

Sesión 16: Genéricos

Programación 2

5. Herencia y polimorfismo

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos
- 👍 Tema 3: Programación Modular
- 🕒 Tema 5: Herencia y Polimorfismo
 - **Profundizamos** en la herencia

En el capítulo de hoy



Tema 5: Herencia y Polimorfismo

- **Subtipado:** conversión, LSP¹, interfaz (herencia)
- **Ad hoc:** mencionaremos sobrecarga
- **Genéricos:** polimorfismo paramétrico



Tema 7: Implementación de TADs lineales

¹*Liskov Substitution Principle*



Herencia

- Las instancias de una subclase heredan todas las *propiedades* (atributos y métodos) de la superclase
- En Java el *enlazado* de los métodos se realiza en **tiempo de ejecución**²
- En las subclases se pueden **sobreescribir** las propiedades



“Un gran poder...”

²*Dynamic Dispatching* o *Dynamic binding* vs. *Static binding*

Conversiones de tipo

Java nos ayuda con la herencia i

```
public class Mamifero extends Animal {...}
```

```
public static Mamifero f() {...}
```

```
Animal x = f();
```

Java nos ayuda con la herencia i

```
public class Mamifero extends Animal {...}
```

```
public static Mamifero f() {...}
```

```
Animal x = f();
```

Java compila 

Java nos ayuda con la herencia ii

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {  
    return new Mamifero();  
}
```

```
Mamifero x = f();
```


Java nos ayuda con la herencia ii

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {  
    return new Mamifero();  
}
```

```
Mamifero x = f();
```

Java no compila 👍

Conversión de tipos

- ¡El código estaba bien! ¿Por qué no compila?
- Podemos *forzar* al compilador a que lo entienda con una conversión de tipos³

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {  
    return new Mamifero();  
}
```

```
Mamifero x = (Mamifero) f();
```

³*Type casting, Type coercion*

Downcasting i

```
public class Mamifero extends Mamifero {...}
```

```
public class Pez extends Animal {...}
```

```
public static Animal f() { return new Pez();}
```

```
Mamifero x = (Mamifero) f();
```

Downcasting i

```
public class Mamifero extends Mamifero {...}
```

```
public class Pez extends Animal {...}
```

```
public static Animal f() { return new Pez();}
```

```
Mamifero x = (Mamifero) f();
```

Java compila y se rompe en ejecución



 ¡Probadlo!

Downcasting ii

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {  
    return new Mamifero();  
}
```

```
Mamifero x = (Mamifero) f();
```

Downcasting ii

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {  
    return new Mamifero();  
}
```

```
Mamifero x = (Mamifero) f();
```

- Pero como yo sé que no se rompe, se lo quiero decir al compilador mediante un *downcasting*
- !Esto no es una buena idea!

instanceof i

```
public class Mamifero extends Animal {...}
```

```
public static Animal f() {...}
```

```
Mamifero x = null;  
if (f() instanceof Mamifero) {  
    x = (Mamifero)f();  
}
```

instanceof ii

- Y ya no se rompe aunque f() devuelva un pez

```
public class Mamifero extends Animal {...}
```

```
public class Pez extends Animal {...}
```

```
public static Animal f() { return new Pez();}
```

```
Mamifero x = null;
```

```
Animal a = f();
```

```
if (a instanceof Mamifero)
```

```
    x = (Mamifero) a;
```

```
assert x == null;
```


instanceof iii

```
/** Método que lee una figura */  
public Figura leerFigura() {...}  


---

/** Imprimir la longitud del lado */  
Figura fig = leerFigura();  
if (fig instanceof PoligonoRegular) {  
    System.out.println(  
        "Lado: "  
        +                fig .lado()  
    );  
}
```

instanceof iii

```
/** Método que lee una figura */  
public Figura leerFigura() {...}  


---

/** Imprimir la longitud del lado */  
Figura fig = leerFigura();  
if (fig instanceof PoligonoRegular) {  
    System.out.println(  
        "Lado: "  
        + ((PoligonoRegular)fig).lado()  
    );  
}
```

Herencia de interfaz

Herencia de **interfaz**

- Muchos lenguajes permite describir **interfaces**⁴
- Un interfaz declara un conjunto de **métodos** (API) que luego las clases tendrán que **implementar**
- **Nos va a recordar** a las clases abstractas
- Pero se usan más y con **más riqueza** que las clases abstractas: **herencia múltiple**

⁴También llamados **traits** en algunos lenguajes

CRUD: un interfaz de persistencia

```
public interface CRUD {  
    boolean create(Data d);  
    Data read(Id id);  
    boolean update(Data d);  
    boolean delete(Id id);  
}
```

- Es *como* una clase abstracta donde *todos los métodos son abstractos*
- No hay atributos, ni constructores, sólo *interfaz*
- *Todo es público*

¿Para qué sirve? i

```
public class Controller {  
    private CRUD storage;  
    public Controller (CRUD storage) {  
        this.storage = storage;  
    }  
    public onSave(Data d) {  
        if (storage.update(d))  
            notifySaveDone();  
        else  
            notifySaveFailed();  
    }  
    ...  
}
```

¿Para qué sirve? i

```
public class Controller {  
    private CRUD storage;  
    public Controller (CRUD storage) {  
        this.storage = storage;  
    }  
    public onSave(Data d) {  
        if (storage.update(d))  
            notifySaveDone();  
        else  
            notifySaveFailed();  
    }  
    ...  
}
```

- Se pueden usar como si fueran un tipo (como las clases)
- ¿Podemos programar sin conocer la implementación?

¿Para qué sirve? i Más ocultación

```
public class Controller {  
    private CRUD storage;  
    public Controller (CRUD storage) {  
        this.storage = storage;  
    }  
    public onSave(Data d) {  
        if (storage.update(d))  
            notifySaveDone();  
        else  
            notifySaveFailed();  
    }  
    ...  
}
```

- Se pueden usar como si fueran un tipo (como las clases)
- ¿Podemos programar sin conocer la implementación?

Dos implementaciones

```
public class DB
    implements CRUD {
    public DB(String host) {
        ...
    }
    public boolean create(Data d) {
        ...
    }
    public Data read(Id id) {
        ...
    }
    boolean update(Data d) {
        ...
    }
    boolean delete(Id id) {
        ...
    }
}
```

```
public class RestClient
    implements CRUD {
    public RestClient(String endpoint) {
        ...
    }
    public boolean create(Data d) {
        ...
    }
    public Data read(Id id) {
        ...
    }
    boolean update(Data d) {
        ...
    }
    boolean delete(Id id) {
        ...
    }
}
```

¿Para qué sirve? ii

- Hoy

```
public class App {  
    public static void main(String[] args) {  
        CRUD storage = new DB("postgresql://localhost:5432");  
        Controller saveController = new Controller(storage);  
    }  
}
```

¿Para qué sirve? ii

- Hoy

```
public class App {  
    public static void main(String[] args) {  
        CRUD storage = new DB("postgresql://localhost:5432");  
        Controller saveController = new Controller(storage);  
    }  
}
```

- Mañana

```
public class App {  
    public static void main(String[] args) {  
        CRUD storage = new RestClient("http://localhost:8080");  
        Controller saveController = new Controller(storage);  
    }  
}
```

¿Para qué sirve? ii Más reusabilidad

- Hoy

```
public class App {  
    public static void main(String[] args) {  
        CRUD storage = new DB("postgresql://localhost:5432");  
        Controller saveController = new Controller(storage);  
    }  
}
```

- Mañana

```
public class App {  
    public static void main(String[] args) {  
        CRUD storage = new RestClient("http://localhost:8080");  
        Controller saveController = new Controller(storage);  
    }  
}
```

Herencia múltiple

```
public class Controller
    implements InputViewListener
               SaveViewListener {
    private Data data;
    public onChanged(Data d) {
        data = d;
    }
    public onSave() {
        storage.update(data);
    }
}
```

- Java no admite herencia múltiple entre clases (**extends**)
- Pero admite herencia múltiple de interfaces (**implements**)
- Masivamente usado para arquitecturar aplicaciones

LSP: “Un gran poder...”

LSP: *Liskov substitution principle*

Let $\phi(x)$ be a property provable about objects x of type T . Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T

Barbara Liskov (1987) and Jeannette Wing, 1994

LSP: Principio de Substitución de Liskov

Sea $\phi(x)$ una propiedad demostrable de los objetos x de tipo T . Entonces $\phi(y)$ debe cumplirse para los objetos y de tipo S si S sea subtipo de T

Barbara Liskov (1987) and Jeannette Wing, 1994

LSP: una buena práctica

Sea $\phi(x)$ una propiedad demostrable de los objetos x de tipo T . Entonces $\phi(y)$ debe cumplirse para los objetos y de tipo S si S sea subtipo de T

- Los compiladores de los lenguajes nos protegen un poco (ej. *downcasting*)
- Pero como nos permiten reemplazar los métodos en las subclases **podríamos destruir** las propiedades conceptuales de las superclases



Ej. en Cuadrado podríamos cambiar `perimetro()` para que calcule el área (iríamos **en contra del principio**)

Últimas palabras sobre la herencia

- Herencia múltiple: difícil, en Java sólo de interfaz
- *Dynamic vs Static dispatching (binding)*
- *American vs. Scandinavian schools*
- Principios **SOLID**⁵:
 - **SRP**: Single-responsibility
 - **OCP**: Open-closed (extension-modification)
 - **LSP**: Liskov substitution[8]
 - **ISP**: Interface segregation (many)
 - **DIP**: Dependency inversion

⁵Para cuando seamos mayores, pero no lo olvidéis

Polimorfismo Ad-hoc

Polimorfismo: varios tipos

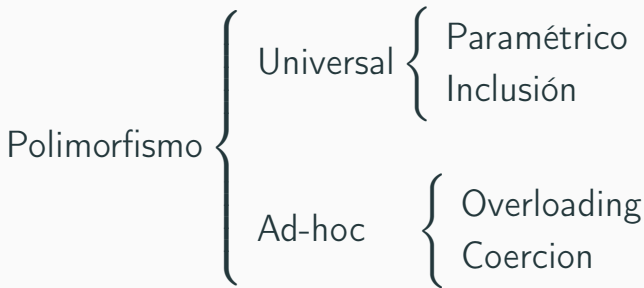
*On Understanding Types, Data Abstraction,
and Polymorphism*

Luca Cardelli and Peter Wegner, 1985

Polimorfismo: varios tipos

*On Understanding Types, Data Abstraction,
and Polymorphism*

Luca Cardelli and Peter Wegner, 1985



Polimorfismo: varios tipos

*On Understanding Types, Data Abstraction,
and Polymorphism*

Luca Cardelli and Peter Wegner, 1985



Ejemplos de poliforfismo Ad-hoc

- Algunos operadores como por ejemplo +:

`27 + 42 // int x int`

`"Hola" + " mundo" // String x String`

`"Lado: " + 5 // String x int`

- Métodos multi-parámetros:

`out.format("Hola %s\n", nombre);`

`out.format("Lado %f, perímetro: %f\n", l, 4*l);`

- Conversión automática de tipos:

`27 + 42.0 // 27 se convierte a float`

Genéricos: polimorfismo paramétrico

Tuplas

- Pero... ¿tuplas de qué?

⁶Vale $\text{Integer} \times \text{Integer}$

Tuplas

- Pero... ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}^6$

⁶Vale Integer \times Integer

Tuplas

- Pero... ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}$ ⁶

```
public class TuplaInt {  
    private Integer x;  
    private Integer y;
```

```
    public TuplaInt(Integer fst,  
                    Integer snd) {  
        x = fst;  
        y = snd;  
    }  
}
```

```
    public Integer fst() {  
        return x;  
    }
```

```
    public Integer snd() {  
        return y;  
    }  
}
```

⁶Vale $\text{Integer} \times \text{Integer}$

Tuplas de booleanos

```
public class TuplaBool {  
    private Boolean x;  
    private Boolean y;  
  
    public TuplaBool(Boolean fst,  
                    Boolean snd) {  
        x = fst;  
        y = snd;  
    }  
}
```

```
public Boolean fst() {  
    return x;  
}  
  
public Boolean snd() {  
    return y;  
}  
}
```

Tuplas de strings

```
public class TuplaString {  
    private String x;  
    private String y;  
  
    public TuplaString(String fst,  
                        String snd) {  
        x = fst;  
        y = snd;  
    }  
}
```

```
    public String fst() {  
        return x;  
    }  
  
    public String snd() {  
        return y;  
    }  
}
```

Tuplas de strings por booleanos

```
public class TuplaStringBool {  
    private String x;  
    private Boolean y;
```

```
    public TuplaStringBool(String fst,  
                           Boolean snd) {  
        x = fst;  
        y = snd;  
    }  
}
```

```
    public String fst() {  
        return x;  
    }
```

```
    public Boolean snd() {  
        return y;  
    }  
}
```

Tuplas de booleanos por strings

```
public class TuplaBoolString {  
    private Boolean x;  
    private String y;  
  
    public TuplaBoolString(Boolean fst,  
                             String snd) {  
        x = fst;  
        y = snd;  
    }  
  
    public Boolean fst() {  
        return x;  
    }  
  
    public String snd() {  
        return y;  
    }  
}
```

Herranz, ¡ya lo he entendido!

Herranz, ¡ya lo he entendido!

```
public class Tupla<T1, T2> {  
    private T1 x;  
    private T2 y;  
  
    public Tupla(T1 fst,  
                 T2 snd) {  
        x = fst;  
        y = snd;  
    }  
  
    public T1 fst() {  
        return x;  
    }  
  
    public T2 snd() {  
        return y;  
    }  
}
```

Vale interpretarlo como una **plantilla**
en la que substituir **T1** y **T2**

Nueva clase: `Tupla<Integer,Integer>`

```
public class Tupla<Integer, Integer> {  
    private Integer x;  
    private Integer y;  
  
    public Integer fst() {  
        return x;  
    }  
  
    public Integer snd() {  
        return y;  
    }  
  
    public Tupla(Integer fst,  
                  Integer snd) {  
        x = fst;  
        y = snd;  
    }  
}
```

Nueva clase: `Tupla<Boolean,String>`

```
public class Tupla<Boolean, String> {  
    private Boolean x;  
    private String y;  
  
    public Tupla(Boolean fst,  
                  String snd) {  
        x = fst;  
        y = snd;  
    }  
  
    public Boolean fst() {  
        return x;  
    }  
  
    public String snd() {  
        return y;  
    }  
}
```



En práctica (java -ea ...)

```
public class PruebaTuplas {  
    public static void main(String[] args) {  
        Tupla<Integer,Integer> t1 =  
            new Tupla<Integer, Integer>(5,1);  
        Tupla<Boolean,Boolean> t2 =  
            new Tupla<Integer, Integer>(true, false);  
        Tupla<String,String> t3 =  
            new Tupla<String, String>("Ángel","Herranz");  
        Tupla<String,Boolean> t4 =  
            new Tupla<String, String>("Ángel",true);  
        assert t1.snd().equals(1);  
        assert t2.fst();  
        assert t3.snd().equals("Herranz");  
        assert !t4.snd();  
    }  
}
```

¿Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

¿Qué es esto!?

```
public class Nodo<T> {  
    public T dato;  
    public Nodo<T> siguiente;  
  
    public Nodo(T dato) {  
        this.dato = dato;  
        siguiente = null;  
    }  
}
```

 Tema 7: Implementación de TADs lineales



Implementar y ¡a dibujar!

```
public class PruebaNodo {  
    public static void main(String[] args) {  
        Nodo<Integer> uno = new Nodo<Integer>(1);  
        Nodo<Integer> dos = new Nodo<Integer>(2);  
        uno.siguiente = dos;  
        Nodo<String> one = new Nodo<String>("one");  
        one.siguiente = dos;  
        Nodo<Integer> primero = null;  
        for (int i = 0; i < 1000000; i++) {  
            Nodo<Integer> segundo = primero;  
            primero = new Nodo<Integer>(i);  
            primero.siguiente = segundo;  
        }  
    }  
}
```