Programación II

Tema 2: Programación orientada a objetos

Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I

Curso 2024/2025

Contenido

Declaración de una clase

modificadores `class` *NombreClase* `{`

```
    // Atributos
    // Constructores
    // Métodos
}
```

- Modificadores: de momento solo consideraremos `public`
- Convenio: nombre de la clase debe comenzar con mayúsculas
- Cada atributo se inicializa al valor por defecto de su tipo.

Ejemplo: una clase para representar complejos

Ejemplo: uso de la clase `Complejo`

```
Complejo c1 = new Complejo(3, 5);  // 3 + 5i
Complejo c2 = new Complejo(4);     // 4 + 0i
Complejo c3 = new Complejo();      // 0 + 0i
Complejo c4;

// c4 = c1 + c2 * c3
c4 = c1.sumar(c2.multiplicar(c3));
System.out.println(c4);

// c4 = (c1 + c2) * c3
c4 = c1.sumar(c2).multiplicar(c3);
System.out.println(c4);
```

Ejemplo: una clase para representar complejos (cont.)

Ejemplo: declaración de la clase `Complejo`

```
// Forma binómica
public class Complejo {
    // Atributos
    private double re;
    private double im;
    // Constructores
    ...
    // Métodos
    ...
}
```

```
// Forma polar
public class Complejo {
    // Atributos
    private double módulo;
    private double argumento;
    // Constructores
    ...
    // Métodos
    ...
}
```

Directamente accesibles solo desde la propia clase `Complejo`

Ejemplo: una clase para representar complejos (cont.)

Ejemplo: declaración de la clase `Complejo` (forma binómica)

```
public class Complejo {
    // Atributos
    private double re;
    private double im;
    // Constructores
    ...
    // Métodos
    public Complejo sumar(Complejo c) {
        return new Complejo(re + c.re, im + c.im);
    }
    ...
}
```


Ejemplo: una clase para representar fechas

Ejemplo: declaración de la clase `Fecha`

```
public class Fecha {
    // Atributos
    private int día;
    private int mes;
    private int año;
    // Constructores
    ...
    // Métodos
    ...
}
```

Directamente accesibles solo desde la propia clase `Fecha`

Constructores

- Indican cómo se inicializa un objeto.
- Llamados *automáticamente* al crear un objeto de una clase.
- Puede haber varios constructores para una misma clase.

Ejemplo: creación de objetos de la clase `Complejo`

```
Complejo c1 = new Complejo(3, 5);  // 3 + 5i
Complejo c2 = new Complejo(4);     // 4 + 0i
Complejo c3 = new Complejo();      // 0 + 0i
```

El constructor correspondiente es llamado de manera automática

Declaración de un constructor

modificador *NombreClase*(*parámetros*) {
    instrucciones
}

Ejemplo: clase `Fecha`

```
// Suponemos que los valores dados representan una fecha válida
public Fecha(int elDía, int elMes, int elAño) {
    día = elDía;
    mes = elMes;
    año = elAño;
}
```


Ejemplo: un constructor para la clase `Complejo`

Ejemplo: clase `Complejo` (forma binómica)

```
// Crea el número complejo a + b i
public Complejo(double a, double b) {
    re = a;
    im = b;
}
```

Ejemplo: clase `Complejo` (forma polar)

```
// Crea el número complejo a + b i
public Complejo(double a, double b) {
    módulo = Math.sqrt(a * a + b * b);
    argumento = Math.atan2(b, a);
}
```


Llamada explícita a un constructor: `this`

La primera sentencia de un constructor puede ser una llamada a otro constructor de la misma clase.

Ejemplo: clase `Complejo`

```
public Complejo(double a) {
    this(a, 0);
}

public Complejo() {
    this(0, 0);
}

// Complejo(double, double)
```

Ejemplo: clase `Fecha`

```
public Fecha() {
    this(1, 1 ,1970);
}

// Fecha(int, int, int)
```


Referencia al objeto actual: `this`

- En el cuerpo de un método o constructor, `this` actúa como una referencia al objeto actual.
- Permite el acceso a atributos que han quedado *ocultos* por la declaración de parámetros con el mismo nombre.

Ejemplo: constructor de la clase `Complejo`

```
public Complejo(double re, double im) {
    this.re = re;
    this.im = im;
}
```

Ejemplo: una clase para representar rectángulos

Ejemplo: clase `Rectángulo`

```
public class Rectángulo {
    private int x, y;  // Vértice superior izquierdo
    private int anchura, altura;

    public Rectángulo(int x, int y, int anchura, int altura) {
        this.x = x;
        this.y = y;
        this.anchura = anchura;
        this.altura = altura;
    }

    public Rectángulo(int anchura, int altura) {
        this(0, 0, anchura, altura);
    }

    ...
}
```


Constructor por defecto

- Cuando en una clase no se declara ningún constructor, se proporciona automáticamente un constructor sin parámetros:

```
public NombreClase() {
}
```

- Los atributos del objeto creado quedan inicializados a los valores por defecto de los tipos correspondientes.
- Si en una clase se declara algún constructor, ya no se dispone del constructor por defecto.

Ejemplo: una clase para representar puntos

Con constructor por defecto

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

    // Ningún constructor
    // (constructor por defecto)

    // Métodos
}
...
Punto p = new Punto();
```

Sin constructor por defecto

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

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

    // Métodos
}
...
Punto p = new Punto();  // Error
```


Métodos de acceso (*getters*)

Métodos que consultan el estado de un objeto sin modificarlo.

Ejemplo: clase `Fecha`

```
public class Fecha {
    ...
    public int getDía() {
        return día;
    }
    ...
}
...
Fecha fechaControl = new Fecha(22, 4, 2013);
System.out.println("Día: " + fechaControl.getDía());
```

Métodos modificadores (*setters*)

Métodos que modifican el estado de un objeto.

Ejemplo: clase `Fecha`

```
public class Fecha {
    ...
    // Suponemos que la nueva fecha es válida
    public void setDía(int día) {
        this.día = día;
    }
    ...
}

...
Fecha fechaControl = new Fecha(22, 4, 2013);
fechaControl.setDía(29);
```


Salida: método `toString`

```
public String toString() {
    ...
    // Devolver una cadena que represente el objeto.
    return cadena;
}
```

Ejemplo: clase `Fecha`

```
public String toString() {
    return día + "/" + mes + "/" + año;
}
```


Comparar objetos: método `equals`

Ejemplo: clase `Fecha`

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (!(obj instanceof Fecha))
        return false;
    Fecha otraFecha = (Fecha) obj;
    return día == otraFecha.día &&
           mes == otraFecha.mes &&
           año == otraFecha.año;
}
```


Atributos estáticos

Atributo estático: compartido por todos los objetos de una clase.

```
public class NombreClase {
    // Atributo (variable de instancia)
    private tipo nombreAtributo;

    // Atributo estático (variable de clase)
    modificador static tipo nombreAtributo;

    ...
}
```

Ejemplo: otra clase para representar fechas

```
public class Fecha {
    // Atributo (variable de instancia)
    private int díasDesdeOrigen;

    // Atributos estáticos (variables de clase)
    public static final int PRIMER_AÑO = 1800;
    public static final int ÚLTIMO_AÑO = 2500;

    private static final int[] díasHastaPrimeroDeMes
        = { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 };
    private static final int[] díasHasta1Enero;

    ...
}
```


Inicializador estático

Bloque de código que permite inicializar atributos estáticos previamente declarados.

```
static {
    // Instrucciones que se ejecutan
    // cuando se carga la clase.
}
```


Ejemplo: otra clase para representar fechas (cont.)

```
public class Fecha {
    ...
    static {
        díasHasta1Enero = new int[ÚLTIMO_AÑO - PRIMER_AÑO + 1];

        // Rellenar díasHasta1Enero
        díasHasta1Enero[0] = 0;
        for (int i = 1; i <= ÚLTIMO_AÑO - PRIMER_AÑO; i++) {
            díasHasta1Enero[i] = díasHasta1Enero[i - 1] + 365;
            if (añoBisiesto((i - 1) + PRIMER_AÑO))
                díasHasta1Enero[i]++;
        }
    }
    ...
}
```


Métodos estáticos

Método estático: invocado sin referencia a un objeto en particular.

```
public class NombreClase {
    ...
    // Método estático (método de clase)
    modificador static tipo nombreMétodo(parámetros) {
        // En el cuerpo del método no se permite el uso de this
        // ni el acceso a miembros no estáticos.
    }
    ...
}
```

Ejemplo: otra clase para representar fechas (cont.)

```
public class Fecha {
    ...
    public static boolean añoBisiesto(int año) {
        ...
    }

    public static int díasMes(int mes, int año) {
        ...
    }

    ...
}
```


Paquetes

Paquete: consta de un conjunto de clases relacionadas.

`java.lang`	Incluye las clases `Integer`, `Math`, `String` y `System`.
`java.util`	Incluye las clases `Scanner`, `Formatter`, `Arrays`, `Date` y `Random`
`java.io`	Usado para entrada/salida.

Cuadro: Algunos paquetes predefinidos de Java

Declaración de paquete

`package` *nombrepaquete*;

- Indica a qué paquete pertenece una clase.
- Primera sentencia del fichero en el que se define la clase.
- El nombre del directorio donde se almacena la clase debe coincidir con el nombre del paquete.

Nombre cualificado de una clase

nombrepaquete.*NombreClase*

Ejemplo

```
java.io.File fichero = new java.io.File(nombreFichero);
java.util.Scanner entrada = new java.util.Scanner(fichero);
...
entrada = new java.util.Scanner(java.lang.System.in);
...
java.lang.System.out.println(java.lang.Math.PI);
```

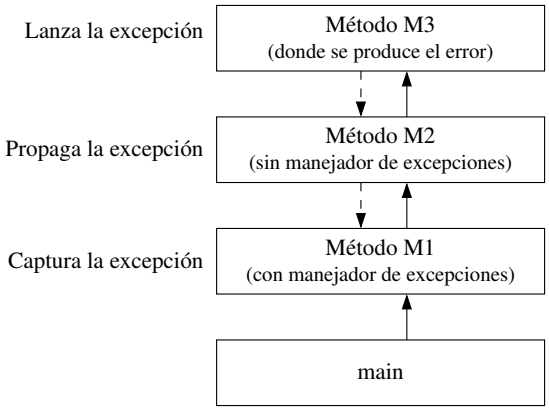
La directiva `import`

```
import nombrepaquete.NombreClase;
```

```
import nombrepaquete.*;
```

- Debe aparecer antes de la declaración de una clase.
- Su uso descuidado puede producir conflictos de nombres.
- El paquete `java.lang` se importa de forma automática.

Propagación de excepciones



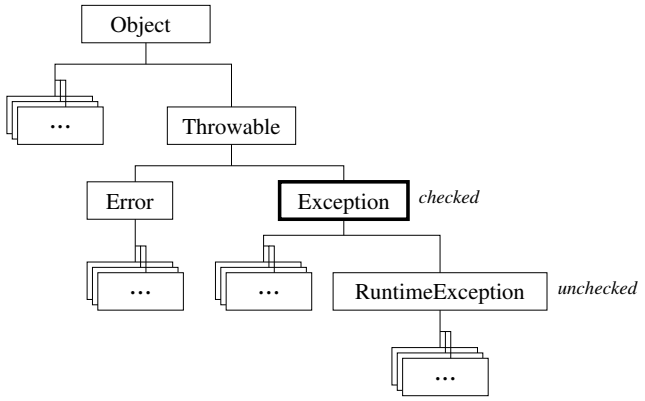
Excepciones

Una excepción es un suceso, ocurrido durante la ejecución de un programa, que interrumpe el flujo normal de instrucciones.

Ejemplos:

- División por 0
- Acceso a un índice no válido
- Fichero no existe
- Error de conversión

La clase `Throwable`



checked: obligan a capturarlas o a indicar explícitamente su propagación

Excepciones `RuntimeException`

`ArithmeticException`	Intento de división entera por cero.
`NumberFormatException`	Intento de conversión a un número de una cadena no numérica.
`IndexOutOfBoundsException`	Índice fuera de rango en un vector o en una cadena.
`NullPointerException`	Intento de utilizar `null` en un caso donde se requiere un objeto.

Cuadro: Algunas excepciones del tipo `RuntimeException`

Excepciones verificadas

`java.io.EOFException`	Final de fichero antes de lo esperado.
`java.io.FileNotFoundException`	Fichero no encontrado.
`java.io.IOException`	Excepciones de E/S.

Cuadro: Algunas excepciones del tipo *standard checked*

Especificación de las excepciones lanzadas por un método

modificadores tipoRetorno nombreMétodo(*parámetros*)
    `throws` *NombreExcepción1*, ..., *NombreExcepciónN*
{
    // Durante la ejecución del método se podría producir
    // alguna excepción de los tipos especificados.
}

Ejemplo: método `crearVectorDni`

```
public static String[] crearVectorDni(String nombreFichero)
        throws FileNotFoundException {
    Scanner entrada = new Scanner(new File(nombreFichero));
    ...
}
```


Lanzar una excepción: sentencia `throw`

`throw` *objetoThrowable*;

Ejemplo: pila de enteros

```
public class Pila {
    ...
    public int desapilar() throws ExcepcionPilaVacia {
        if (esVacia())
            throw new ExcepcionPilaVacia();
        ...
    }
    ...
}
```

Declaración de nuevas excepciones

```
// Declaración de una nueva excepción verificada
public class NombreExcepción extends Exception {
    ...
}
```

Ejemplo: ExcepcionPilaVacia

```
public class ExcepcionPilaVacia extends Exception {
    // De momento lo dejaremos vacío
}
```


Captura de excepciones: sentencia try-catch

```
try {
    // Código que puede lanzar alguna excepción
} catch (NombreExcepción1 e) {
    // Manejador de excepción
} ... {
} catch (NombreExcepciónN e) {
    // Manejador de excepción
} finally {
    // Código ejecutado siempre (se produzca o no excepción)
}
// Código posterior
```