Tema 6: Clases

Antonio J. Sierra

Índice

Fundamentos (I)

- La clase es el núcleo de Java.
- La clase constituye la base de la programación orientada a objetos en Java.
- Cualquier concepto que desee implementar en un programa Java debe ser encapsulado en una clase.
- Las clases creadas hasta ahora se utilizaban para encapsular al método **`main()`**, lo que nos ha permitido mostrar las bases de la sintaxis de Java.

Fundamentos (II)

- Define la forma y la naturaleza de un objeto.
- Representa un nuevo tipo de dato. Una vez definido, este nuevo tipo se puede utilizar para crear objetos de ese tipo.
- Una clase es un template (modelo) para un objeto y un objeto es una instancia de una clase.
- Debido a que un objeto es una instancia de una clase, a menudo se utilizan las palabras objeto e instancia de manera indistinta.

Definición

- La definición de una clase incluye
  - su forma y
  - naturaleza.
- Esto se realiza especificando los datos de la clase y el código que opera sobre ellos.
- El código de una clase define la interfaz a sus datos.
- Una clase se declara utilizando la palabra clave **class**.

Forma general de la definición

```
class nombre_de_clase {
        tipo variable_de_instancia1;
        //...
        tipo variable_de_instanciaN ;

        tipo nombre_de_método1(lista_de_parámetros){
                // cuerpo del método
        }
        //...
        tipo nombre_de_métodoN (lista_de_parámetros){
                // cuerpo del método
        }
}
```

Variables de instancia y métodos

- Los datos o variables definidos en una clase se llaman **variables de instancia**.
- El código está contenido en los **métodos**.
- En conjunto, los métodos y variables definidos dentro de una clase son los **miembros** de la clase.
- Las variables de instancia suelen estar modificadas por los métodos definidos en esa clase. Por lo que se puede decir que los métodos son los que determinan cómo se puede utilizar los datos de una clase.

Los objetos

- Para obtener objetos de una clase es necesario realizar dos pasos:

1. Declarar una variable del tipo de la clase.
   - Esta variable no define un objeto, sino que es simplemente una variable que puede referirse a un objeto.
2. Obtener una copia física y real del objeto y asignarla a esa variable.
   - Esto se puede hacer utilizando el operador **`new`**.

new

- El operador **`new`** permite una asignación dinámica
  - en tiempo de ejecución,
  - reserva memoria para un objeto y
  - devuelve una referencia al mismo.
  - esta referencia se puede almacena en la variable.
  - todos los objetos en Java deben ser asignados dinámicamente.
- Si no es capaz de reservar memoria se produce una excepción.

Sintaxis para el operador new

```
Variable = new nombre_de_la_Clase();
```

- **`Variable`** es una variable de la clase que se quiere crear.
- **`nombre_de_la_Clase`** es el nombre de la clase que está siendo instanciada.
- El nombre de la clase seguido por los paréntesis especifica el **constructor de la clase**.

Constructor

- Un constructor define qué ocurre cuando se crea un objeto de una clase.
- Si no se ha especificado explícitamente el constructor, entonces Java automáticamente utiliza un constructor por defecto.
- Se pueden definir explícitamente los constructores dentro de la definición de la clase.
- Los tipos de datos simples no necesitan constructor ya que no son objetos.

Ejemplo

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa();
    }
}
```

Ejemplo (detalle Primer paso)

```
Mesa miMesa = new Mesa();
```

- Los dos pasos anteriormente descritos se pueden escribir de la siguiente manera:

```
Mesa miMesa; //declara la referencia a un objeto
```

- Declara una referencia a un objeto del tipo Mesa.
- Cuando se ejecuta esta línea, miMesa contiene el valor **`null`**, que indica que todavía no referencia a un objeto.
- Cualquier intento de utilización de **`miMesa`** en esta situación provocará un error.

Ejemplo (detalle Segundo paso)

```
miMesa = new Mesa(); // reserva espacio para el objeto
```

- La segunda línea reserva memoria para un objeto real y asigna a **`miMesa`** una referencia a dicho objeto.
- cuando se ejecuta la segunda línea se puede utilizar **`miMesa`** como si fuese un objeto del tipo **`Mesa`**.
- En realidad, **`miMesa`** simplemente guarda la dirección de memoria del objeto real.

Ejemplo de declaración e instanciación de un objeto

Setencia

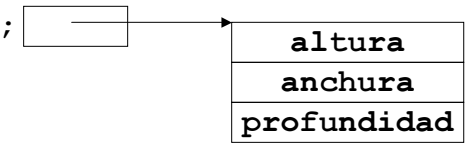
```
Mesa miMesa;
```

miMesa

null

```
miMesa = new Mesa();
```

miMesa



Asignación de objetos a variables referencia

```
Mesa m1 = new Mesa();
Mesa m2 = m1 ;
```

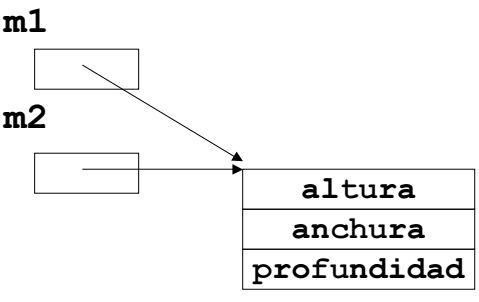
- No se asigna una copia del objeto.
- **m1** y **m2** se refieren al mismo objeto.

m1

m2

- **m2 = null;**

Lo “desengancha”



Ciclo de vida de un objeto

1. **Creación de un objeto**

   La sentencia `Mesa m1 = new Mesa();` tiene tres acciones:

   1. Declaración: el compilador puede usar la variable `m1` para referirse a un objeto `Mesa`.
      - La declaración no crea un objeto.
      - La declaración se realiza con tipo nombre;
   2. Instanciación: El operador **`new`** crea un nuevo objeto y lo pone en memoria.
   3. Inicialización: `Mesa()` es una llamada al constructor, que inicializa al objeto.
2. **Uso del objeto**. Par hacer referencia a variables o métodos del objeto mediante el punto (.).
3. **Eliminación**. Los objetos no usados.

Métodos

- Sintaxis general de un método:

```
tipo nombre_de_método(lista_de_parametros){
    //cuerpo del método
}
```

- `tipo` especifica el tipo devuelto por el método.
  - Si no devuelve ningún tipo, `void`.
- El valor devuelto: **`return expresion;`**

Ejemplo

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    void muestraMesa(){
       System.out.println("la mesa es de "+
       altura+"x"+ anchura+"x"+ profundidad);
    }
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa();
       miMesa.altura = 100.;
       miMesa.anchura = 120.;
       miMesa.profundidad = 90.;
       miMesa.muestraMesa();
    }
}
```

Devolución de un valor

- El tipo de dato devuelto por un método debe ser compatible con el tipo especificado por el método.
- Si un método devuelve un **`boolean`**, no se puede devolver un entero.
- La variable que recibe el valor devuelto por el método debe ser compatible también con el tipo especificado por el método.

Ejemplo

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    double area(){
        return (anchura* profundidad);
    }
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa();
       miMesa.altura = 100.;
       miMesa.anchura = 120.;
       miMesa.profundidad = 90.;
       System.out.println("El area es "+
                          miMesa.area());
    }
}
```

Métodos con parámetros

- Un **parámetro** es una variable definida por un método y que recibe un valor cuando se llama a ese método.
- Un **argumento** es un valor que se pasa a un método cuando éste es invocado.

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    void area(){
        return (anchura* profundidad);
    }
    void configuraDim(double h, double a, double p) {
       altura = h;
       anchura = a;
       profundidad = p;
  }
}
class Principal{
   public static void main (String args[]){
        Mesa miMesa = new Mesa();
        miMesa.configuraDim(100.0, 120., 90.);
        System.out.println("Area "+miMesa.area());
    }
}
```

Constructores

- Un constructor inicializa un objeto inmediatamente después de su creación.
- Tiene exactamente el mismo nombre que la clase en la que reside y sintácticamente es similar a un método.
- Una vez definido, se llama automáticamente al constructor después de crear el objeto, antes de que termine el operador **`new`**.
- No devuelven ningún tipo.

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    Mesa() {
       altura = 100.0;
       anchura = 120.;
       profundidad = 90.;
    }
    void area(){
        return (anchura* profundidad);
    }
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa();
       System.out.println("Area "+miMesa.area());
    }
}
```

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    Mesa(double h, double a, double p) {
       altura = h;
       anchura = a;
       profundidad = p;
    }
    void area(){
        return (anchura* profundidad);
    }
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa(100.0, 120., 90.);
       System.out.println("Area "+miMesa.area());
    }
}
```

this

- Permite hacer referencia al objeto que lo invocó.
- Puede ser utilizada dentro de cualquier método para referirse al objeto actual.
- **`this`** siempre es una referencia al objeto sobre el que ha sido llamado el método.

```
class Mesa{
    double altura;
    double anchura;
    double profundidad;
    Mesa(double h, double a, double p) {
        this.altura = h;
        this.anchura = a;
        this.profundidad = p;
    }
    void area(){
        return (anchura* profundidad);
    }
}
class Principal{
   public static void main (String args[]){
       Mesa miMesa = new Mesa(100.0, 120., 90.);
       System.out.println("Area "+miMesa.area());
    }
}
```

Recogida de basura

- Cuando no hay ninguna referencia a un objeto determinado, se asume que ese objeto no se va a utilizar más, y la memoria ocupada por el objeto se libera.
- No hay necesidad explícita de destruir los objetos, como sucede en C++.
- La recogida de basura ocurre de forma esporádica durante la ejecución de un programa y no se produce simplemente porque uno o más objetos hayan dejado de utilizarse.
- Las diferentes implementaciones de los intérpretes de Java siguen un procedimiento diferente cuando realizan la recogida de basura.

El método **`finalize()`**

- Objetos que ya no se van a seguir usando, llamada a **`finalize()`**.
  - Última oportunidad para hacer algo con el objeto.
  - **`java.lang.Object`**
- Formato

```
protected void finalize(){
    //código de finalización
}
```

Ejecución del recolector de basura

- Se puede pedir la ejecución del recolector de basura en cualquier momento:

```
System.gc()
```