a la de paquetes y subpaquetes (cada clase irá en el directorio/subdirectorio correspondiente a su paquete/subpaquete).

Cuando queremos indicar que una clase pertenece a un determinado paquete o subpaquete, se coloca al principio del fichero la palabra reservada package seguida por los paquetes/subpaquetes, separados por . :

```
package paq1.subpaq1;
...
class MiClase {
...
```

Paquetes

Si queremos desde otra clase utilizar una clase de un paquete o subpaquete determinado (diferente al de la clase en la que estamos), incluimos una sentencia import antes de la clase (y después de la línea package que pueda tener la clase, si la tiene), indicando qué paquete o subpaquete queremos importar:

```
import paq1.subpaq1.*;
```

```
import paq1.subpaq1.MiClase;
```

La primera opción (*) se utiliza para importar todas las clases del paquete (se utiliza cuando queremos utilizar muchas clases del paquete, para no ir importando una a una). La segunda opción se utiliza para importar una clase en concreto.

Puntero this

El puntero this apunta al objeto en el que nos encontramos. Se utiliza normalmente cuando hay variables locales con el mismo nombre que variables de instancia de nuestro objeto:

```
public class MiClase
{
  int i;
  public MiClase(int i)
  {
    this.i = i;     // i de la clase = parametro i
  }
}
```

También se suele utilizar para remarcar que se está accediendo a variables de instancia.

Puntero super

El puntero super se usa para acceder a un elemento en la clase padre. Si la clase Usuario tiene un método getPermisos , y una subclase UsuarioAdministrador sobrescribe dicho método, podríamos llamar al método de la super-clase con:

```
public class UsuarioAdministrador extends Usuario {
  public List<String> getPermisos() {
    List<String> permisos = super.getPermisos();
    permisos.add(PERMISO_ADMINISTRADOR);
    return permisos;
  }
}
```

También podemos utilizar this y super como primera instrucción dentro de un constructor para invocar a otros constructores.

¿Preguntas?