ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Universidade Vigo

TEMA 1: Clases y Objetos

Programación II - 2017/2018

Pedro Cuesta Morales, Baltasar García Pérez-Schofield,
Encarnación González Rufino (Dpto. de Informática)

Índice

1. Introducción

Prog. Imperativa

Programación Estructurada

- Diseño descendente o "Top-down"
- Estructuras de control: secuencial, condicional e iterativas
- ...

Programación Modular

- Descomposición de un programa en módulos: procedimientos y funciones
- Programas: algoritmos + estructuras de datos
- ...

C → JAVA

P. Orientada a Objetos

Identificar entidades en el problema a resolver y simular la forma de interaccionar entre ellas

Basada en:

- Clases y objetos
- Encapsulación
- Herencia
- Polimorfismo

Java

Orientado a Objetos → todos los conceptos de la POO (clases y objetos, encapsulación, herencia, polimorfismo, ...)

Portable (multiplataforma) → Bycodes + Maquina Virtual

Simple → basado en C++ se eliminan características fuente de errores (punteros, ...)

Potente → amplio conjunto de bibliotecas reutilizables

Distribuido → biblioteca de clases (tcp/ip, http, ...) facilitan desarrollo de aplicaciones distribuidas (Cliente/Servidor, Web, ...)

Robusto → programas menos fallos o errores: comprobaciones en compilación y tiempo de ejecución, recolección automática de basura, gestión de excepciones, ...

Multihilo → sincronización de múltiples hilos de ejecución (multithreading)

Seguro → establecimiento de políticas/niveles de seguridad ejecución código

Ediciones de JAVA

http://www.oracle.com/technetwork/es/java/index.html

- Java Standar Edition (Java SE): para desarrollo de aplicaciones de escritorio y servidores (proporciona paquetes de clases de uso general, para la creación de aplicaciones gráficas, ...)
  - Entorno de ejecución: Java SE Runtime Environment (JRE)
  - Entorno de desarrollo: Java SE Development Kit (JDK)
- Java Enterprise Edition (Java EE): estándar en el desarrollo de software empresarial (utiliza Java Community Process)
- Java Micro Edition (Java ME): entorno para aplicaciones que se ejecutan en dispositivos móviles y sistemas embebidos en Internet de las cosas (microcontroladores, sensores, ...)

Ejecución de un programa en Java

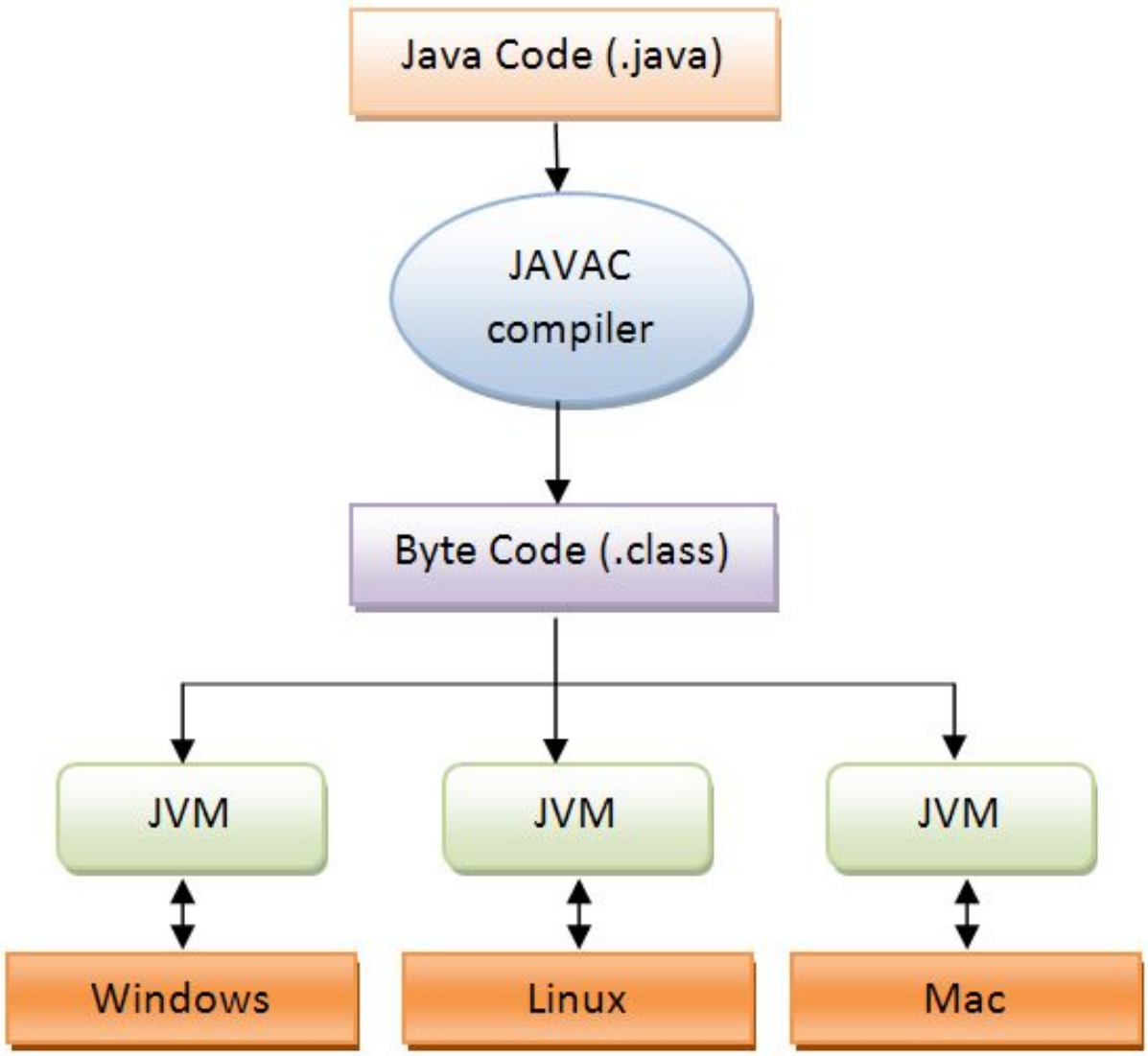


Imagen: https://commons.wikimedia.org/wiki/File:Java-program-execution.png

Integrated Development Environmet - IDE

The Top 7 Free IDEs for Java Development & Programming(Poll)

Berk SOYSAL on Thursday, June 16, 2016

ANDROID | ANDROID STUDIO | BLUEJ | DRJAVA

http://www.codemio.com/2016/06/the-top-7-free-ides-for-java.html

2. Clases y objetos

CLASE (TIPO)

- Describir objetos similares mediante la definición de sus estructuras de datos (atributos) y métodos comunes (operaciones)
- Plantillas para crear objetos, permitiendo la agrupación de objetos que comparten las mismas propiedades y comportamiento

OBJETO (VARIABLE)

- Objeto: instancia de una clase. Un elemento concreto de la clase con un valor específico en cada uno de los atributos de la clase (estado), sobre el que podemos realizar una serie de operaciones (comportamiento)

2. Clases y objetos

Clase Punto:

atributos: x, y

métodos: calculaDistanciaOrigen()

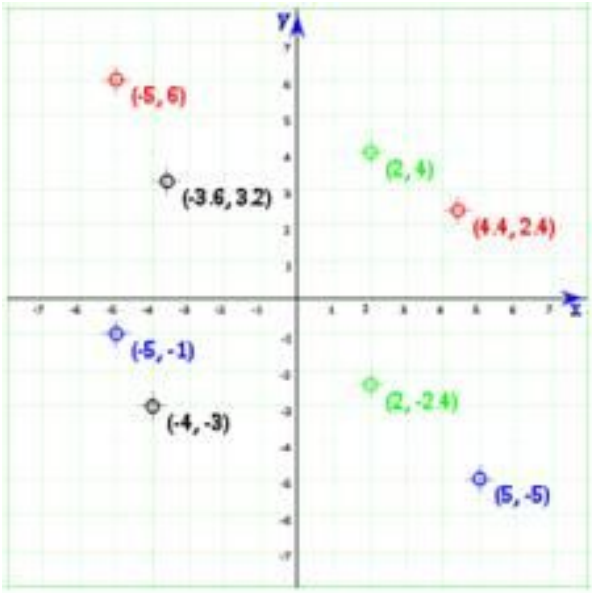
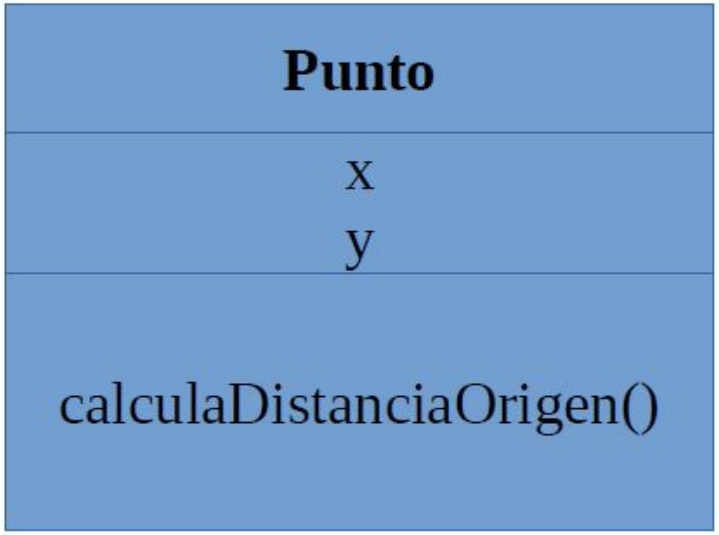


Diagrama representativo de una clase → UML



2.2. Implementación en Java de clases

Tomando como referencia lenguaje de programación C

CLASE = estructura a la que se añaden funciones que actúan directamente sobre los atributos

```
class Punto {
	int x;
	int y;

	double calculaDistanciaOrigen() {
		return Math.sqrt( ( x * x ) + ( y * y ) );
	}
}
```

sqrt( ) método estático de la clase *Math*

Método estático: se invoca sobre la clase, no sobre un objeto (instancia concreta de la clase)

2.3. Los distintos tipos de datos

Declaración de atributos en la clase: *[acceso] Tipo atributo;*

Tipo	Explicación	Tamaño
int	Números enteros.	32bits -2147483648 a 2147483647
long	Números enteros de precisión doble.	64bits -9223372036854775808 a 9223372036854775807
double	Números reales de precisión doble. Norma IEEE 754.	64bits $\pm 1.8x10^{-308}$ a $\pm 1.8x10^{308}$
char	Carácter	16bits. Unicode, UTF-16.
String	Texto o cadena de caracteres.	N/A
boolean	Booleano: acepta valores **true**, **false**.	N/A

2.4. Objetos

Operador creación de objetos: new

Clase referenciaObjeto = new Clase ([argumentos]);

Se crea una referencia y se hace apuntar a un nuevo objeto de la clase

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto();
    p.x = 5;
    p.y = 5;

    System.out.println( p.calculaDistanciaOrigen() );
}
```

stdout

7.071067811865475

Ejemplo1: http://ideone.com/NjjMnQ

3. Visibilidad básica

Principio de Ocultamiento de Información - Encapsulación:

- Atributos de una clase privados (no visibles desde el exterior)
- Acceso para consulta o modificación de los atributos se realiza a través de métodos públicos

Interfaz Clase = conjunto de sus métodos públicos

```
Class IdentificadorClase {
    [private] Tipo atributo;
    [public] Tipo metodo ( [parametros] ) {...}
}
```

private | *public* : modificadores de acceso (lenguajes OO)

Pueden utilizarse métodos privados (no visibles desde el exterior de la clase) como ayuda para el resto de métodos

3. Visibilidad básica

Atributos no son accesibles desde el exterior

```
class Punto {
        private int x;
        private int y;

        public double calculaDistanciaOrigen()
        {
                return Math.sqrt( ( x * x ) + ( y * y ) );
        }
}
```

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto();
    p.x = 5;
    p.y = 5;
```

```
rigen() );
```

información de compilación

```
Main.java:23: error: x has private access in Punto
        p.x = 5;
         ^
Main.java:24: error: y has private access in Punto
        p.y = 5;
         ^
2 errors
```

3. Constructores

- Métodos especiales que se ejecutan justo después de crear el objeto.
- Típicamente se utilizan para inicializar los atributos con valores que se pasan en el momento de crear el objeto con new.
- Los constructores tienen el mismo nombre que la clase, y no devuelven nada

```
Class IdentificadorClase {
    [private] Tipo atributo;
    [public] Tipo metodo ( [parametros] ) {...}
    [public] IdentificadorClase( [parametros] ) {...}
}
```

3. Constructores

```
class Punto {
        private int x;
        private int y;

        public Punto(int a, int b)
        {
                x = a;
                y = b;
        }

        public double calculaDistanciaOrigen()
        {
                return Math.sqrt( ( x * x ) + ( y * y ) );
        }
}
```

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto ( 5, 5 );
    System.out.println( p.calculaDistanciaOrigen() );
}
```

5. Getters

```
class Punto {
	private int x;
	private int y;

	public Punto(int a, int b)|
	{
		x = a;
		y = b;
	}

	public int getX()
	{
		return x;
	}

	public int getY()
	{
		return y;
	}

	public double calculaDistanciaOrigen()
	{
		return Math.sqrt( ( x * x ) + ( y * y ) );
	}
}
```

Métodos que se crean para devolver la información guardada por los atributos.

Su nombre siempre empieza por get, continúan con el nombre del atributo, y se caracterizan por devolver directamente el valor de ese atributo.

6. El método toString()

- Método que devuelve como texto la información que aloja el objeto (el valor de su atributos → estado)
- Se utiliza siempre que se desea mostrar la información completa del objeto por pantalla
- Es altamente recomendable aportar siempre un método toString() en las clases que creemos
- Clase Punto:

```
public String toString()
{
    return "(" + x + ", " + y + ")";
}
```

```
public String toString()
{
    return "(" + getX() + ", " + getY() + ")";
}
```

Ejemplo2: http://ideone.com/18l68x

7. Constantes

- Se indican como cualquier otro atributo anteponiendo el modificador *final*
- No pueden cambiar durante la vida del objeto

```
class Punto {
        public final int ORIGEN_X = 0;
        public final int ORIGEN_Y = 0;

        private int x;
        private int y;
```

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto( 5, 5 );
    Punto origen = new Punto (p.ORIGEN_X, p.ORIGEN_Y);

    System.out.println( "La distancia de "
                    + p.toString() + " a " + origen.toString() + " es "
                    + p.calculaDistanciaOrigen() );
}
```

7. Constantes

Inconvenientes:

- ORIGEN_X y ORIGEN_Y dupicadas en todos los objetos que se creen de la clase Punto
- Es necesario crear previamente al menos un objeto de la clase Punto

SOLUCIÓN: modificador **static**

- Ese miembro no pertenece a los objetos que se creen de esa clase, sino a la clase propiamente dicha
- Acceso: NombreClase.miembroStatic
- Sólo existirá una copia de ese miembro

```
class Punto {
        public static final int ORIGEN_X = 0;
        public static final int ORIGEN_Y = 0;
```

7. Constantes

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto( 5, 5 );
    Punto origen = new Punto (Punto.ORIGEN_X, Punto.ORIGEN_Y);

    System.out.println( "La distancia de "
                    + p.toString() + " a " + origen.toString() + " es "
                    + p.calculaDistanciaOrigen() );
}
```

Ejemplo3: http://ideone.com/2Mve14

7.1. Constantes complejas

Ejemplo4:http://ideone.com/oo1c9u

Los atributos se crean cuando se crea el objeto

Los atributos estáticos ya existen cuando el primer objeto de la clase puede ser creado

Objetos también pueden ser atributos estáticos de una clase

```
class Punto {
    public static final Punto ORIGEN = new Punto( 0, 0 );

    private int x;
    private int y;
```

```
public static void main (String[] args) throws java.lang.Exception
{
    Punto p = new Punto( 5, 5 );

    System.out.println( "La distancia de "
                + p.toString() + " a " + Punto.ORIGEN.toString() + " es "
                + p.calculaDistanciaOrigen() );
}
```

8. La referencia this

En cualquier método de una clase **this** es una referencia que apunta al objeto actual (está ejecutando el método)

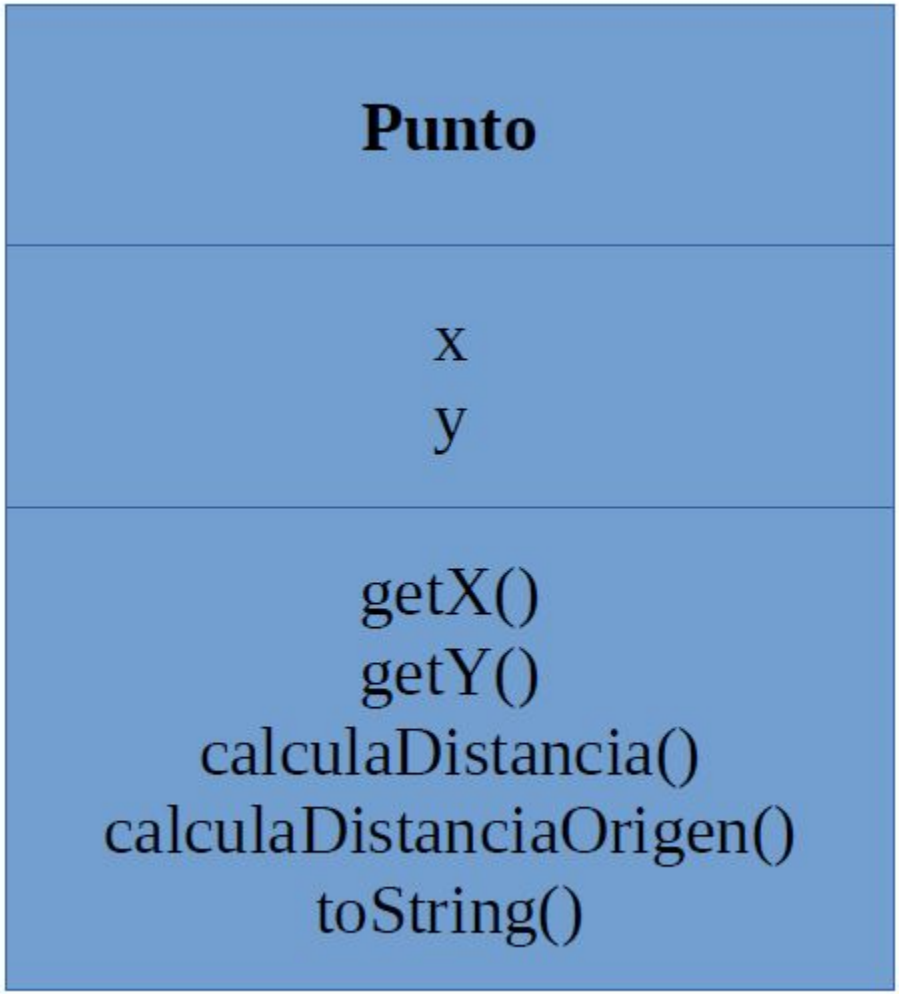
No es obligatorio usar this, salvo cuando coinciden nombre de parámetros y atributos

```
class Punto {
      //...
      public Punto(int x, int y)
      {
            this.x = x;
            this.y = y;
      }
```

```
public double calculaDistancia(Punto p2)
{
      int difX = p2.getX() - this.getX();
      int difY = p2.getY() - this.getY();

      return Math.sqrt( ( difX * difX ) + ( difY * difY ) );
}
```

9. La clase Punto



La clase Punto:
http://ideone.com/EcbMgs