Clases Abstractas e Interfaces

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2019-2020)

Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  - Emojis, `https://pixabay.com/images/id-2074153/`
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

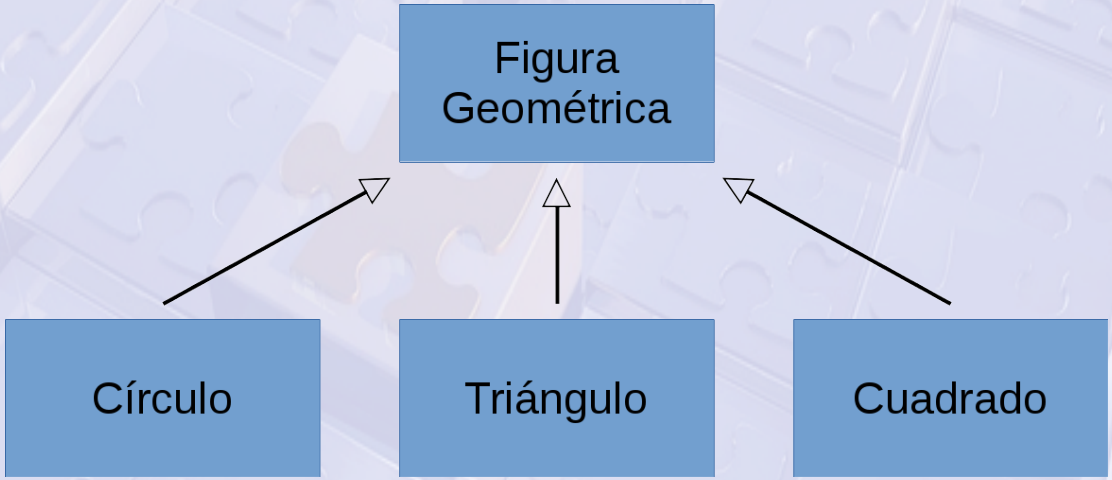
Objetivos

- Tanto para clases abstractas como para interfaces
  - Conocer los conceptos y su utilidad en el diseño
  - Saber reconocerlos en un diagrama de clases, así como sus relaciones con otros elementos del diagrama
  - Saber implementarlos en Java

Contenidos

Introducción a las clases abstractas

- Puede haber entidades a modelar de las que ...
  - Se sabe qué información contienen
  - Se sabe qué funcionalidad tienen
  - Pero no se sabe cómo realizan alguna de su funcionalidad
- Representan de manera genérica a otras entidades que sí concretan el funcionamiento desconocido



- Estas entidades se modelan mediante Clases abstractas

Clases abstractas

- Se declaran como tal y normalmente no proporcionan la implementación para alguno de sus métodos
  - Esos métodos sin implementación (solo cabecera) se denominan abstractos
- No es posible instanciar una clase abstracta
  - Pero sí declarar una variable usando una clase abstracta como tipo
- Son una **herramienta de diseño**:
  - *Obligan* a sus subclases a implementar una serie de métodos
    - Si no implementan algún método, también serán abstractas
  - Proporcionan métodos y atributos comunes a esas subclases
    - Sin perjuicio de que las subclases añadan atributos y/o métodos o redefinan métodos heredados
  - Definen un tipo de dato común a todas sus subclases
    - Facilita usar objetos de dichas subclases sin conocer ni consultar explícitamente a qué clase pertenecen

Ejemplo de uso de clase abstracta

- Se desea tener una colección de figuras geométricas y poder calcular la sumatoria de sus áreas

Java: Un uso práctico de clases abstractas

```
1 abstract class FiguraGeometrica {
2   public abstract float area();
3 }
4 class Triangulo extends FiguraGeometrica { . . . }  // Implementa area() adecuadamente
5 class Cuadrado extends FiguraGeometrica { . . . }   // Implementa area() adecuadamente
6
7 // En algún otro sitio ...
8 ArrayList<FiguraGeometrica> coleccionDeFiguras = new ArrayList<>();
9
10 // Se rellena la colección con figuras de todo tipo y sin un orden concreto
11 coleccionDeFiguras.add (new Triangulo (lado1, lado2, lado3));
12 coleccionDeFiguras.add (new Cuadrado (lado1, lado2));
13
14 float suma = 0.0f;
15 for (FiguraGeometrica unaFigura : coleccionDeFiguras) {
16   // No es necesario conocer de qué clase se instanció
17   // el objeto concreto que en cada momento está referenciado por unaFigura
18   suma += unaFigura.area();
19 }
```

Clases Abstractas en Java y Ruby

- Java
  - Se usa la palabra reservada `abstract` para indicar que una clase y/o método son abstractos
  - Permite clases abstractas sin métodos abstractos
- Ruby
  - Ruby no soporta las clases abstractas
    - ★ No incorpora ningún mecanismo de comprobación por adelantado que el uso de una variable se ajusta a lo especificado en una clase

      ★ ¿Cómo se implementaría el ejemplo de las figuras geométricas?

El ejemplo de las figuras geométricas en Ruby

Ruby: El ejemplo de las figuras geométricas

```ruby
class Triangulo
  . . .
  def area
    ...
  end
end

class Cuadrado
  . . .
  def area
    ...
  end
end

coleccionDeFiguras = []
coleccionDeFiguras << Triangulo.new(lado1, lado2, lado3)
coleccionDeFiguras << Cuadrado.new(lado1, lado2)

suma = 0.0
for figura in coleccionDeFiguras do
  suma += figura.area()
end
```

- No ha sido necesario disponer de una clase `FiguraGeometrica`

Clases no instanciables en Ruby

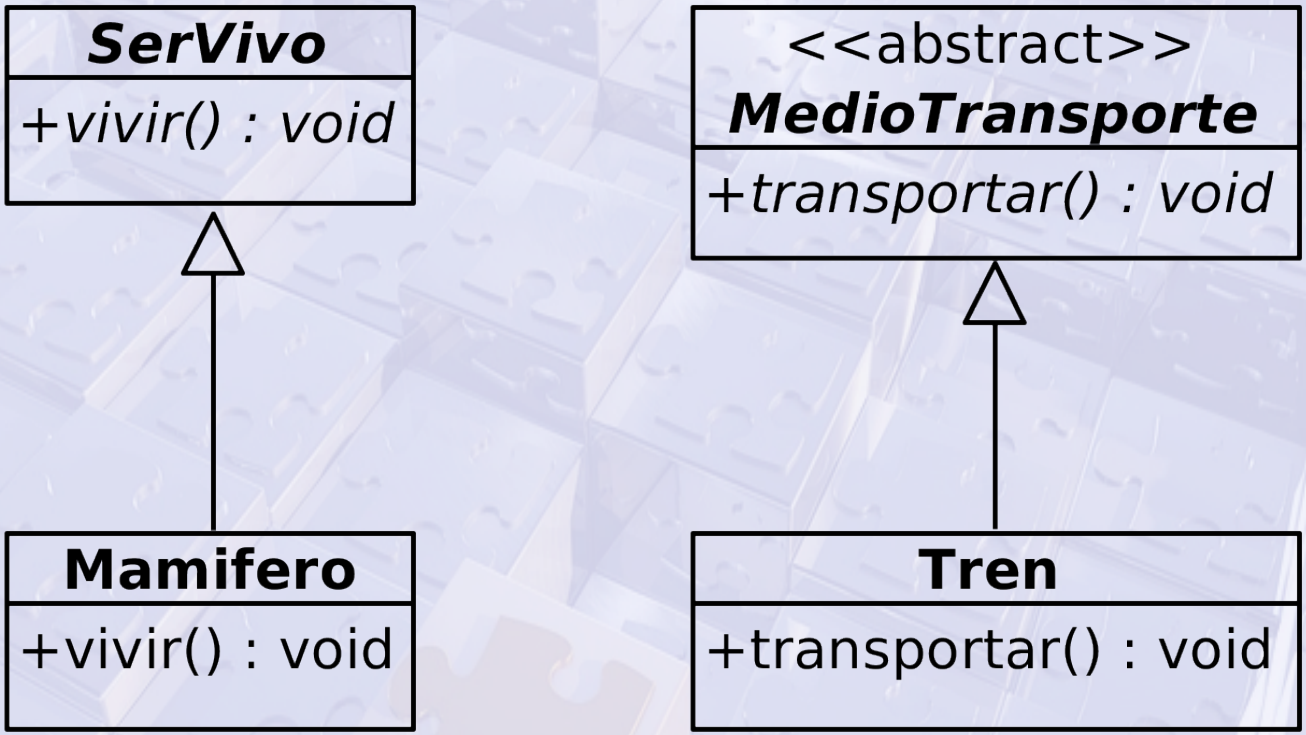
- Supuesto práctico:
  - Se necesita una clase, en Ruby, que aglutine atributos y/o métodos comunes de sus clases derivadas
  - Esa clase no podrá instanciarse
  - Sus clases derivadas sí podrán instanciarse
- Solución:
  - Se hace privado el método `new` en la clase padre
  - Se vuelve a hacer público el método `new` en las clases derivadas

Ruby: Clases no instanciables

```ruby
class FiguraGeometrica
  . . .   # Atributos y métodos comunes
  private_class_method :new
end

class Cuadrado < FiguraGeometrica
  public_class_method :new
  . . . # Atributos y métodos específicos de esta subclase
end
```

Representación UML de las clases abstractas



- El nombre de la clase abstracta y de los métodos abstractos se escribe en *cursiva*
  - ¡Cuidado! A veces se os pasa inadvertido en algún examen

Ejemplo

Java: Ejemplo de clase y método abstracto

```
1  abstract class SerVivo {
2    String planeta;
3    SerVivo (String p) {
4      planeta = p;
5    }
6    public String existir() {return "Existiendo";}
7    public abstract String vivir();
8  }
9
10 class Humano extends SerVivo {
11     String nombre;
12     Humano (String p, String n) {
13         super (p);
14         nombre = n;
15     }
16     // "Obligatorio"  Si no se redefine, esta clase también será abstracta
17     @Override
18     public String vivir() {
19       return "Viviendo como humano";
20     }
21
22     // No obligatorio
23     @Override
24     public String existir() {
25         return super.existir() + " como humano";}
26     }
27 }
```

Interfaces

- Una interfaz define un determinado protocolo de comportamiento (T. Budd p.88) y permite reutilizar la especificación de dicho comportamiento
- Será una clase la que lo implemente (**realización**)
- Una interfaz define un contrato que cumplen las clases que realizan dicha interfaz
- Cada interfaz define un tipo
  - Se pueden declarar variables de ese tipo
  - Dichas variables podrán referenciar instancias de clases que realicen dicha interfaz

Interfaces en Java

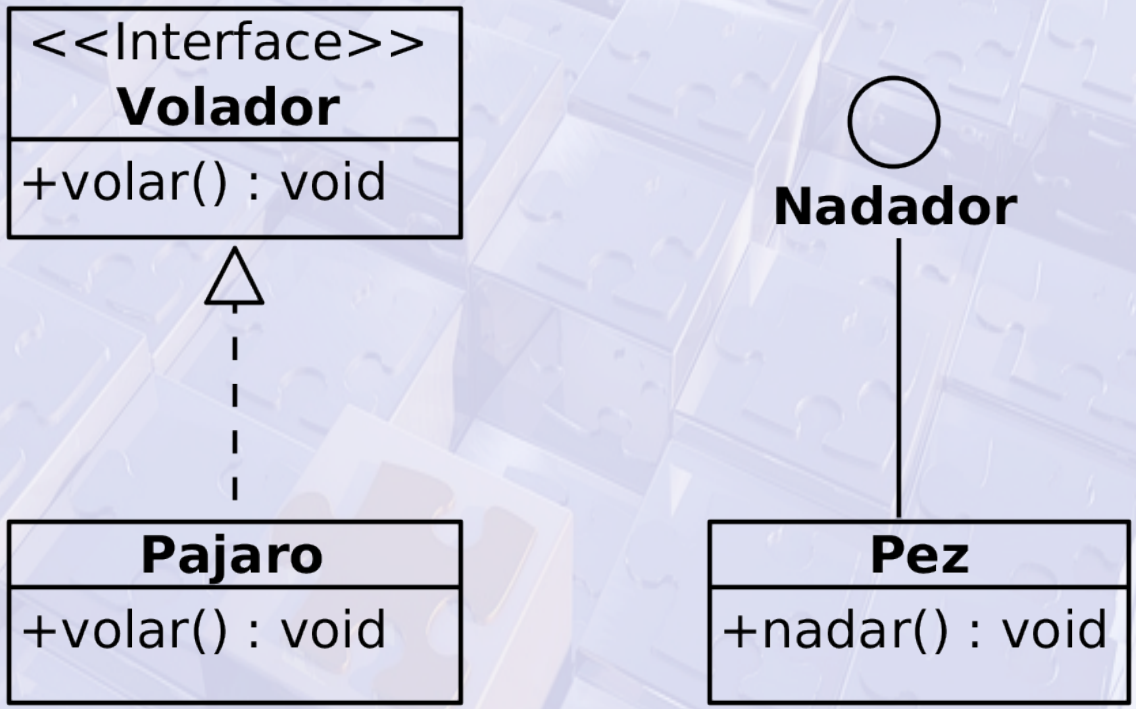
- Una clase puede realizar varias interfaces
- Una interfaz puede heredar de una o más interfaces
- Una interfaz solo puede tener:
  - Constantes
  - Signaturas de métodos
  - Métodos tipo `default`
  - El equivalente a los métodos de clase `static`
- Solo los métodos tipo `default` y `static` pueden tener asociada implementación
- Los métodos son `public` y las constantes `public`, `static`, y `final`
- No pueden ser instanciadas, solo realizadas por clases o extendidas por otras interfaces

Interfaces en Java

- Se pueden redefinir los métodos `default` en interfaces que heredan y en clases que realizan esa interfaz
- Una clase puede heredar de una clase y además realizar varias interfaces
- Una clase abstracta puede indicar que realiza una interfaz sin implementar alguno de sus métodos.
  - *Fuerza* a hacerlo a sus descendientes no abstractos
- Una clase puramente abstracta se parece a una interfaz pero Java no permite herencia múltiple

- **Ruby** no soporta de forma nativa el concepto de interfaz

Representación UML de las interfaces



Ejemplo

Java: Ejemplo de interfaces

```
1  interface Interfaz1 {
2      int CONSTANTE = 33;
3
4      String hazAlgo1 ();
5      default String hazAlgo12 () {return "1";}
6      default String hazAlgo11 () {return "11";}
7  }
8
9  interface Interfaz2 {
10     String hazAlgo2 ();
11     default String hazAlgo12() {return "2";}
12 }
13
14 class Test implements Interfaz1, Interfaz2 {
15     @Override
16     public String hazAlgo1() {return "algo1";}
17
18     @Override
19     public String hazAlgo2() {return "algo2";}
20
21     @Override
22     public String hazAlgo12() { // Por colisión debe redefinir
23         String a=Interfaz1.super.hazAlgo12();
24         String b=Interfaz2.super.hazAlgo12();
25         return (a+" "+b+" "+Integer.toString(Interfaz1.CONSTANTE));
26     }
27 }
```

Clases abstractas

- Recurso muy utilizado para *aglutinar clases similares*
  - Obviamente, debe tener sentido en el contexto del modelo
- Unido a otro mecanismo, polimorfismo, (que veremos más adelante) permite usar esas clases de una manera muy limpia
  - Recordar el ejemplo de las figuras geométricas
  - Se puede codificar sin tener que consultar explícitamente a qué clase concreta pertenece cada objeto
  - Permite que el sistema sea fácilmente extensible con relativamente poco trabajo
    - ★ Por ejemplo, añadir nuevas clases que deriven de la clase abstracta

Interfaces

→ *Diseño* ←

- Muy usados para independizar:
  - El “qué” (cabeceras de métodos).
    En la interfaz
  - El “cómo” (implementación de los mismos).
    En las clases que lo realizan
- Permitiendo tener varias implementaciones
- Ejemplo
  - Imaginad una interfaz Lista que declara todo lo que se puede hacer con una lista
  - Se pueden tener varias clases que realizan esa interfaz implementando la lista con arrays, con punteros, doblemente enlazada, etc.
  - En una aplicación que use listas a través de la interfaz se puede cambiar de una implementación a otra sin apenas modificar nada en la aplicación

Clases Abstractas e Interfaces

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2019-2020)