Tecnología de la Programación

Tema 3. Clases y objetos
Primera parte

Conceptos generales de la POO

- Los objetos (información que almacenan y mensajes que aceptan) se declaran típicamente por medio clases
- Una clase es una plantilla de objetos, que describe la información y mensajes aceptados por un conjunto de objetos
- Una clase es a un objeto los que un tipo es a una variable en un lenguaje de programación imperativa
- El mecanismo de creación de objetos a partir de clases se denomina creación de ejemplares
- En casi todos los lenguajes de POO, todo objeto es un ejemplar de alguna clase

Conceptos generales de la POO

- La recepción de un mensaje se corresponde con la ejecución de un método (o función de manipulación de la información de un objeto)
- El funcionamiento del sistema software viene determinado por la existencia de una colección de objetos (agentes autónomos) que interactúan entre sí por medio del paso de mensajes.
- Un programa orientado a objetos es un conjunto de clases que describen el comportamiento de los objetos del sistema.

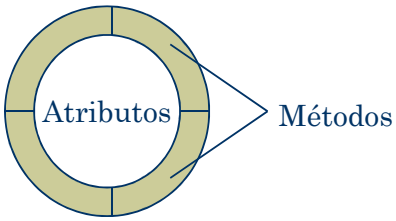
Conceptos generales de la POO

- Un lenguaje de POO es aquél que favorece y casi fuerza el desarrollo de programas orientados a objetos
- Un lenguaje de POO ha de reunir al menos las siguientes capacidades
  - Encapsulamiento y ocultación
  - Herencia
  - Polimorfismo
  - Vinculación dinámica

Encapsulamiento y ocultación

- Dos principios guían la definición de objetos
  - Agrupamiento de datos y funciones para manejarlos (encapsulamiento)
  - Privacidad de todo aquello que no se precise para utilizar un objeto (ocultación)

Representación intuitiva



Clase

- Una definición de clase comprende:
  - Cabecera
  - Campos o atributos:
    - Variables
    - Constantes
  - Métodos:
    - Funciones
    - Constructores
    - Bloques de inicialización static
    - Destructores (finalizador)

Convenio de nombres

- Nombres de clase: Sustantivo.
  - Primera letra (de cada palabra) en mayúsculas:
    - NombreDeClase
- Nombres de métodos: Verbo.
  - Primera letra en minúscula:
    - nombreDeMétodo
- Nombres de atributos y variables: Nombre significativo (normalmente sustantivo)
  - Primera letra en minúscula:
    - nombreDeAtributo
- Nombres de constantes: Nombre significativo (normalmente sustantivo)
  - Mayúsculas.
  - Separación con subrayados (guión bajo).
    - NOMBRE_DE_CONSTANTE

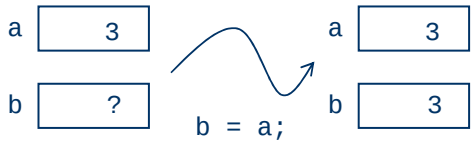
Contenedor y referencia

- Variables
  - Variable como contenedor de información (tipos básicos)
  - Variable como referencia a un objeto
- Creación
  - Contenedor ⇒ espacio reservado
  - Referencia ⇒ uso del operador new
- Asignación
  - Contenedor => Cambio del contenido
  - Referencia => Cambio del objeto apuntado

Contenedor y referencia

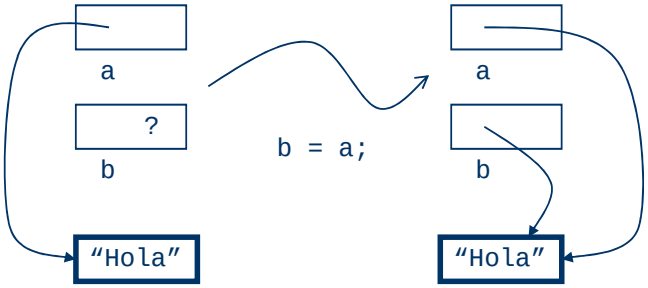
Contenedor

```
int a = 3;
int b;
```



Referencia

```
String a = new String("Hola");
String b;
```



Atributos

- Inicialización
  - Declaración:

```
public class EjemploDeClase {
        private TipoDelAtributo atributoDelObjeto=VALOR;
...
}
```

  - Constructora:

```
public EjemploDeClase {
        atributoDelObjeto=VALOR
}
```

Atributos

```
public class EjemploDeClase {
    private int atributoEntero=1;
    private String atributoCadena = "ejemplo de cadena";
    private boolean atributoBooleano = true;
}
```

- Tener atributos públicos es en general una muy mala práctica y está justificado en muy pocos casos, como en el de la definición de constantes que veremos más adelante.

Métodos

- Un objeto también puede poseer métodos
- Los métodos se definen en la declaración de la clase
- La definición de un método tiene las siguiente sintaxis

  tipoDevuelto nombreMétodo(tipo nombre, ...)

  { instrucciones }
- Si el procedimiento no devuelve nada, se usa void
- Se puede acceder a
  - parámetros y variables locales del método
  - atributos del objeto
    - Es conveniente que se haga usando *this* (referencia al propio objeto)

this

- *this* es una referencia al propio objeto con el que se está invocando el método

```
public class EjemploDeClase {              // Una variable local no debe tener el mismo nombre que un atributo
    private int atributoEntero=1;
    private void ejemploDeMetodo1() {
        this.atributoEntero=2; // asigna el valor 2 al atributo
        atributoEntero=3;      // asigna el valor 3 al atributo
    }
    private void ejemploDeMetodo2() {
        int atributoEntero; // Define una variable que se llama igual que el atributo
        atributoEntero = 2; // Esta asignación no supone modificación del atributo, aunque se llamen igual
    }
    private void ejemploDeMetodo3(){
        int atributoEntero = 2; // Define una variable que se llama igual que el atributo.
        this.atributoEntero=atributoEntero; // asigna el valor local de la variable al atributo
    }
}
```


Constructoras

- Método que inicializa el objeto en su creación
- Se llama automáticamente cuando se crea un objeto
- En Java implica el uso del operador new
- Su nombre es igual que el de la clase y no tiene tipo de retorno
- Como regla general, tras la invocación de una constructora no pueden quedar atributos sin inicializar
- Java proporciona un constructor sin parámetros por defecto que deja de estar disponible cuando se añade algún constructor

A a1 = new A();

- a1 es un ejemplar de la clase A (una referencia)
- A es la clase del objeto a1

Constructoras

- La constructora puede tener parámetros
- Se pueden colocar varias constructoras, cada una con distintos parámetros
- Durante la creación de un objeto, se invoca aquélla que encaja con los argumentos

Constructoras

```
public class EjemploDeClase  {
        private int atributoEntero =1;

         public EjemploDeClase ()
         { this.atributoEntero =2; }
         public EjemploDeClase (int nuevoValor )
        { this.atributoEntero = nuevoValor ; }

public static void main ( String args []) {
        EjemploDeClase unaVariable = new EjemploDeClase ();          // crea un objeto donde el atributo vale 2
        EjemploDeClase otraVariable = new EjemploDeClase (5);        // crea un objeto donde el atributo vale 5
}
}
```

Constructoras

```
class A
{
  private int x, y;
  public A() { x = 0; y = 0; }
  public A(int ix, int iy) { x = ix; y = iy; }
  public A(A from) { x = from.x; y = from.y; }
  ...
}

A a1 = new A(); a1.print();      // 0 0
A a2 = new A(1,2); a2.print();   // 1 2
A a3 = new A(a2); a3.print();    // 1 2
```


Constructoras

```
class A
{
  private int x, y;
  public A() { this.x = 0; this.y = 0; }                    // usando this
  public A(int ix, int iy) {this.x = ix; this.y = iy; }
  public A(A from) {this.x = from.x; this.y = from.y; }
  ...
}

A a1 = new A(); a1.print();      // 0 0
A a2 = new A(1,2); a2.print();   // 1 2
A a3 = new A(a2); a3.print();    // 1 2
```

Destructoras

- Método cuyo cometido es liberar los recursos que el objeto ha solicitado desde su creación:
  - Ficheros abiertos,
  - Memoria,
  - Puertos de comunicaciones ...
- Sólo hay un destructor por clase (por defecto o definido por el programador)

Destructoras

- En C++ :
  - No existe recolección de basura, por lo que la liberación de la memoria es tarea del programador.
  - Hay que implementar los destructores.
- En Java:
  - Es el sistema el que se encarga de destruir los objetos.
  - El entorno de ejecución de Java dispone de un recolector de basura que destruye los objetos (liberando su memoria) cuando detecta que los objetos ya no son accesibles.
  - Cada clase puede incorporar opcionalmente un método que se debe llamar "public void finalize()..."
    - Realiza tareas de cierre de recursos en clases que lo requieran (e.g. clases que realicen entrada/salida).

Destructoras

```
public class EjemploDeClase {
        public static final int TAM_POR_DEFECTO = 10;
        private int []vectorEnteros;

        public EjemploDeClase() {
          this.vectorEnteros = new int[TAM_POR_DEFECTO];
        }
        public EjemploDeClase(int tam){
          this.vectorEnteros = new int[tam];
        }
        protected void finalize() throws Throwable{
        // A pesar de haber pedido memoria para el atributo, en Java no hace falta liberarlo, pues ya lo hará el
        // recolector de basura .
        }
}
```


Métodos

- Métodos accedentes
  - Nos devuelven el valor de un atributo
  - Llamados métodos *get*
- Métodos mutadores
  - Nos permiten ajustar el valor de un atributo
  - Llamados métodos *set*

- Clase Fecha

```
public int getDia()
{ return this.dia; }
public void setDia(int dia)
{ this.dia = dia; }
```

Métodos

- Invocación de métodos
  - dentro de la clase que lo define (menos en main)
    - nombreMétodo(parámetros) ⇔ this.nombreMétodo(parámetros)
  - fuera de la clase en la que se define (y en main)
    - objeto.nombreMétodo(parámetros)

Métodos

- Ejemplo

```
class A
{
  private int x, y; // Atributos
  public void set(int vx, int vy) { x = vx; y = vy; }// Método
  public void incx() { x++; } // Otro método
  public void print() { System.out.println(x + " " + y); }
}
A a= new A();
a.set(10, 20);
a.print();      // 10 20
a.incx();
a.print();      // 11 20
incx();         // error, falta indicar el objeto
A a2= new A();
a2.set(5, 6);
a2.print();     // 5 6
a.print();      // 11 20
```

Métodos

- Ejemplo

```
class A
{
   ...
  public void otro()
  {incx(); print();}
}
A a= new A();
a.set(10, 20); a.otro();      // 11 20
a.incx(); a.print();      // 12 20
otro();         // error, falta indicar el objeto
```


Clases y objetos

- Las variables se pueden inicializar con la referencia nula

```
A a;
a = null;
a.x = 0;    // error, a es la ref. Nula

A a2 = new A();
a2 = null;  // se pierde el objeto
            // (recolector de basura)
```

Clases y objetos

```
public class OtraClase {
…
}
public class EjemploDeClase {
        private OtraClase atributo ;
        public EjemploDeClase () { this.atributo = new OtraClase (); }
        public EjemploDeClase (OtraClase nuevoValor ){ this.atributo = nuevoValor ; }

public static void main ( String args [])  {
        EjemploDeClase unaVariable =new EjemploDeClase ();
        EjemploDeClase otraVariable =new EjemploDeClase (new OtraClase ());
}
}
```


Clases y objetos

- Una referencia puede ser pasada como argumento y puede ser devuelta por un método

```
class A
{
        private int x, y;
        void copyFrom(A from) { x = from.x; y = from.y }
        // Se puede acceder a los atributos de objetos de la misma clase
        A makeCopy()
                { A acopy = new A(); acopy.set(x, y); return acopy;}
}
A a1 = new A(); a1.set(1,2);
A a2 = new A(); a2.copyFrom(a1);
a2.incX(); a2.print(); // 2 2
A a3 = a1.makeCopy();
a3.print(); // 1 2
```

Clases y objetos

- Se pueden crear matrices de referencias a objetos
- No se pueden crear matrices de objetos

```
A[] aArray = new A[10]; //se inicializa la matriz
aArray[0].x = 1;    // error, aArray[0] es nulo
for (int i = 0; i < aArray.length; i++)
   aArray[i] = new A();
```


Atributos de clase

- Atributos de clase (estáticos) ⇔ se encuentra en la clase (static) y no en los objetos

```
class A
{
   private int iv;                  // variable de ejemplar (atributos)
   private static int cv;           // variable de clase
                                    (atributo estático o atributo de clase)
   public A() { iv = 0; } // Constructor para los objetos
   static { cv = 0; } // Inicializador de la clase

  public void inc()
  {
   iv++; cv++;
   System.out.println(iv + " “ + cv);
  }
}
```

Atributos de clase

- Los atributos de clase se modifican en el acceso de cada objeto

```
A a1= new A();
A a2= new A();
a1.inc(); // 1 1
a2.inc(); // 1 2 Modificada por a2
a1.inc(); // 2 3 Modificada por a1
```


Atributo de clase finales

- Atributos de clase (estáticos) cuyo valor no se puede modificar
- final + static = constante de clase

```
final static int MAXALUMNOS = 50;
```

- public static final double PI = 3.1415....;(Math.PI)

Métodos de clase

- Un método de clase (static) es un método que sólo accede a variables de la clase

```
public static void inc2() { cv++; }
public static void inc3()
{
        iv++;   // error iv es de un objeto
}
pulic static void inc4()
{
        inc();  // error inc necesita un obj.
        inc2(); // Ok, porque Inc2 es static
}
```


Métodos de clase

- Los métodos de clase se usan para implementar funciones de utilidad (que sólo actúan sobre sus parámetros)
- Invocación => NombreClase.nombreMétodo(parámetros)
  - Integer.parseInt(“23”);
  - System.out.println(“Hola”);
  - Math.cos(0); // coseno