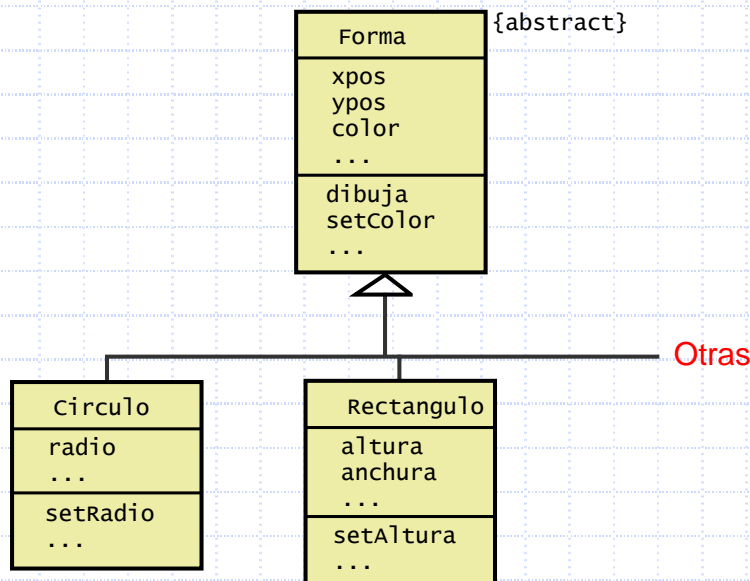


Clases abstractas e interfaces

Clases abstractas

- ◆ Clases cuya descripción es incompleta. Una clase abstracta declara métodos, pero no tiene que implementarlos.
 - No proporcionan la implementación de todos sus métodos
 - ◆ Los métodos no implementados se declaran como *abstract*
 - Una clase con un método abstracto debe declararse como clase abstracta
 - ◆ Pero una clase puede declararse como abstracta aunque no tenga ningún método abstracto

Clases abstractas



clases abstractas e Interfaces

3

Clases abstractas

```
public abstract class Forma {
    private int xpos, ypos;
    private Color color;
    // ...
    public abstract void dibuja();
    public void setColor(Color c){ /*...*/ };
}
```

los métodos
abstractos no
tienen
cuerpo

```
public class Circle extends Forma{
    private int radio;
    // ...
    public void dibuja(){ /*...*/ };
    public void setRadio(int){ /*...*/ };
}
```

dibuja un
círculo

```
public class Rectangle extends Forma{
    private int altura, anchura;
    // ...
    public void dibuja(){ /*...*/ };
    public void setAltura(int){ /*...*/ };
}
```

dibuja un
rectángulo

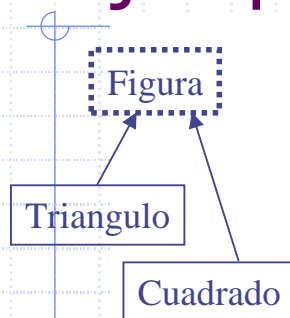
clases abstractas e Interfaces

4

Clases abstractas

- Las subclases de una clase abstracta deben:
 - ◆ Sobreescribir todos los métodos abstractos de la superclase, o bien
 - ◆ Ser declaradas como clases abstractas
- Una clase abstracta no puede instanciarse
 - ◆ No se pueden crear objetos de una clase abstracta
- Una clase abstracta puede incluir variables y métodos no abstractos.
- No se pueden definir constructores abstractos o métodos estáticos abstractos.

Ejemplo clase abstracta



```
public abstract class Figura {
    int x, y;
    public void mostrarOrigen() {
        System.out.println("x= "+x+" y= "+y);}
    public abstract double area(); // No tiene implementación
    public abstract double mostrarNombre();
}

public class Triangulo extends Figura {
    protected int base, altura;
    public Triangulo (int ba, int al) { base=ba; altura=al; }
    public double area() { return base*altura/2; }
    public void mostrarNombre() { System.out.println("triangulo"); }
}

public class Cuadrado extends Figura {
    protected int lado;
    public Cuadrado (int lado) { this.lado=lado; }
    public double area() { return lado*lado; }
    public void mostrarNombre() { System.out.println("cuadrado"); }}
```

Prueba clase abstracta

```
public class PruebaClaseAbstracta {
    public static void main(String args[]) {
        Figura fig;
        Triangulo tri;
        Cuadrado cua;

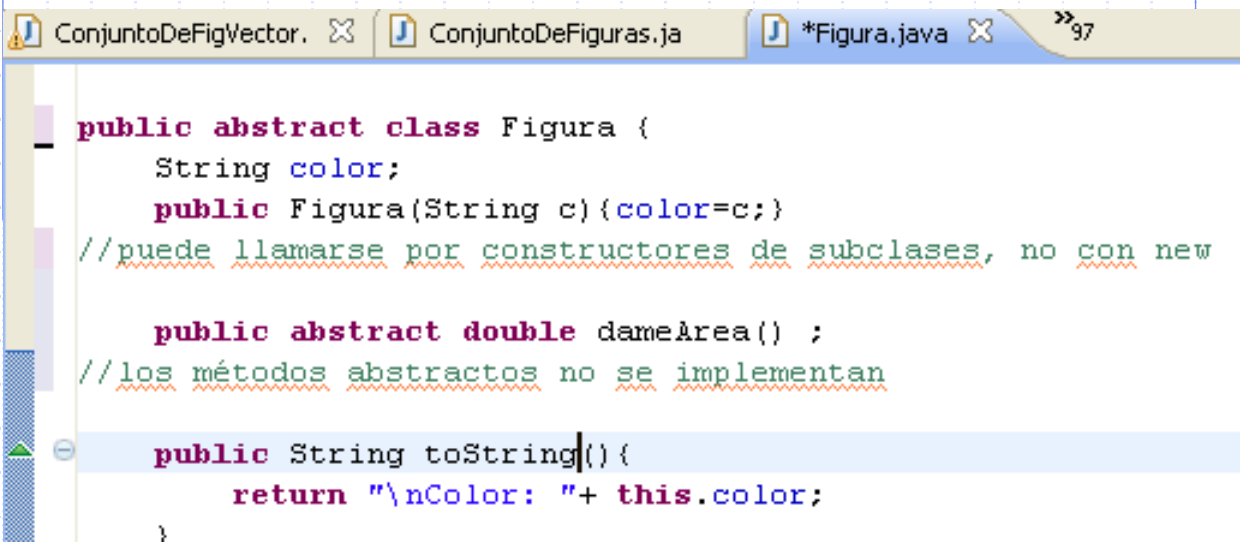
        fig = new Figura(); // error no se puede
                           //instanciar una clase abstracta
        tri = new Triangulo(4,3);
        tri.mostrarOrigen();
        tri.mostrarNombre();

        fig = tri;
        fig.mostrarNombre();
        System.out.println("Area triangulo: "+fig.area());

        cua = new Cuadrado(5);
        fig = cua;
        System.out.println("Area cuadrado: "+fig.area());
    }
}
```

Ejercicio 1

Declarar como abstracta la clase Figura del tema anterior



The screenshot shows an IDE with three tabs: 'ConjuntoDeFigVector', 'ConjuntoDeFiguras.java', and '*Figura.java'. The code in the active tab is as follows:

```
public abstract class Figura {
    String color;
    public Figura(String c){color=c;}
    //puede llamarse por constructores de subclases, no con new

    public abstract double dameArea() ;
    //los métodos abstractos no se implementan

    public String toString(){
        return "\nColor: "+ this.color;
    }
}
```

Ejercicio 1

Todas las Figuras deben tener método dameArea()

```
package conjuntoFiguras.figuras;

The type Circulo must implement the inherited abstract method Figura.dameArea()

double radio;

public Circulo(String c, double radio) {
    super(c);
    this.radio=radio;
}

/* public double dameArea() {
    return Math.PI*radio*radio;
} */

public String toString(){
    return super.toString()+"\nRadio: "+radio;
}
```

clases abstractas e Interfaces 9

Interfaces

- ◆ Sólo declaran comportamiento
 - Se utiliza la palabra clave *interface*
 - Por defecto todos sus métodos son públicos y abstractos
 - ◆ No implementan el comportamiento
 - Por defecto todos sus atributos son públicos, constantes y de clase
 - ◆ Por legibilidad normalmente los declaramos *static* y *final*

Interfaces

- ◆ Permite simular algunos aspectos de la herencia múltiple
 - Define un tipo de datos
 - Posibilita el enlace dinámico
- ◆ Otras clases pueden implementar un interfaz
 - Cualquier clase que implemente un interfaz debe definir todos los métodos de dicho interfaz
 - ◆ Debe proporcionar la implementación de dichos métodos
 - Si la clase no proporciona la implementación para todos los métodos del interfaz debe ser declarada como abstracta

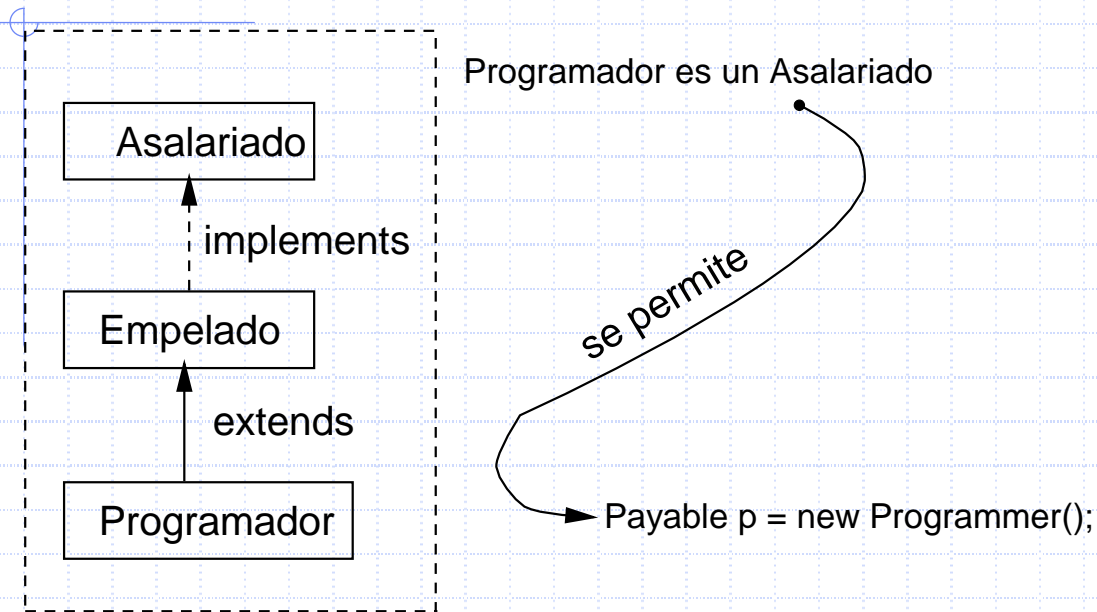
Declaración de interfaces

◆ Sintaxis

```
interface NombreInterfaz {  
    tipo static final NOMBRECONSTANTE1 = valor;  
    .....  
    public tipoDevuelto nombreMetodo1(listaParámetros);  
    .....  
}
```

```
class NombreClase implements NombreInterfaz1  
    [, NombreInterfaz2 ..] {  
    // declaración atributos y métodos de la clase  
    .....  
}
```

interfaces



Ejemplo de interfaz

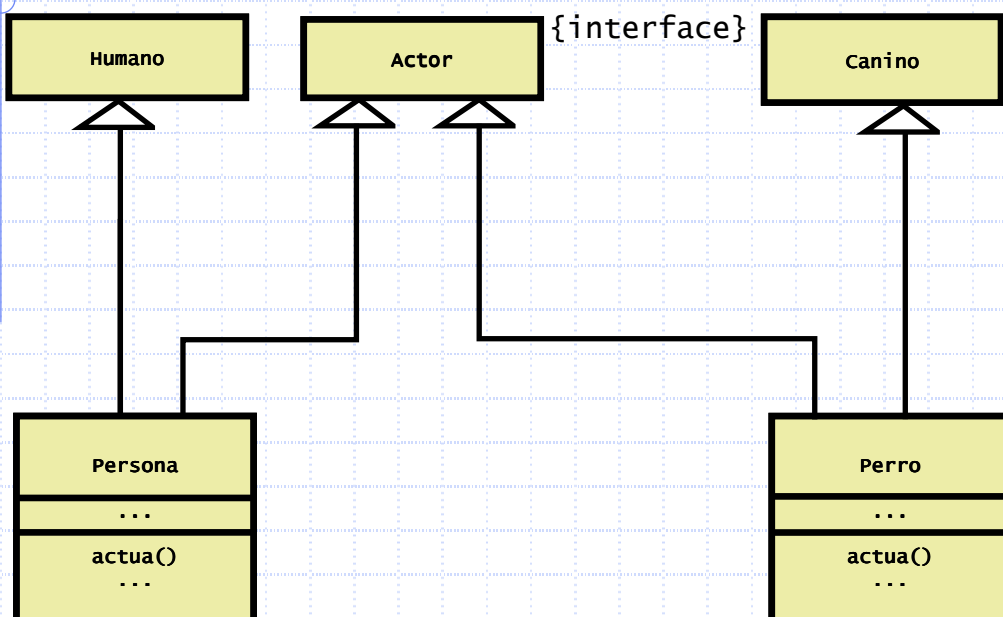
```
public interface Nombrable {
    static final boolean CIERTO = true;
    public void mostrarNombre();
}

public class Elemento implements Nombrable {
    String nombre;
    public Elemento(String nom) {
        nombre = nom;
    }
    // obligatorio implementar método mostrarNombre
    public void mostrarNombre(){
        System.out.println("Nombre: "+nombre);
        if (CIERTO)
            System.out.println("Constante CIERTO ");
    }
}
```

Uso del interfaz con enlace dinámico

```
public class PruebaInterfaz {  
    public static void main(String args[]) {  
        Elemento elem;  
        Nombrable inter;  
  
        elem = new Elemento("Luis");  
        elem.mostrarNombre();  
  
        // una referencia a interfaz puede  
        // utilizarse con una instancia de  
        // una clase que lo implemente  
        inter = elem;  
        inter.mostrarNombre();    }  
}
```

Ejemplo de interfaces



Ejemplo de interfaces

```
interface Actor
{
    void actua();
}
```

sin cuerpo

```
public class Persona extends Human implements
Actor {
    public void actua(){ /*...*/};
    //...
}
```

```
public class Perro extends Canino implements
Actor {
    public void actua(){ /*...*/};
    //...
}
```

clases abstractas e Interfaces

17

Extensión de interfaces

◆ Se puede definir un interface que especialice a otro interface mediante *extends*

- Es similar a la herencia de clases

clases abstractas e Interfaces

18

Extensión de interfaces

- ◆ No obstante un interface puede extender a varios interfaces a la vez
 - Aquí la herencia múltiple no plantea problemas porque no se hereda código

```
interface ElementoOrdenado extends
Comparable, Cloneable, java.io.Serializable {
// miembros y métodos propios del interfaz
//ElementoOrdenado
.....
}
```

Résumé de interfaces

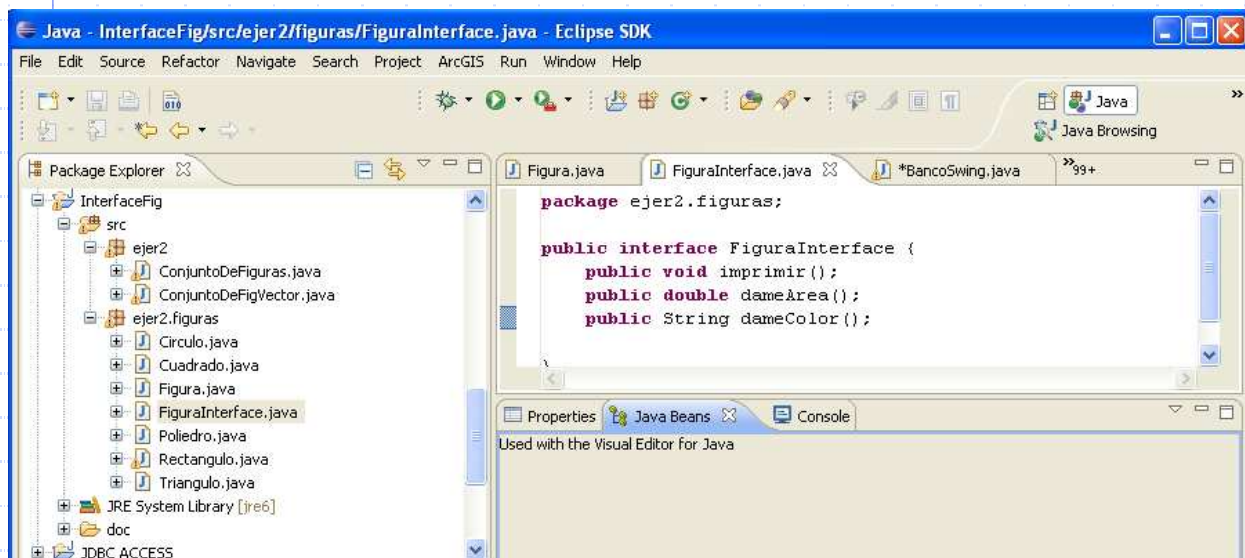
- ◆ *Las interfaces sirven para:*
 - *Declarar métodos que serán implementados por una o más clases.*
 - *Determinar la interface de programación de un objeto, sin mostrar el cuerpo de la clase.*
 - *Capturar similitudes entre clases no relacionadas, sin forzar una relación entre ellas.*
 - *Describir objetos "tipo-función", que podrán ser utilizados como argumentos al invocar métodos sobre objetos.*

Résumen de interfaces

Tipo	Class	Abstract Class	Interface
herencia	extends (simple)	extends (simple)	implements (multiple)
instanciable	yes	no	no
implementa	metodos	algún método	nada
datos	Se permiten	Se permiten	no se permiten

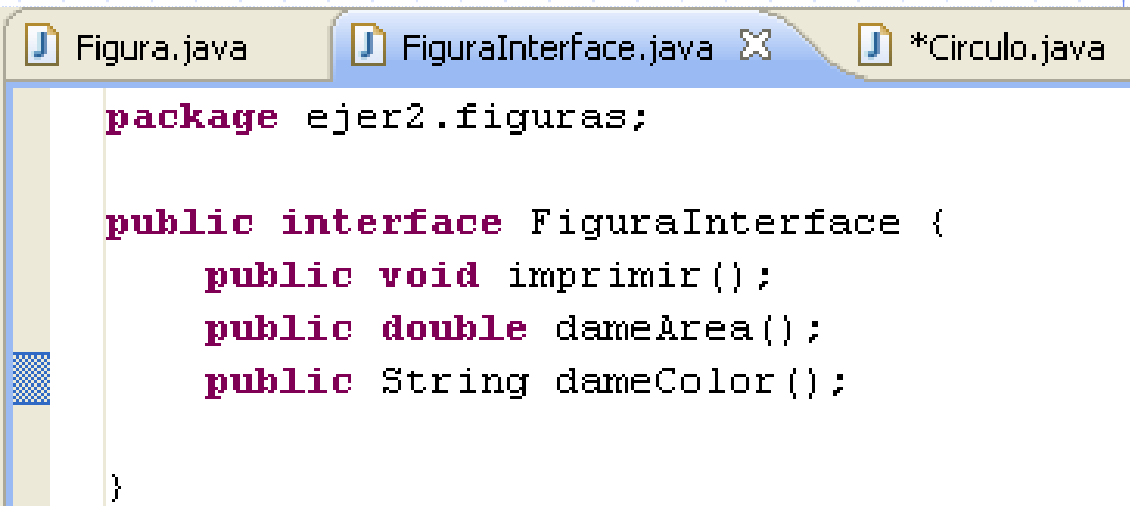
Ejercicio 2: Interface Figura

Definir el comportamiento de Figura del ejercicio anterior con Interface



Ejercicio 2: Interface Figura

Definición de la Interface Figura



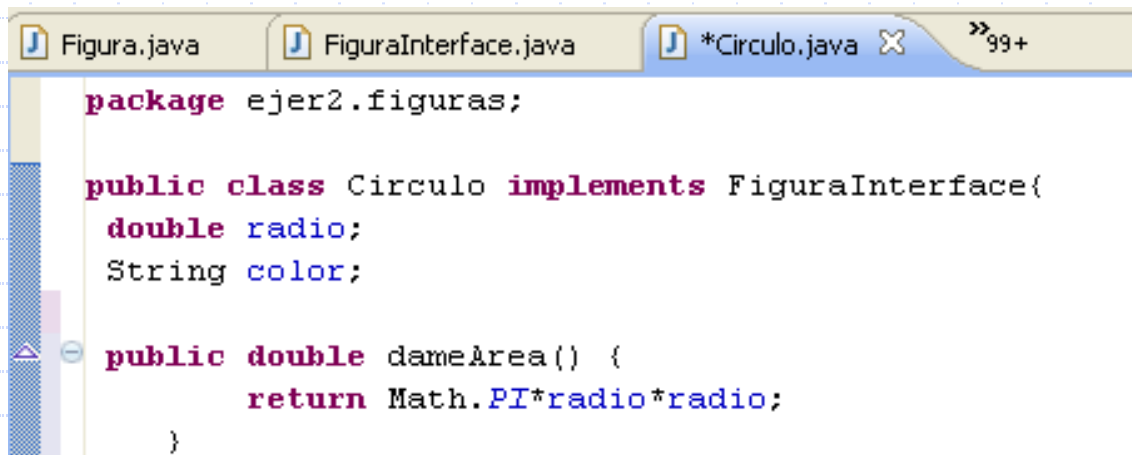
```
Figura.java  FiguraInterface.java  *Circulo.java

package ejer2.figuras;

public interface FiguraInterface {
    public void imprimir();
    public double dameArea();
    public String dameColor();
}
```

Ejercicio 2: Interface Figura

Utilizar la interface Figura para imponer comportamientos de todas las Figuras



```
Figura.java  FiguraInterface.java  *Circulo.java  >>99+

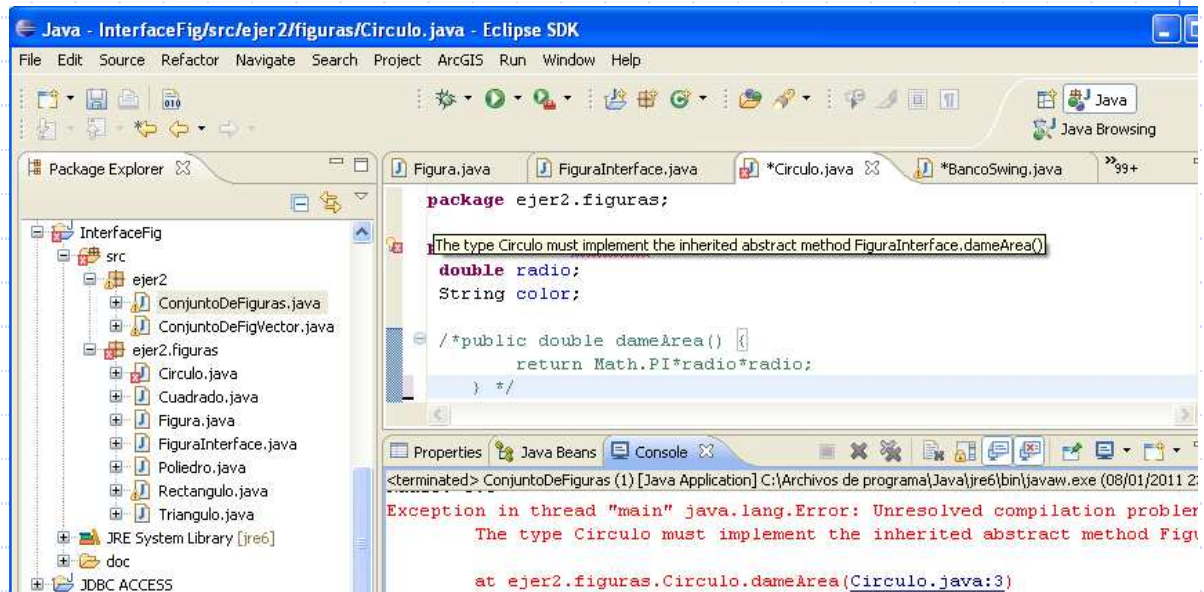
package ejer2.figuras;

public class Circulo implements FiguraInterface{
    double radio;
    String color;

    public double dameArea() {
        return Math.PI*radio*radio;
    }
}
```

Ejercicio 2: Interface Figura

Observar mensaje si no se implementa algún método de la interface Figura



clases abstractas e Interfaces

25

Ejercicio 2: Inmobiliaria

Se quiere modelar una agencia inmobiliaria para lo cual se deben tener en cuenta las siguientes entidades e informaciones asociadas:

a. Un inmueble viene dado por su ubicación y los metros cuadrados que ocupa. Hay dos tipos de inmuebles: superficies y construcciones. Los primeros tienen su precio por metro cuadrado. A su vez, hay dos tipos de superficies: solares y plazas de garaje. Los solares pueden estar en zona rústica o urbana. Las plazas de garaje pueden formar parte de un garaje público o de uno privado. Por su parte, las construcciones pueden ser nuevas o de segunda mano. Hay dos tipos de construcciones: viviendas y locales comerciales. Las viviendas tienen precio, número de habitaciones y piso. Los locales tienen precio por metro cuadrado.

clases abstractas e Interfaces

26

Ejercicio 2: Inmobiliaria

b. La agencia que queremos modelar se dedica a la venta y alquiler de inmuebles. Ahora bien, solamente alquila plazas de garaje y locales comerciales, mientras que solo vende solares y viviendas.

c. La agencia inmobiliaria viene dada por sendas secuencias de alquileres y ventas.

Se pide:

- Definir la jerarquía de clases de forma que se cumplan los requisitos anteriores.

- Para cada una de las clases, definir su(s) constructora(s), métodos de acceso a cada uno de los atributos y un método muestra() que escriba, de la manera que consideres más conveniente, la información de la entidad de que se trate.

- Definir un método público precio() que calcula el precio que tiene una superficie.

Ejercicio 2: Inmobiliaria

- Definir un método añadeVentaInmueble(in), que añade el inmueble dado por el parámetro a la secuencia de inmuebles en venta de la agencia, siempre que no estuviera ya antes en venta.

- Definir un método añadeAlquilerInmueble(in), que añade el inmueble dado por el parámetro a la secuencia de inmuebles en alquiler de la agencia, siempre que no estuviera ya antes en alquiler.

- Definir métodos que respondan a los siguientes servicios:

1. inmueblesVenta(p), que muestra los inmuebles con un precio de venta inferior al parámetro.

2. localesSegundaMano(m), que muestra los locales comerciales de segunda mano con una superficie superior al parámetro.

3. solaresRusticos(), que averigua cuántos solares no urbanos están en venta.

Ejercicio 2: Inmobiliaria

2. Define el método `equals()` y `toString()` para las clases `AgenciaInmobiliaria`, `Inmueble`, `Superficie` y `Solar`.

3. Dos agencias inmobiliarias han decidido fusionarse. Define un método `fusion(ag)` dentro de la clase de las agencias inmobiliarias, que permita crear una agencia nueva a partir de la agencia que invoca el método y la del parámetro dado.