Sesión 16: Genéricos

Programación 2

Ángel Herranz

Abril 2019

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
  - Herencia $\Rightarrow$ Subtipado
- 🕒 Tema 7: Implementación de TADs lineales

En el capítulo de hoy

👍 Tema 5: Herencia y Polimorfismo

[1] *Automatic casting*

En el capítulo de hoy

- 👍 Tema 5: Herencia y Polimorfismo
  - **Subtipado**: interfaz (aún herencia)

[1] *Automatic casting*

En el capítulo de hoy

- 👍 Tema 5: Herencia y Polimorfismo
  - **Subtipado**: interfaz (aún herencia)
  - ***Ad hoc***: sobrecarga y conversión automática[1]

[1] *Automatic casting*

En el capítulo de hoy

- 👍 Tema 5: Herencia y Polimorfismo
  - **Subtipado**: interfaz (aún herencia)
  - ***Ad hoc***: sobrecarga y conversión automática[1]
  - **Genéricos**: polimorfismo paramétrico

[1] *Automatic casting*

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*

LSP

Principio de Substitución de Liskov[3]

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 and Jeannette Wing

[3] **LSP**: *Liskov substitution principle*

Ejemplo de aplicación de LSP i

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public S f() {...}
```

```
/* Permitido por LSP */
T x = f();
```

Ejemplo de aplicación de LSP i

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public S f() {...}
```

```
/* Permitido por LSP */
T x = f();
```

Java compila 👍

Ejemplo de aplicación de LSP ii

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public T f() { return new S();}
```

```
/* No permitido por LSP */
S x = f();
```

Ejemplo de aplicación de LSP ii

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public T f() { return new S();}
```

```
/* No permitido por LSP */
S x = f();
```

Java no compila 👍

Downcasting i

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public T f() { return new S();}
```

```
S x = (S) f();
```

Downcasting i

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public T f() { return new S();}
```

```
S x = (S) f();
```

Java compila 👎

Downcasting ii

- Supóngase que e es una expresión de tipo T

$$e : T$$

- sabemos que es una instancia de la clase S
- y sabemos que S es una subclase de T

$$S <: T$$

- en Java podemos convertir (*casting*) e al tipo S:

$$(S)\, e$$

- Porque

$$(S)e : S$$

instanceOf i

```
/** S es subclase de T */
public class S extends T {...}
```

```
/** Método que devuelve una instancia de S */
public T f() { return new S();}
```

```
S x;
if (f() instanceOf S) {
  x = (S)f();
}
```

instanceOf ii

```
/** 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 ii

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

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

instanceOf iii

```
public B extends A {}
```

```
public class C {
  public static void main(String[] args) {
    A a = new A();
    B b = (B)a;
  }
}
```

 ¿Qué va a pasar?

Herencia de interfaz

Herencia de interfaz

- Muchos lenguajes permite describir interfaces[4]
- Un interfaz declara un conjunto de métodos que luego las clases tendrán que implementar
- Nos va a recordar a las clases abstactas
- Pero se usan más y con más riqueza que las clases abstractas

[4] También 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? ¡ 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);
  }
}
```

Genéricos

Tuplas

- Pero. . . ¿tuplas de qué?

[5] Vale `Integer` $\times$ `Integer`

Tuplas

- Pero. . . ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}^5$

[5] Vale `Integer` $\times$ `Integer`

Tuplas

- Pero... ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}^5$

```
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;
  }
}
```

[5] Vale `Integer` $\times$ `Integer`

Tuplas de booleanos

```
public class TuplaBool {
  private Boolean x;
  private Boolean y;

  public TuplaInt(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 TuplaInt(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 TuplaInt(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 TuplaInt(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 TuplaInt(T1 fst,
                  T2 snd) {
    x = fst;
    y = snd;
  }

  public T1 fst() {
    return x;
  }

  public T2 snd() {
    return y;
  }
}
```

Y ahora la magia (`javac -ea ...`)

```
public class PruebaTuplas {
  public void static 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;
  }
}
```

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;
    }
  }
}
```