Java: Clases y Objetos

Clases

- La unidad fundamental de programación en Java es la clase
- Un programa Java está formado por un conjunto de clases
- Una clase es una "plantilla" que describe un conjunto de objetos con atributos y comportamiento similares
- Un programa Java en ejecución crea y manipula (mediante llamadas a métodos) objetos concretos (ejemplares o instancias)

Clases

- Cada objeto es un ejemplar de una clase
  - Cuando se invoca un método de un objeto, se mira en el código de su clase las acciones a ejecutar
  - Un objeto puede usar otros para realizar su trabajo

Clases

- Una definición de clase comprende:
  - Cabecera ← estado del objeto
  - Campos o atributos: ← comportamiento
    - Variables
    - Constantes
  - Métodos:
    - Funciones
    - Constructores
    - Bloques de inicialización static
    - Finalizador

Definición de una clase

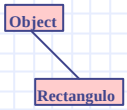
```
[Modificadores] class NombreClase [extends SuperClase] {
       // definición de los atributos de la clase
       tipo1  identificador1;
       tipo2  identificador2;
……. . ……. .
       // definición de los métodos de la clase
       tipoDevuelto nombreMetodo1 (listaParametros) {
      //instrucciones del método1
       }
       tipoDevuelto nombreMetodo2 (listaParametros) {
      //instrucciones del método2
       }
……. . ……. .
}
```


Modificadores de clase

- Modificadores:
  - **public** class NombreClase // visible fuera del paquete
    - Normalmente la clase se almacena en un fichero NombreClase.java
    - En un fichero .java puede haber como mucho una clase `public`
  - class C **extends** superclase // la clase hereda de otra
    - Sólo herencia simple (una sóla superclase)
    - Si no aparece **extends** la clase definida hereda (es una subclase) de un objeto general del sistema llamada **Object**

Modificadores de clase

```
public class Rectangulo{
        int x;
        int y;
        int ancho;
        int alto;
   // faltan los métodos de Rectángulo
}
```



Variables, objetos y referencias

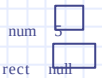
- Una variable de un determinado tipo simple proporciona
  - Capacidad para almacenar un valor simple
  - Un conjunto predifinido de operadores

Variables, objetos y referencias

- Un objeto de una determinada clase proporciona
  - Capacidad para almacenar diversos valores (atributos)
  - Define su propio conjunto de métodos para operar sobre las instancias o ejemplares de esa clase
  - Cuando se declara un objeto de una clase se crea una referencia a una instancia de dicha clase
    - Incialmente toma el valor *null* porque no existe objeto al que referirse
    - No son punteros con los que se pueda trabajar directamente

Variables, objetos y referencias

```
int num = 5;
Rectangulo rect;
```



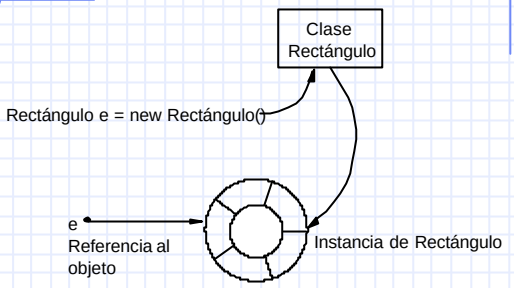
Variables, objetos y referencias

- La declaración de tipos primitivos reserva memoria
- La declaración de tipos no primitivos no reserva memoria.
- Los objetos declarados no son objetos, sino punteros a objetos.
- Antes de utilizar un objeto dentro del programa es necesario reservar el espacio necesario

Creación de instancias (objetos)

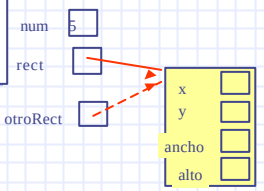
- Operador *new*
  - Crea una instancia o ejemplar de la clase indicada y devuelve una referencia a dicho objeto
  - Se reserva espacio de memoria para los datos del objeto
    - Un ejemplar es una copia individual de la plantilla de la clase que tiene su propio conjunto de datos

Creación de instancias (objetos)

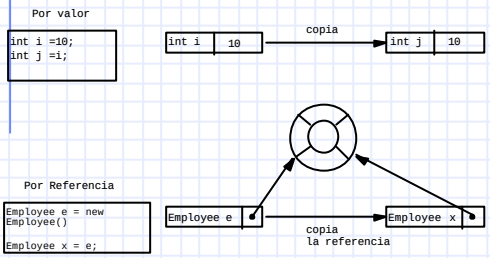


Creación de instancias (objetos)

```
int num = 5;
// crea una referencia
Rectangulo rect;
rect = new Rectangulo();
Rectangulo otroRect;
otroRect = rect;
// otroRect se refiere al
mismo
// rectangulo que rect
```



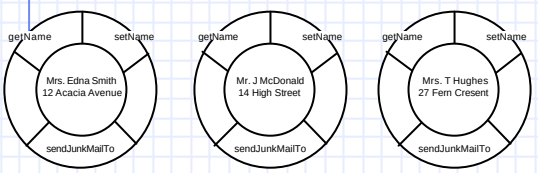
Asignación



Objetos

Podemos instanciar varios objetos de una clase

Los métodos disponibles son los mismos



Creación de instancias o ejemplares

Cuando se crea un objeto, las variables miembro (atributos) se inicializan al reservar memoria con los siguientes valores:

byte, short, int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000' (null)
boolean	false
tipos de referencia	null

Las variables locales a un método se deben inicializar. Éstas no se inicializan automáticamente.

Acceso a los atributos de un objeto

- Desde un objeto se puede acceder a los atributos o miembros con la siguiente sintaxis
  referenciaObjeto.atributo;

Acceso a los atributos de un objeto

```
public class Rectangulo {
    int x;
    int y;
    int ancho;
    int alto;
    //faltan las funciones miembro
    public static void main(String args[ ]) {
     Rectangulo rect;
     rect = new Rectangulo();
     rect.x = 5;
     rect.y = 7;
     rect.ancho = 4;
     rect.alto = 3;
     System.out.println( "x = " + rect.x + ”
                         y = " + rect.y );
     System.out.println( "ancho = " + rect.ancho
              + ” alto = " + rect.alto ); } }
```


Declaración de métodos

- Funciones declaradas en la clase y que determinan su comportamiento

```
tipoDevuelto nombreMetodo
(listaParametros) {
    //instrucciones del método
}
```


Declaración de métodos

```
// calcula la superficie y la devuelve como un
número entero
int calcularSuperficie(){
        int area;
        area = ancho * alto;
        return area;
}
// muestra los valores pero no devuelve nada
void mostrarValores(){
  System.out.println( "x = " + x + " y = " + y );
  System.out.println( "ancho = " + ancho + " alto
= " + alto );
}
```


Parámetros

- El lenguaje Java sólo pasa los argumentos de tipos básicos por valor.
- Cuando se pasa un objeto instanciado como argumento a un método, el valor del argumento es el puntero al objeto.
- Los contenidos del objeto se pueden cambiar dentro del método al que se ha llamado (los objetos se pasan por referencia), pero el puntero no se puede cambiar.

Llamada a los métodos

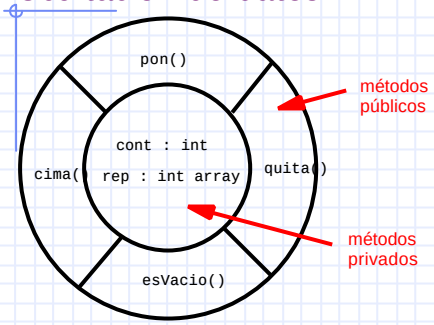
- La invocación a los métodos desde una instancia se hace mediante el operador de acceso (.)

referenciaObjeto.nombreMetodo(listaArgumentos);

Llamada a los métodos

```
public class Rectangulo {
   int x; ……..
   int calcularSuperficie(){ ……...    }
   void mostrarValores(){…….}
   public static void main(String args[ ]) {
      Rectangulo rect;
      rect = new Rectangulo();
      rect.x = 5; rect.y = 7;       rect.ancho = 4;
      rect.alto = 3;
      int area = rect.calcularSuperficie();
      rect.mostrarValores();
      System.out.println( "Superficie: " + area );
      System.out.println( "x = " + rect.x + ” y = " +
rect.y );
      System.out.println( "ancho = " + rect.ancho + ”
alto = " + rect.alto ); }
}
```

Ocultación de datos



Ocultación de datos

- La palabra reservada private permite una accesibilidad total desde cualquier método de la clase, pero no desde fuera de esta.

Ocultación de datos

```
public class Date {
  private int day, month, year;
  public void tomorrow () {
    this.day = this.day + 1;
  }//tomorrow
}//Date
public class DataUser {
  public static void main (String args[]){
    Date mydate = new Date();
    mydate.day = 21; //Incorrecto
  }
}
```


Ocultación de datos

Como los datos son inaccesibles, la única manera de leer o escribirlos es a través de los métodos de la clase. Esto proporciona consistencia y calidad.

Supongamos una clase que permite acceso libre:

```
MyDate d = new MyDate();
d.day = 32; // día no valido
d.month = 2;
d.day = 30; // posible pero incorrecto
d.month = d.month +1; // no se controla
```


Ocultación de datos

Regla: definir datos privados y métodos accesores y modificadores públicos

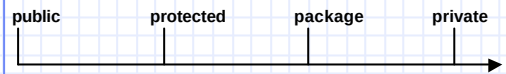
Encapsulación

Ocultar los detalles de implementación de la clase.

Fuerza al usuario a utilizar una interfaz para acceder a los datos.

Hace que el código sea más fácil de mantener.

Acceso a clases



Al menos una clase o interface de una unidad de compilación (package) debe ser public

Sobrecarga de métodos

Se puede utilizar:

```
public void print(int i)
public void print(float i)
public void print(String i)
```

La lista de argumentos tiene que ser diferente.
El tipo que devuelve puede ser diferente.

Constructores

- Método que inicializa el objeto en su creación
- Se llama automáticamente cuando se crea un objeto
- Su nombre es igual que el de la clase y no tiene tipo de retorno
- Java proporciona un constructor sin parámetros por defecto que deja de estar disponible cuando se añade algún constructor

Constructores

```
public class Rectangulo{
        int x;
        ……..…..
// constructor
  public  Rectangulo(int x1, int y1, int
w, int h){
        x=x1;
        y=y1;
        ancho=w;
        alto=h;   }
……..…..
```


Constructores

```
 public static void main(String args[ ]) {
        Rectangulo rect;
        rect = new Rectangulo(5, 7, 4, 3);
        int area=rect.calcularSuperficie();
        rect.mostrarValores();
        System.out.println( "Superficie: "
+ area );
     …….
   }
}
```


Constructores

- Pueden sobrecargarse
  - Una clase puede tener más de un constructor
    - Si no se declara ninguno se hereda el de la superclase (o el de Object)
  - Se puede crear un objeto con distintos tipos de parámetros
- Pueden llamar al constructor del padre
  - super(argumentos);
    - Debe ser la primera instrucción del constructor
- o a otros constructores de la misma clase
  - this(argumentosPorDefecto);

Constructores

```
public class Empleado {
  private String nombre;
  private int salario;
  public Empleado(String n, int s){
     nombre = n;
     salario = s;  }
  public Empleado(String n){
    this (n,0);  }
  public Empleado( ){
    this ("Desconocido");  }
}
```


El constructor por defecto

- Existe para cualquier clase
- Permite crear una instancia de un objeto con el método
  new Xxx().
- La definición de un constructor invalida al constructor por defecto.

Referencia a objeto *this*

- Referencia especial que utilizada dentro de un método de cualquier clase se refiere a la instancia actual
  - Permite parámetros con igual nombre que atributos
  - Posibilita la llamada a otros constructores

Referencia a objeto *this*

```
class Rectangulo{
       int x;
       …….……..
       int ancho;
       int alto;
// constructor
   Rectangulo(int x1, int y1, int w, int h){
       x=x1;   y=y1; ancho=w; alto=h;   }
// otro constructor polimorfico
   Rectangulo(int ancho, int alto){
       x=0;   y=0;
       this.ancho= ancho;
       this.alto= alto;   }
```

Referencia a objeto *this*

```
public static void main(String args[ ]) {
      Rectangulo rect;
      rect = new Rectangulo(5, 7, 4, 3);
      rect.mostrarValores();
      Rectangulo nuevo;
      nuevo = new Rectangulo(6, 9);
      rect.mostrarValores();
   ……..
  }
}
```


Variables (static) de la clase

- Pertenece a todas las instancias de la clase.
- Puede estar como pública o como privada.
- Si está marcada como pública, se puede acceder desde fuera de la clase, sin necesidad de una instancia de la clase.

Variables (static) de la clase

```
public class Count {
    private int serialNumber;
    private static int counter = 0;
    public Count () {
            counter++;
            serialNumber = counter;
    }
}
```

Se le suele llamar variable clase.

Variables (static) de la clase

- Desde fuera de la clase se puede acceder a una variable static si no se marca como private (que es lo habitual).
- Las variables de tipo static son, en algunos aspectos, parecidas a las variables globales de algunos lenguajes.
- Las instancias de la clase comparten la variable static

Métodos (static) de la clase

- Un método static se puede llamar sin necesidad de una instancia de la clase a la que pertenecen.
- Con los métodos estáticos no se puede utilizar el puntero this, ya que no se necesitan instancias.

Métodos (static) de la clase

```
public class GeneralFunction {
  public static int add(int x, int y) {
  return x + y;     }
}
public class UseGeneral {
  public void método () {
    int a = 9;
    int b = 10;
    int c = GeneralFunction.add(a, b);
    System.out.println ("add devuelve: "
  + c";     }
}
```


Métodos (static) de la clase

- Los métodos estáticos, sólo pueden acceder a sus propios argumentos y a las variables estáticas.
- El método main() es estático, porque la aplicación tiene que acceder a él para ejecutarse, antes de que realice cualquier instancia.
- Los métodos estáticos no se pueden sobreescribir

Inicializadores estáticos

- Bloques de inicialización estática
- Una clase puede tener código estático que no exista dentro del cuerpo de un método.
- El bloque estático se ejecuta sólo una vez, cuando se carga la clase.
- Los diferentes bloques dentro de una misma clase, se ejecutan en el orden en que aparecen.

Inicializadores estáticos

```
class Universidad {
  private static Vector profesores =
new Vector();

  static {
    profesores.addElement("Luis");
    profesores.addElement(“Balta");
    profesores.addElement(“Antonio");
  }
}
```

Puede servir para inicializar algunas variables de clase

Inicializadores estáticos

```
public class StaticInitDemo {
static int i = 5;
 static {
 System.out.println ("Static code i = "+i++);
  }
}
public class Test {
   public static void main (String args []){
     System.out.println ("Main code: i =" +
                StaticInitDemo.i);   }
}
imprimirá:
Static code: i = 5
Main code: i = 6
```


La palabra clase final

Si se aplica esta palabra a una clase, esa clase no puede tener subclases.

Por ejemplo, la clase java.lang.String, es una clase final y se ha definido así por motivos de seguridad.

La palabra clase final

Los métodos marcados con esta palabra no se pueden sobreescribir.

Los métodos marcados como estáticos o privados son finales automáticamente.

Si una variable se marca con la palabra reservada final, se convierte en una constante. Si se intenta cambiar el valor de cualquier variable final,se produce un error.