Introducción POO y Java

Objetivos

- Conocer y comprender las principales características y diferencias de la programación orientada a objetos frente a otros paradigmas
- Conocer y comprender los principales conceptos de la POO
- Conocer y comprender las principales características del lenguaje Java
- Conocer y comprender las características generales de los entornos de desarrollo de aplicaciones en lenguaje Java

Resumen

- POO
  - Motivación de la POO
  - Conceptos básicos de la POO
  - Mecanismos básicos de la POO
- Java
  - Historia
  - Características generales
  - Desarrollo y ejecución de aplicaciones
  - Ediciones de Java

Programación Orientada a Objetos

Resumen

- Motivación de la POO
- Conceptos básicos de la POO
  - Clases y objetos
  - Atributos y métodos
  - Creación y destrucción de objetos
  - Composición
- Mecanismos básicos de la POO
  - Encapsulación
  - Herencia
  - Polimorfismo

Motivación POO - Diseño OO

- En general, toda construcción de "software" exige un proceso metodológico → Planificación e Ingeniería del Software
- En los programas orientados a objetos, el diseño es fundamental (discusión histórica)
- Gran parte de las metodologías actuales (OMT, Booch, UML, Proceso Unificado de Desarrollo) "sugieren" la implementación con objetos

Motivación POO - Paradigma de la OO

- La orientación a objetos se puede considerar como una evolución (no revolución) de la programación estructurada
- Equilibrio entre los datos y los procesos, frente al predominio de los procesos en la programación estructurada
- Resuelve problemas complicados, no está pensada para tareas sencillas

Conceptos de la POO - Clases

- Abstracción de alto nivel, modela el mundo real (y el mundo informático)
- Se puede ver como un molde, frente al objeto, que es la instanciación concreta de la clase (también como un tipo, frente al objeto, que es la variable).
- Conjunto de:
  - **Atributos**: Variables
  - **Métodos**: Funciones para manipular esas variables
- Concepto fundamental en OO (y aún más fundamental en JAVA)

Conceptos de la POO - Atributos

- Dentro de una clase, variables que la definen
- Representa la importancia de los datos en la OO
- Concepto importante (también para métodos): Con el modificador static es propio de la clase, no de cada objeto.

Conceptos de la POO – Creación y destrucción de objetos

- Creación:
  - Uso del método del constructor. Se ocupa de reservar la memoria suficiente para el objeto en cuestión, y de inicializar los atributos
  - Ejemplo:
    - robot1 = Robot(1,2)
    - robot2 = Robot(3)
- Destrucción: "garbage collection"
  - Explícita (C++)
  - Automática (Java)

Conceptos de la POO – Métodos

- Dentro de una clase, definición de una función que manipula sus atributos
- Un método se ejecuta cuando el objeto recibe un mensaje de ejecución del método
- Concepto importante → Sobrecarga
  - Un método puede tener distintas implementaciones con distinto número de argumentos
  - Ejemplo:
    - void avanzar (int, int)
    - void avanzar (int)

Conceptos de la POO – Composición

- Se produce cuando un objeto es atributo de una clase distinta
- Es una forma de reutilizar código
- Debe ser fruto de un buen diseño (Coche, Motor, Rueda)
- No hay que olvidar los objetos puramente informáticos

Conceptos de la POO – Ejemplo – Clase Robot

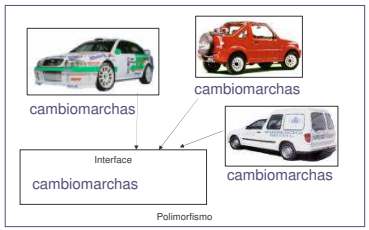
- Atributos:
  - x (int), y (int)
- Métodos:
  - void avanzar (int, int)
  - int posicionX ()
  - int posicionY ()
  - void avanzar (int)
- Constructores:
  - Robot (int, int)
  - Robot (int)
- Instancias:
  - robot1 = Robot(1,2)
  - robot1.avanzar(3,5)

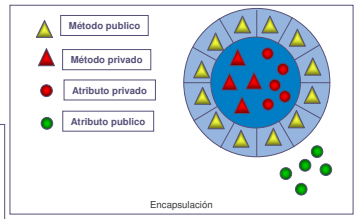
Mecanismos básicos de la POO - Encapsulación

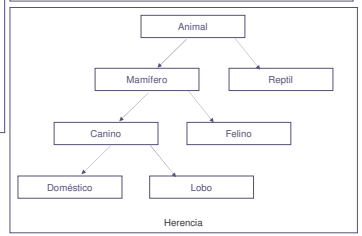
- El contenido de alguna información está oculto
  - Interfaz pública: se puede invocar desde fuera de la clase
  - Implementación privada
- Control de acceso para evitar un uso inadecuado
- Clases:
  - *public*
  - *package*
- Variables y métodos:
  - *public* (cualquier clase)
  - *private* (clase)
  - *protected* (clase, derivadas y paquete)
  - *package* (paquete)

Mecanismos básicos de la POO

- Encapsulación
- Herencia
- Polimorfismo

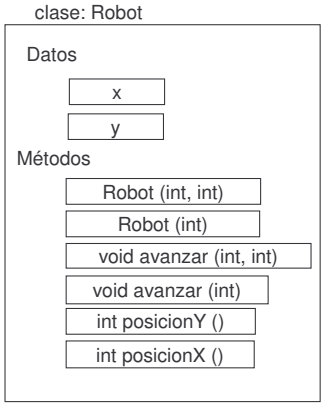






Mecanismos básicos de la POO – Encapsulación - Ejemplo

- Datos:
  - private int x
  - private int y
- Constructores:
  - public Robot (int, int)
  - public Robot (int)
- Métodos:
  - public void avanzar (int, int)
  - public int posicionX ()
  - public int posicionY ()
  - public void avanzar (int)



Mecanismos básicos de la POO – Herencia

- Unas clases (hijas o subclases) se dice que heredan de otras (padres) cuando son particularizaciones y extensiones de ellas
- Ejemplo:
  - Empleado: Fecha de inicio contrato, sueldo
  - Jefe: Empleado, con X empleados a su servicio
  - Becario: Empleado, al servicio de un jefe
- Se puede entender como las características genéticas que se reciben de un padre
  - Algunas características son las mismas
  - Otras son parecidas pero no iguales
  - Otras son totalmente diferentes


Mecanismos básicos de la POO – Herencia - Ejemplo

- Clase: RobotConFrontera
- Clase padre: Robot
- Datos:
  - private limX (int)
  - private limY (int)
- Métodos:
  - void avanzar (int, int)
  - void avanzar (int)
- Constructores:
  - RobotConFrontera (int, int, int, int)
  - RobotConFrontera (int, int, int)
- Instanciación:
  - objeto "robot3" (RobotConFrontera)
  - robot3 = RobotConFrontera(1,2,0,0)


Mecanismos básicos de la POO – Herencia

- Si una clase deriva de otra:
  - Hereda sus variables y métodos
  - Puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados
  - Si no se especifica de qué clase deriva → Object (base)
- Tipos:
  - Herencia simple (Java)
    - Figura → Círculo, Rectángulo, Cuadrado, Triángulo
  - Herencia múltiple (otros lenguajes, p.e. C++)
    - Problemas de ambigüedad


Mecanismos básicos de la POO – Polimorfismo

- Capacidad de permitir referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas según sea el objeto que se referencia en ese momento
- Por ejemplo para una clase Mamifero se puede tener un método comer()
- Creamos otras subclases Rumiante, Hombre y Felino que heredan de Mamifero
  - Por ejemplo, todas ellas heredan el método comer(), sin embargo para cada clase el método funciona de diferente manera (el método comer adopta varias formas)
  - Por ejemplo, puedo declarar una variable Mamifero y construir sobre ella un objeto Hombre (el objeto Mamifero adopta varias formas)


Introducción a Java

Introducción

- Históricamente, la creación de un programa:
  - Un programa escrito en lenguaje máquina es propenso a errores (0,1)
  - Lenguajes ensamblador con instrucciones mnemotécnicas (errores frecuentes, tedioso): ensamblador
  - Lenguajes de alto nivel (C, Pascal, Ada, C++,...): El compilador genera código máquina a partir de un fuente
  - Algunos lenguajes de alto nivel se interpretan (Java o Basic): El intérprete no genera código máquina a partir del fuente, efectúa la traducción y ejecución simultáneamente (+ lentos en general)

Resumen

- Introducción a Java
  - Historia
  - Características generales
  - Bytecode y máquina virtual
  - Entornos de desarrollo: JDK, JRE, IDEs
  - Desarrollo y ejecución de aplicaciones
    - PATH y CLASSPATH
  - Ediciones de Java

Origen de Java

- Java surgió en 1991→ Sun MicroSystems
- Lenguaje de POO
- Distintos tipos de CPUs y continuos cambios → herramienta independiente del tipo de CPU utilizada (multiplataforma)
- JVM (*Java Virtual Machine*) interpreta código neutro y lo convierte en código particular de la CPU utilizada
- Inclusión de un intérprete Java en *Netscape 2.0* → lenguaje de propósito general en 1995

Características de Java

- Orientado a Objetos
- Fácil de usar:
  - No utiliza punteros
  - No hay herencia múltiple
- Robusto:
  - Lenguaje estrictamente tipado, comprueba el código en tiempo de compilación y de ejecución
  - "*Garbage collection*": Reserva de memoria y liberación automática
  - Control de excepciones (p.e. división por cero o fichero no encontrado)

Bytecode y máquina virtual de Java (JVM)

- **Objetivo** → Código compilado para ejecutar en cualquier máquina independientemente del procesador utilizado
- La salida del compilador de Java NO es un ejecutable, es '**bytecode**': conjunto de instrucciones de nivel alto que se ejecutarán en una máquina virtual de Java (JVM)
- La **JVM** interpreta los 'bytecodes' (*.class) creados por el compilador (javac.exe) y los convierte a código particular de la CPU utilizada → Independencia de la plataforma
- Las JVMs están implementadas para cada plataforma pero interpretan el mismo bytecode → programas portables

Características de Java

- Multiproceso: Permite programas que hacen tareas a la vez usando hilos o *threads*
- Portabilidad: Un programa se puede ejecutar en cualquier lugar (*bytecodes*)
- Distribuido: Objetos situados en ordenadores diferentes pueden ejecutar procedimientos remotos (paquete RMI - *Remote Method Invocation*)
- Inconvenientes: En algunos casos, velocidad frente a programas compilados

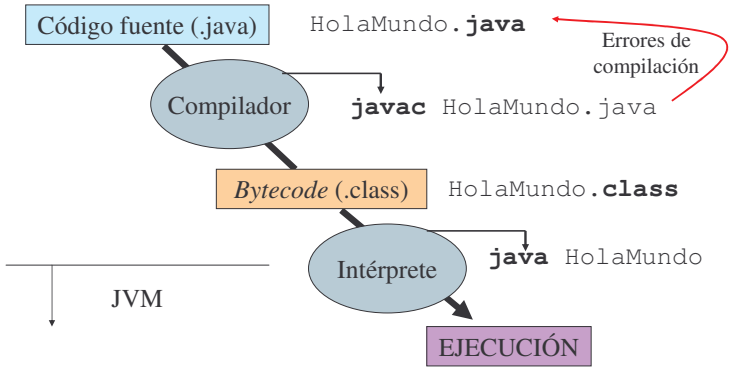
Entornos de desarrollo - JDK

- *Java Development Kit* (**JDK**): Conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java desde consola
  - **javac.exe**: Compilador de Java. Transforma el código fuente (.java) en bytecodes (.class) para la JVM.
  - **java.exe**: Intérprete de Java. Ejecuta la aplicación de Java compilada (.class)
  - **jdb.exe**: Depurador de Java
  - **javadoc.exe**: Generador de documentación
  - **AppletViewer**: Visor de Applets Java
  - **jar.exe**: Compresor de ficheros

Entornos de desarrollo – JREs e IDEs

- **JRE** (*Java Runtime Environment*): Versión reducida de la JDK destinada únicamente a ejecutar código Java
- **IDE**s (*Integrated Development Environment*): Entornos de desarrollo integrados que permiten escribir código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación (debug gráfico). Ejemplos:
  - Netbeans
  - Eclipse
  - JBlue


Desarrollo y ejecución de aplicaciones



Desarrollo y ejecución de aplicaciones

- Para desarrollar y ejecutar aplicaciones en Java:
  - Compilador: Analiza el código fuente (*.java) y genera los ficheros con bytecode (*.class)
  - Intérprete: Interpreta y ejecuta los programas previamente compilados (*.class)
- Herramientas de desarrollo incluidas en la JDK


Desarrollo y ejecución de aplicaciones

Exactamente el mismo
Nombre de archivo que de clase

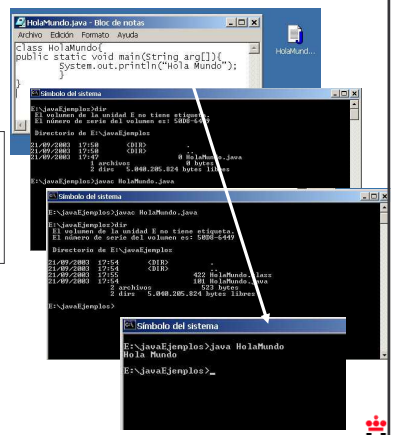
HolaMundo.java

```
class HolaMundo{
public static void main(String arg[]){
   /*Escribimos en pantalla*/
   System.out.println("Hola Mundo");
   }
}
```

```
javac HolaMundo.java
```

HolaMundo.class

```
java HolaMundo.class
```




PATH y CLASSPATH

- En una consola MS-DOS, sólo se pueden ejecutar los programas que se encuentran en los directorios indicados en la variable PATH del ordenador (o en el directorio activo)
- CLASSPATH indica el lugar donde se encuentran tanto las clases o librerías de Java (API) como otras clases de usuario:
  - A partir de la JDK 1.1.4 sólo es necesario especificar las clases del usuario que no vengan en esa JDK

Ediciones de Java

- Java2 a partir de la versión 1.2 de Java
- Plataforma de desarrollo de Java:
  - http://java.sun.com/
  - http://java.sun.com/j2se/1.4.2/download.html
- J2SE: *Java 2 Standard Edition*
  - *Java 2 SDK*: Herramientas de desarrollo, colección de APIs (*Application Programming Interface)* útiles
  - *Java 2 Runtime Environment*: Máquina Virtual
  - Documentación

Configuración PATH y CLASSPATH

- 1ª Solución: Fichero .bat con los valores de estas dos variables. Cada vez que se abre una consola se debe ejecutar el fichero:
  - set JAVAPATH=C:\jdk1.1.7
  - set PATH=.;%JAVAPATH%\bin;%PATH%
  - set CLASSPATH=.\;%JAVAPATH%\lib\libr.zip;%CLASSPATH%
- 2ª Solución: Añadir estos valores de forma permanente.
  - Windows XP: Mi PC → Propiedades (botón derecho) → Opciones Avanzadas → Variables de entorno
- 3ª Solución: Opción –classpath al compilar o ejecutar. P.e., MiPrograma.java necesita la librería OtrasClases.jar:
  - javac -classpath .\;G:\d1\OtrasClases.jar MiPrograma.java
  - java -classpath .\;G:\d1\OtrasClases.jar MiPrograma

Ediciones de Java

- J2EE (*Java 2 Enterprise Edition*): Incluye características adicionales como servlets, JSP, JDBC, JavaBeans, etc.
- J2ME (*Java Micro Edition*): Colección de *APIs* para dispositivos con recursos limitados (*PDAs*, teléfonos móviles, electrodomésticos…)
- Java Card: Permite ejecutar de forma segura pequeñas aplicaciones (*applets*) para *SmartCards* (*SIMs* de teléfonos móviles, tarjetas de monedero electrónico bancarias, …)