




Implementación e implantación de sistemas software

María Woodruff Vázquez



Delegación



Cuando usamos delegación un objeto transfiere la responsabilidad de realizar una tarea a otro objeto. En lugar de realizar directamente la tarea en sí mismo, el objeto delega la tarea a otro objeto que es capaz de realizarla de manera más eficiente o especializada.

A diferencia de la herencia, evita la dependencia directa de otras clases.



Orquesta

En este ejemplo, tenemos una clase Orquesta que es un conjunto de instrumentos, que son de la clase Instrumento. Ambas clases usan Enumerable, que es un módulo en Ruby que proporciona métodos que se pueden utilizar para hacer que una clase sea iterable. Esto es el equivalente al Iterable de Java que se usa en la clase Orquesta original.

```
class Orquesta
  include Enumerable

  def initialize
    @instrumentos = Instrumentos.new(3)
  end

  def addInstrumento(i)
    @instrumentos.add(i)
  end

  def removeInstrumento(i)
    @instrumentos.remove(i)
  end

  def each(&block)
    @instrumentos.each(&block)
  end

  def tocar
    @instrumentos.each { |i| i.tocar unless i.nil? }
  end

  def afinar(i)
    i.afinar
    i.tocar
  end
end
```

```
class Instrumentos
  include Enumerable

  def initialize(numero)
    @instrumentos = Array.new(numero)
  end

  def add(i)
    @instrumentos.push(i)
  end

  def remove(i)
    @instrumentos.delete(i)
    i
  end

  def each(&block)
    @instrumentos.each(&block)
  end
end
```

```
class Viento
  def tocar
    puts "Tocando instrumento de viento."
  end

  def afinar
    puts "Afinando instrumento de viento."
  end
end
```

```
class Cuerda
  def tocar
    puts "Tocando instrumento de cuerda."
  end

  def afinar
    puts "Afinando instrumento de cuerda."
  end
end
```

```
class Percusion
  def tocar
    puts "Tocando instrumento de percusión."
  end

  def afinar
    puts "Afinando instrumento de percusión."
  end
end
```

```
class PruebaOrquesta
  def self.main

    orquesta = Orquesta.new
    viento = Viento.new
    cuerda = Cuerda.new
    percusion = Percusion.new

    orquesta.addInstrumento(viento)
    orquesta.addInstrumento(cuerda)
    orquesta.addInstrumento(percusion)

    orquesta.each do |i|
      orquesta.afinar(i) unless i.nil?
    end

    orquesta.tocar
  end
end
```

PruebaOrquesta.main



Ejecución en Ruby


Instalar Ruby en Mac: `brew install ruby`

Compilar y ejecutar: `ruby archivo.rb`

- mariawoodruff@wlanipr10193 delegacion % ruby orchestra.rb
Afinando instrumento de viento.
Tocando instrumento de viento.
Afinando instrumento de cuerda.
Tocando instrumento de cuerda.
Afinando instrumento de percusión.
Tocando instrumento de percusión.
Tocando instrumento de viento.
Tocando instrumento de cuerda.
Tocando instrumento de percusión.
-



Anotaciones



Queremos comparar dos cuentas de banco y hay varios comparadores: comparador por id y comparador por día de creación.

Usando las anotaciones de Java, podemos inyectar como dependencia el comparador a usar.

Con `@Comparator` indicamos qué implementación de la interfaz `Comparator` debe usarse para comparar dos objetos de `BankAccount`.

```
public final class BankAccount implements Comparable<BankAccount> {  
    private final String id;  
    private LocalDate creationDate;  
  
    @Comparator(BankAccountComparatorByCreationDate.class)  
    private ComparatorInterface comparator;  
  
    public BankAccount(String number) {  
        this.id = number;  
    }  
  
    public LocalDate getCreationDate() {  
        return creationDate;  
    }  
  
    public void setCreationDate(LocalDate date) {  
        this.creationDate = date;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public void setComparator(ComparatorInterface cmp) {  
        comparator = cmp;  
    }  
  
    @Override  
    public int compareTo(BankAccount other) {  
        if (this == other)  
            return 0;  
        assert this.equals(other) : "compareTo inconsistent with equals.";  
        return comparator.compare(this, other);  
    }  
}
```

```
public class BankAccountComparatorByCreationDate implements ComparatorInterface {  
    @Override  
    public int compare(BankAccount bankAccount, BankAccount other) {  
        return bankAccount.getCreationDate().compareTo(other.getCreationDate());  
    }  
}
```

```
public class BankAccountComparatorById implements ComparatorInterface {  
    @Override  
    public int compare(BankAccount bankAccount, BankAccount other) {  
        return bankAccount.getId().compareTo(other.getId());  
    }  
}
```

```
public interface ComparatorInterface {  
    public int compare(BankAccount bankAccount, BankAccount other);  
}
```

```
@Retention(RUNTIME)  
@Target({ FIELD })  
public @interface Comparator {  
    Class<? extends ComparatorInterface> value() default BankAccountComparatorById.class;  
}
```

```
public class Main {
```

```
Run | Debug
```

```
public static void main(String[] args)
```

```
{
```

```
    BankAccount account1 = new BankAccount(number:"123");
```

```
    BankAccount account2 = new BankAccount(number:"456");
```

```
    account1.setCreationDate(LocalDate.of(year:2022, month:3, dayOfMonth:27));
```

```
    account2.setCreationDate(LocalDate.of(year:2021, month:1, dayOfMonth:1));
```

```
    // Inyectar un comparador diferente a BankAccountComparatorById
```

```
    account1.setComparator(new BankAccountComparatorByCreationDate());
```

```
    System.out.println("account1: " + account1 + " " + account1.getCreationDate());
```

```
    System.out.println("account2: " + account2 + " " + account2.getCreationDate());
```

```
    // Comparing two BankAccount objects using the ComparatorInterface implementation
```

```
    // specified by the @Comparator annotation in the BankAccount class
```

```
    int result = account1.compareTo(account2);
```

```
    if (result < 0) {
```

```
        System.out.println(x:"By creation date: account1 is less than account2");
```

```
    } else if (result > 0) {
```

```
        System.out.println(x:"By creation date: account1 is greater than account2");
```

```
    } else {
```

```
        System.out.println(x:"By creation date: account1 is equal to account2");
```

```
    }
```

```
    // Changing the comparator to BankAccountComparatorById
```

```
    account1.setComparator(new BankAccountComparatorById());
```

```
    result = account1.compareTo(account2);
```

```
    if (result < 0) {
```

```
        System.out.println(x:"By id: account1 is less than account2");
```

```
    } else if (result > 0) {
```

```
        System.out.println(x:"By id: account1 is greater than account2");
```

```
    } else {
```

```
        System.out.println(x:"By id: account1 is equal to account2");
```

```
    }
```

```
}
```

```
}
```




Ejecución en Java

Compilación: `javac Main.java BankAccount.java
BankAccountComparatorById.java
ComparatorInterface.java Comparator.java`

Ejecución: `java Main`

```
mariawoodruff@wlanipr10193 anotaciones % java Main  
account1: 123 2022-03-27  
account2: 456 2021-01-01  
By creation date: account1 is greater than account2  
By id: account1 is less than account2
```

—



Aspectos



Los aspectos en Java sirven para el registro, la seguridad o el control de errores. Vamos a usar aspectos en Java usando interceptación de métodos con Guice.

En este ejemplo tenemos a la clase `LoggingAspect` que intercepta las llamadas a los métodos `Line` y `Point` y escribe por pantalla: “Se ha llamado al método x”. Para aplicar el aspecto usamos la clase `MyAppModule` que indica que `LoggingAspect` se debe aplicar a las clases `Line` y `Point`. Además es necesaria una interfaz de anotación personalizada llamada `Interceptor`. Finalmente, en la clase `Main` se crea un objeto `Injector` y se llama a los métodos comprobando que se imprime el mensaje por pantalla.

```
public class Line {  
    private Point p1, p2;  
  
    Point getP1() { return p1; }  
    Point getP2() { return p2; }  
  
    void setP1(Point p1) { this.p1 = p1; }  
    void setP2(Point p2) { this.p2 = p2; }  
}
```

```
public class Point{  
    private int x = 0, y = 0;  
  
    int getX() { return x; }  
    int getY() { return y; }  
  
    @Interceptor  
    void setX(int x) { this.x = x; }  
  
    @Interceptor  
    void setY(int y) { this.y = y; }  
}
```

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Interceptor {}
```

```
public class LoggingAspect implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation invocation) throws Throwable {
        String methodName = invocation.getMethod().getName();
        System.out.println("LoggingAspect: " + methodName + " was called.");
        return invocation.proceed();
    }
}
```

```
public class MyAppModule extends AbstractModule {  
    @Override  
    protected void configure() {  
        bind(clazz:Line.class);  
        bind(clazz:Point.class);  
        bindInterceptor(  
            Matchers.any(),  
            Matchers.annotatedWith(annotationType:Interceptor.class),  
            new LoggingAspect() // Aquí se usa la instancia de LoggingAspect  
        );  
    }  
}
```

```
public class MyApp {
```

Run | Debug

```
public static void main(String[] args) {
```

```
    Injector injector = Guice.createInjector(new MyAppModule());
```

```
    Line line = injector.getInstance(type:Line.class);
```

```
    Point point = injector.getInstance(type:Point.class);
```

```
    line.setP1(new Point());
```

```
    line.setP2(new Point());
```

```
    point.setX(x:10);
```

```
    point.setY(y:20);
```

```
}
```

```
}
```


Ejecución en Java



Compilación: `javac MyApp.java
MyAppModule.java Point.java
Line.java Interceptor.java
LoggingAspect.java`

Ejecución: `java MyApp`

```
[INFO]
[INFO] --- compiler:3.8.1:compile (default-compile) @ aspectos ---
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.489 s
[INFO] Finished at: 2023-03-30T16:05:23+02:00
[INFO] -----
mariawoodruff@wlanipr10193 aspectos % mvn exec:java -Dexec.mainClass="maria.MyApp"
[INFO] Scanning for projects...
[INFO] -----< maria:aspectos >-----
[INFO] Building aspectos 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- exec:3.1.0:java (default-cli) @ aspectos ---
LoggingAspect: setX was called.
LoggingAspect: setY was called.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.550 s
[INFO] Finished at: 2023-03-30T16:05:27+02:00
[INFO] -----
mariawoodruff@wlanipr10193 aspectos %
```