

Polimorfismo

Polimorfismo

- El polimorfismo es una habilidad de tener varias formas; por ejemplo, la clase Jefe tiene acceso a los métodos de la clase Empleado.
- Un objeto tiene sólo una forma.
- Una variable tiene muchas formas, puede apuntar a un objeto de diferentes maneras.
- En Java hay una clase que es la clase padre de todas las demás: `java.lang.Object`.
- Un método de esta clase (por ejemplo: `toString()` que convierte cualquier elemento de Java a cadena de caracteres), puede ser utilizada por todos.

Polimorfismo

- Java permite apuntar a un objeto con una variable definida como tipo de clase padre.

```
Empleado e = new Jefe ();
```

- Sólo se puede acceder a las partes del objeto que pertenecen a la clase Empleado;
- Las partes específicas de la clase Jefe no se ven.
- Este efecto se consigue porque, para el compilador, e es sólo una variable de tipo Empleado (referencia estática), no Jefe (referencia dinámica).

```
e.mandar() //no compila
```

El operador instanceof

redundante

```
public class Empleado extends Object
public class Jefe extends Empleado
public class Contractor extends Empleado

public void método (Empleado e){
    if (e instanceof Jefe) {
        // Obtiene beneficios por su salario
    }
    else if (e instanceof Contractor) {
        //Obtiene tarifa por horas
    }
    else {
        //empleos temporales
    }
}
```

Conversión de objetos

Utiliza `instanceof` para verificar el tipo de objeto.

Restablecer la funcionalidad total de un objeto mediante una conversión.

Comprobar la conversión apropiada con:

La conversión hacia clases superiores en la jerarquía se hace implícitamente (con una asignación).

```
Empleado e = new Jefe ();  
  
((Jefe)e).mandar() //si compila  
//ejecuta mandar() de clase Jefe
```

Polimorfismo

5

Conversión de objetos

Conversión hacia abajo, han de ser hacia subclases y el compilador las comprueba.

El tipo del puntero se comprueba en tiempo de ejecución, cuando hay errores. Estos errores en tiempo de ejecución se llaman excepciones.

Conversión de objetos

```
if (e instanceof Jefe) {  
    Jefe m = (Jefe) e;  
    System.out.println("Este es el  
director de " +  
m.departamento);  
}  
//resto de la función
```

Casting

- Casting automático

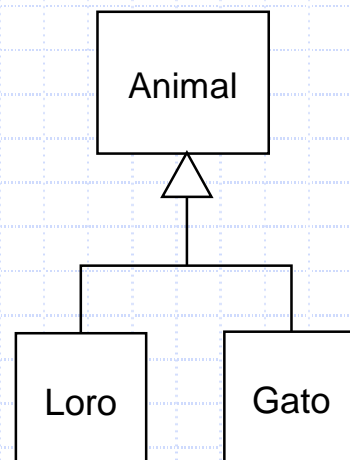
```
Loro c = new Loro();  
Animal a = c;
```

- Se necesita casting explícito

```
Animal a = new Loro();  
Loro c = (Loro) a;
```

- Error de compilación:

```
Loro c = new Loro();  
Gato d = (Gato) c;
```



Sobreescritura de métodos

Una subclase puede modificar los métodos que ha heredado del padre.

Una subclase puede crear un método con diferente funcionalidad al método del padre, pero con el mismo:

- Nombre
- Tipo de retorno
- Lista de argumentos

Recordar que los métodos, con el mismo nombre y lista de argumentos distinta, dentro de la misma clase, se denomina sobrecarga. La lista de argumentos es lo que indica al compilador que se invoca.

Polimorfismo

9

Sobreescritura de métodos

Llamada de métodos virtuales.

```
Empleado e = new Jefe;  
e.getDetails ();
```

Se comprueba el tipo de la referencia estática (Empleado) en tiempo de compilación y el tipo de la referencia dinámica (Jefe) en tiempo de ejecución.

¿Cual de los getDetails() se ejecutará, el de la clase Empleado o el de la clase Jefe?.

Polimorfismo

10

Sobreescritura de métodos

Para poder sobreescribir métodos en las clases descendientes se debe verificar:

- El tipo de retorno de los dos métodos ha de ser igual.
- El método de la subclase no puede ser menos accesible que el de la clase padre.
- El método de la subclase no puede provocar más excepciones que el método del padre.

Sobreescritura de métodos

```
public class Padre {  
    public void método () { }  
}  
  
public class Hijo extends Padre {  
    public void método () { }  
    //no puede ser private  
}  
  
public class OtraClase {  
    public void otroMetodo () {  
        Padre p1 = new Padre();  
        Padre p2 = new Hijo();  
        p1.método();  
        p2.método();  
    }  
}
```

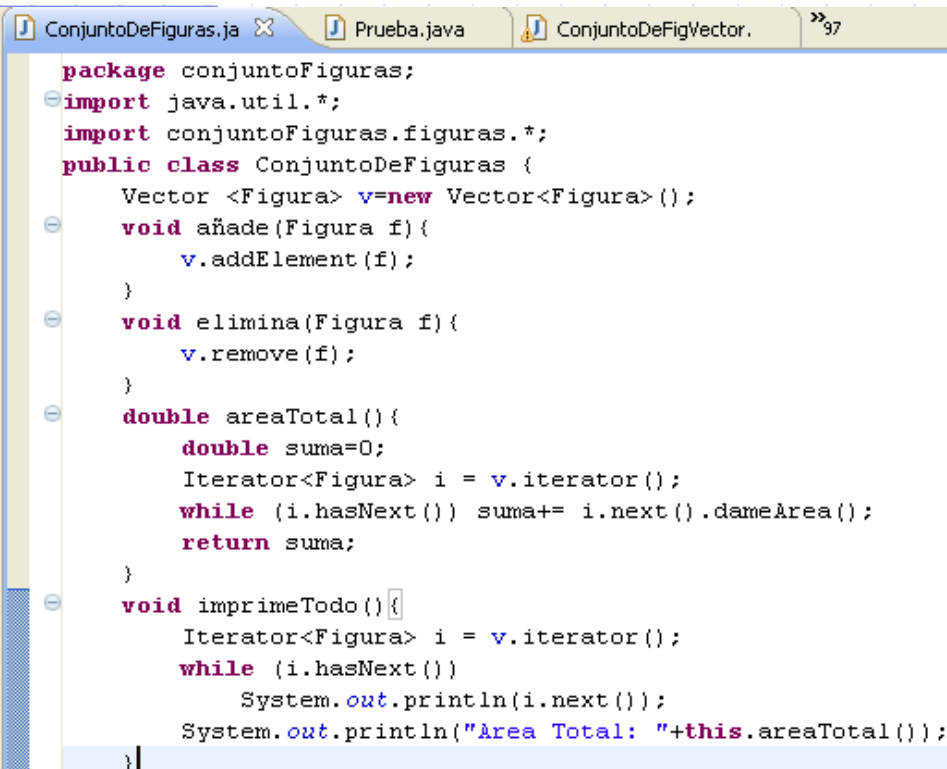
Ejercicio 1

Construye la clase **ConjuntoDeFiguras** para que gestione un array de figuras de cualquier clase. Entre sus métodos públicos debes incluir las siguientes operaciones con carácter polimórfico:

- Añadir una nueva figura
- Eliminar una figura
- Calcular el área total de las figuras almacenadas
- Imprimir por pantalla los datos de todas las figuras almacenadas.

Prueba la clase **ConjuntoDeFiguras** desde el programa principal usando objetos de las clases especializadas **Triángulo**, **Círculo** y **Cuadrado**.

Ejercicio 1: teniendo un Vector de Figuras



```
package conjuntoFiguras;
import java.util.*;
import conjuntoFiguras.figuras.*;
public class ConjuntoDeFiguras {
    Vector <Figura> v=new Vector<Figura>();
    void añade(Figura f){
        v.addElement(f);
    }
    void elimina(Figura f){
        v.remove(f);
    }
    double areaTotal(){
        double suma=0;
        Iterator<Figura> i = v.iterator();
        while (i.hasNext()) suma+= i.next().dameArea();
        return suma;
    }
    void imprimeTodo(){
        Iterator<Figura> i = v.iterator();
        while (i.hasNext())
            System.out.println(i.next());
        System.out.println("Área Total: "+this.areaTotal());
    }
}
```

Ejercicio 1: teniendo un Vector de Figuras. Usando for moderno

```
ConjuntoDeFiguras.java Prueba.java ConjuntoDeFigVector. »97

double areaTotal(){
    double suma=0;
    // Iterator<Figura> i = v.iterator();
    // while (i.hasNext()) suma+= i.next().dameArea();
    for(Figura f: v ) suma+=f.dameArea();
    return suma;
}

void imprimeTodo(){
    /*Iterator<Figura> i = v.iterator();
    while (i.hasNext())
        System.out.println(i.next()); */
    for(Figura f: v ) f.imprimir();
    System.out.println("Área Total: "+this.areaTotal());
}
```

Polimorfismo

15

Ejercicio 1: Pruebas

```
public static void main(String[] args) {
    ConjuntoDeFiguras c= new ConjuntoDeFiguras();
    c.añade(new Triangulo("Verde",3,4));
    c.añade(new Circulo("Rojo",5));
    c.imprimeTodo();
}
```

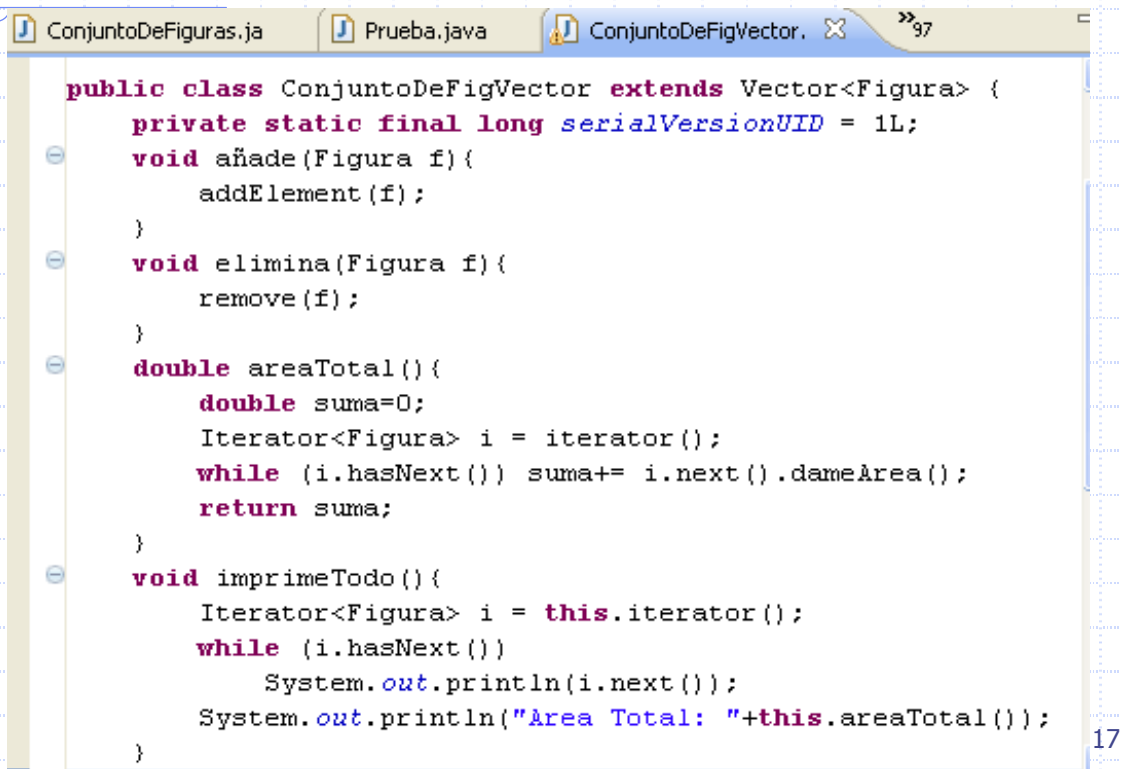
<terminated> ConjuntoDeFiguras [Java Application] C:\Archivos de programa\Java\jre6\bin\javaw

Color: Verde
Base: 3.0
Altura:4.0
Area: 6.0

Color: Rojo
Radio: 5.0
Area Total: 84.53981633974483

6

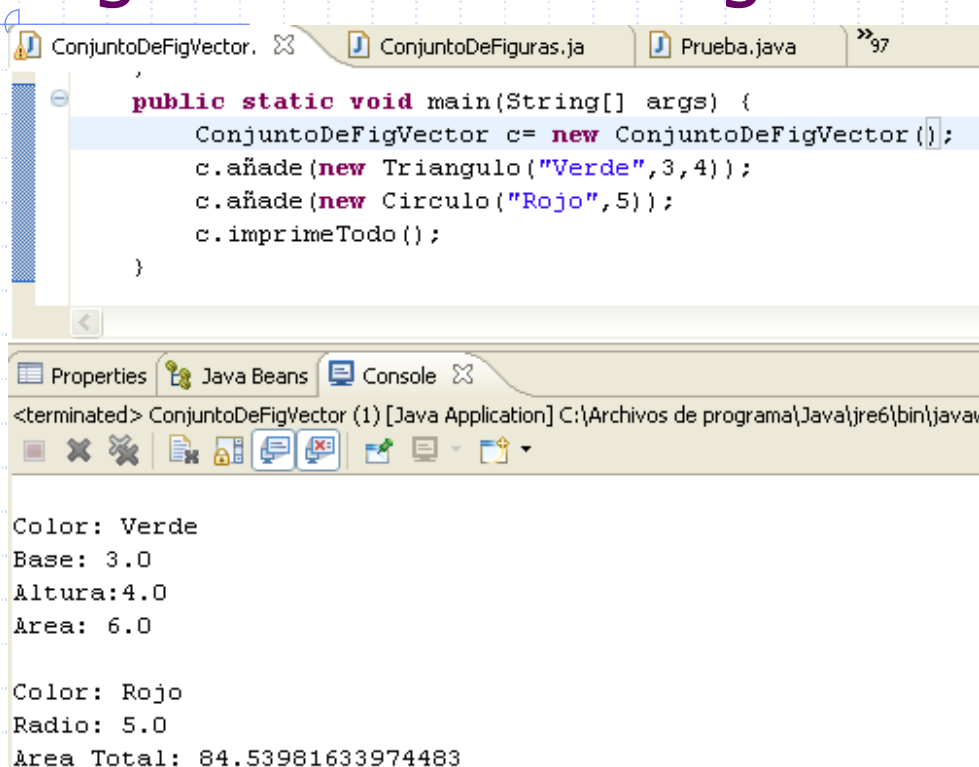
Ejercicio 2: siendo un Vector de Figuras.



```
public class ConjuntoDeFigVector extends Vector<Figura> {
    private static final long serialVersionUID = 1L;
    void añade(Figura f){
        addElement(f);
    }
    void elimina(Figura f){
        remove(f);
    }
    double areaTotal(){
        double suma=0;
        Iterator<Figura> i = iterator();
        while (i.hasNext()) suma+= i.next().dameArea();
        return suma;
    }
    void imprimeTodo(){
        Iterator<Figura> i = this.iterator();
        while (i.hasNext())
            System.out.println(i.next());
        System.out.println("Area Total: "+this.areaTotal());
    }
}
```

17

Ejercicio 2: siendo un Vector de Figuras. ¿cambia algo?



```
public static void main(String[] args) {
    ConjuntoDeFigVector c= new ConjuntoDeFigVector();
    c.añade(new Triangulo("Verde",3,4));
    c.añade(new Circulo("Rojo",5));
    c.imprimeTodo();
}
```

Color: Verde
Base: 3.0
Altura:4.0
Area: 6.0

Color: Rojo
Radio: 5.0
Area Total: 84.53981633974483

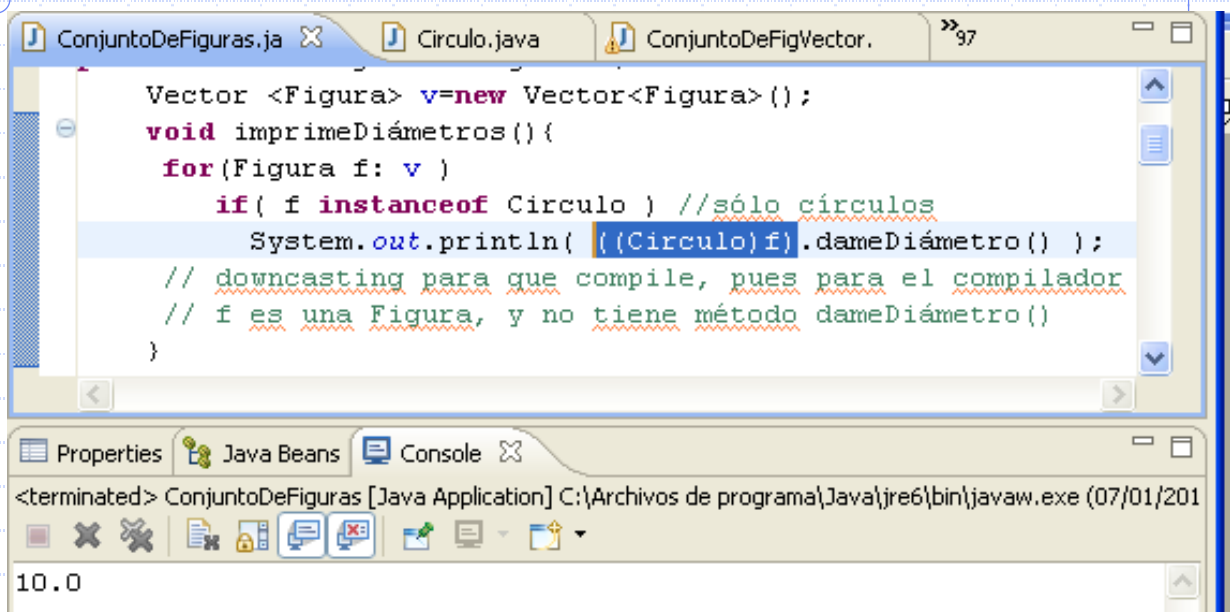
Ejercicio 3: métodos polimórficos

Añadir método double dameDiámetro() en clase Circulo.

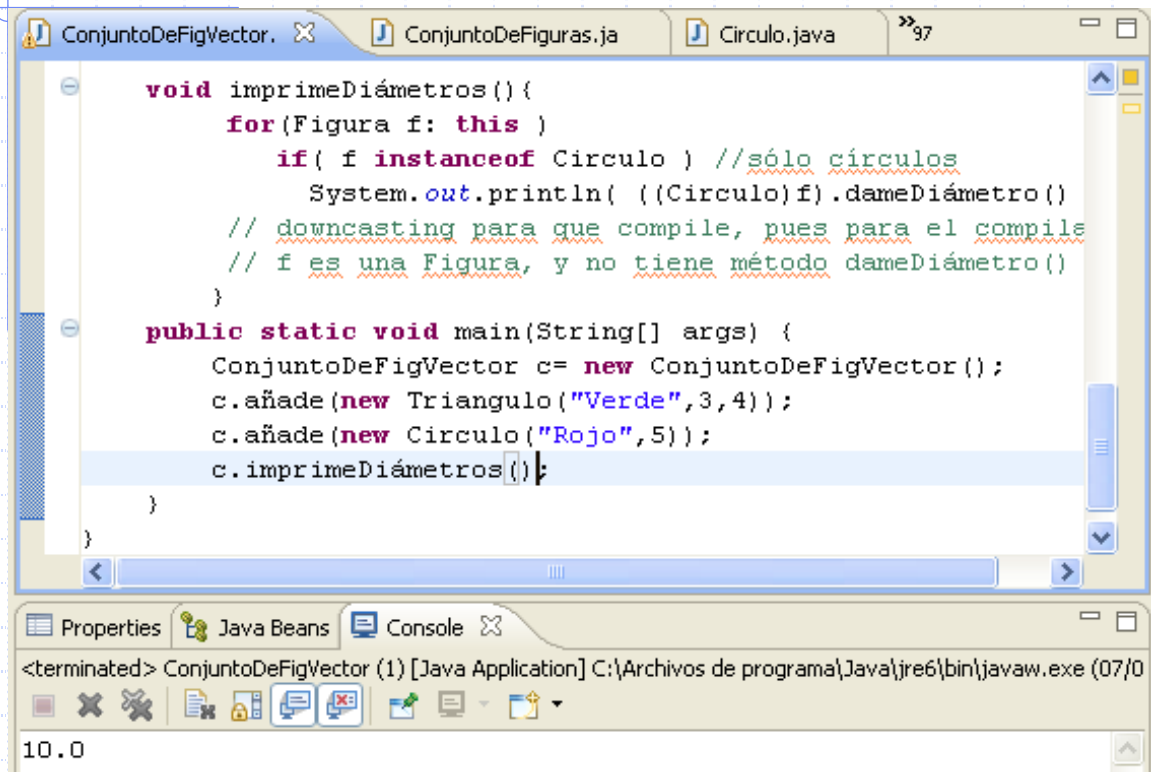
```
*Circulo.java X ConjuntoDeFigVector. ConjuntoDeFiguras.ja >>
package conjuntoFiguras.figuras;
public class Circulo extends Figura {
    double radio;
    public double dameDiámetro() { return 2*radio; }
```

Añadir un método polimórfico en ConjuntoDeFiguras que imprima los diámetros de los círculos

Ejercicio 3: métodos polimórficos



Ejercicio 3: siendo un Vector



```
void imprimeDiámetros() {
    for(Figura f: this)
        if( f instanceof Circulo ) //sólo círculos
            System.out.println( ((Circulo)f).dameDiámetro()
            // downcasting para que compile, pues para el compile
            // f es una Figura, y no tiene método dameDiámetro()
        }

    public static void main(String[] args) {
        ConjuntoDeFigVector c= new ConjuntoDeFigVector();
        c.añade(new Triangulo("Verde",3,4));
        c.añade(new Circulo("Rojo",5));
        c.imprimeDiámetros();
    }
}
```

Properties Java Beans Console

<terminated> ConjuntoDeFigVector (1) [Java Application] C:\Archivos de programa\Java\jre6\bin\javaw.exe (07/0

10.0

Ejercicio 4: museo

Se quiere elaborar el catálogo de un museo para lo cual se deben modelar las siguientes entidades:

- Los artistas, de los que se quiere guardar información acerca de su nombre y lugar de nacimiento, así como la fecha de su natalicio y de su fallecimiento.
- Las obras artísticas, de cada una de las cuales se desea conocer su título, su número de inventario, su autor y el año en que la realizó. En el museo solo hay dos tipos de obras: esculturas y pinturas. De las primeras se quiere guardar información sobre el material en que fueron hechas y la altura que poseen. De las pinturas en cambio, se guardará información de sus dimensiones y del soporte en que han sido realizadas.
- El catálogo es una secuencia de obras artísticas.

Ejercicio 4: museo

Se pide:

- Para cada una de las clases, definir su(s) constructora(s), métodos de acceso a cada uno de los atributos y un método muestra(), equals y toString().
- Definir métodos de forma que el catálogo del museo cuente con los siguientes servicios:

1. añadeObra(ob), que permite añadir una nueva obra artística al catálogo. Puede suponerse que la obra no es una falsificación y que, por tanto, no se encuentra en el museo. Se debe informar de si la operación ha tenido éxito o no.

2. eliminaObra(nI), que permite eliminar la obra artística dada por el número de inventario de su parámetro. Como antes, se informará de si se ha podido eliminar la obra o no.

3. buscaObra(nI), que busca la obra artística dada por el número de inventario de su parámetro.

Ejercicio 4: museo

Métodos polimórficos:

4- Define un método superficie() para la clase del catálogo del museo, que calcule la suma de las superficies de las pinturas que posee el museo.

5- Define un método masAlta() para la clase del catálogo del museo, que averigüe cuál es la escultura más alta que posee el museo y responda devolviendo su número de inventario.

Hacer pruebas.