Patrones de Diseño: Strategy

Análisis & Diseño de Software/Fundamentos de Ingeniería de Software

Pablo Cruz Navea-Gastón Márquez-Hernán Astudillo
Departamento de Informática
Universidad Técnica Federico Santa María

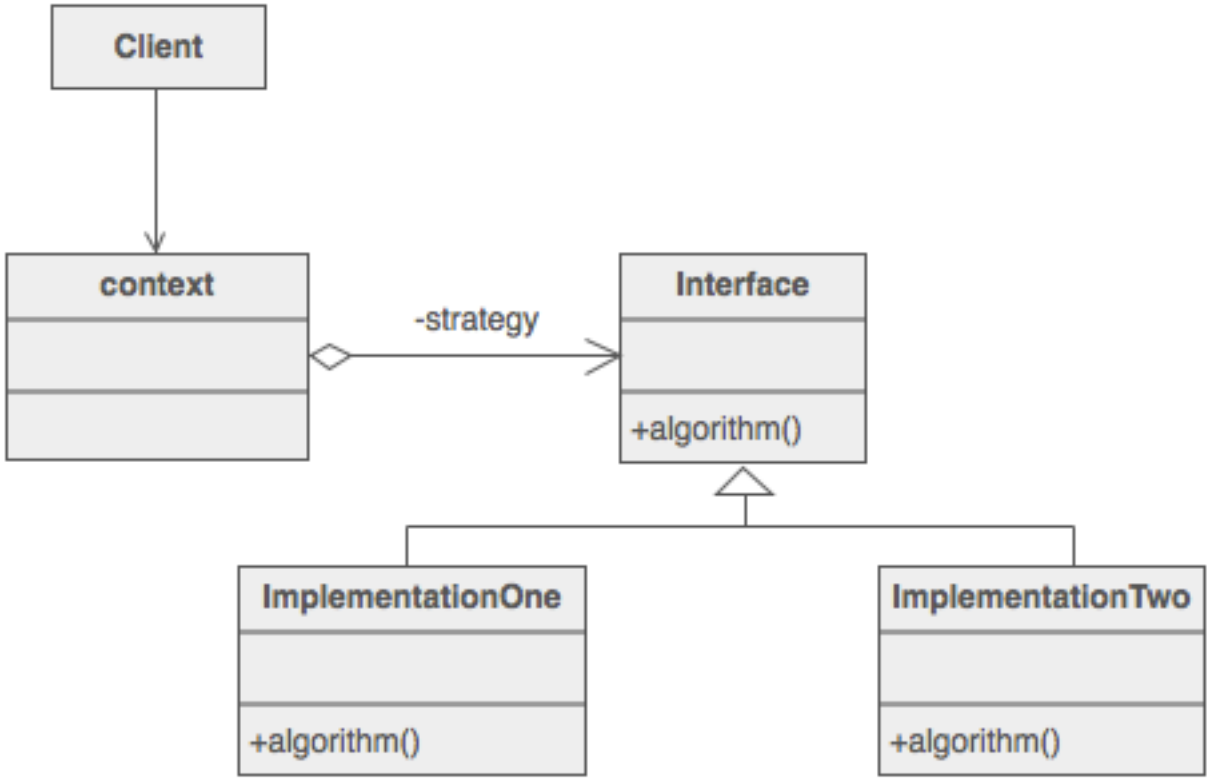

STRATEGY-

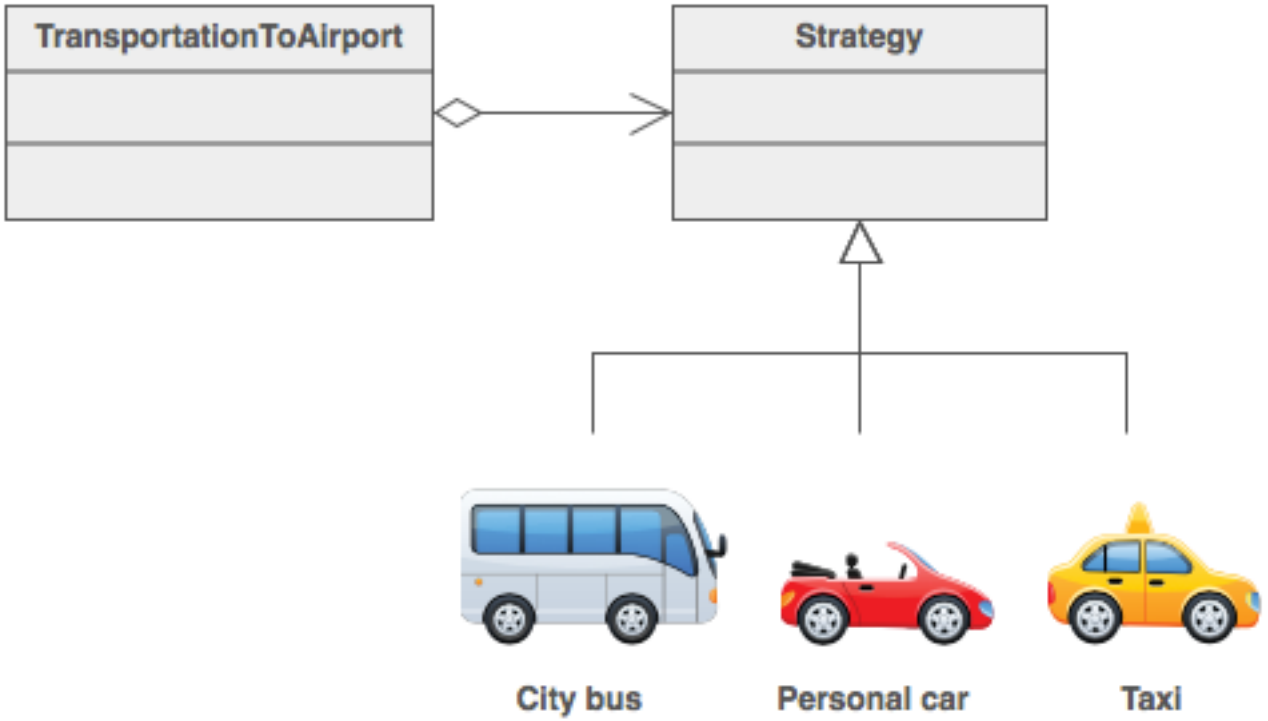
Strategy [1]

- Patrón de diseño de comportamiento
- Propósito: cuando existe una familia de algoritmos, encapsulamos cada uno de ellos para hacerlos intercambiables
- Con Strategy los algoritmos pueden variar independiente de los clientes que hacen uso de ellos
- Strategy aplica de manera natural a los casos en los que debe seleccionarse uno de los algoritmos según el contexto

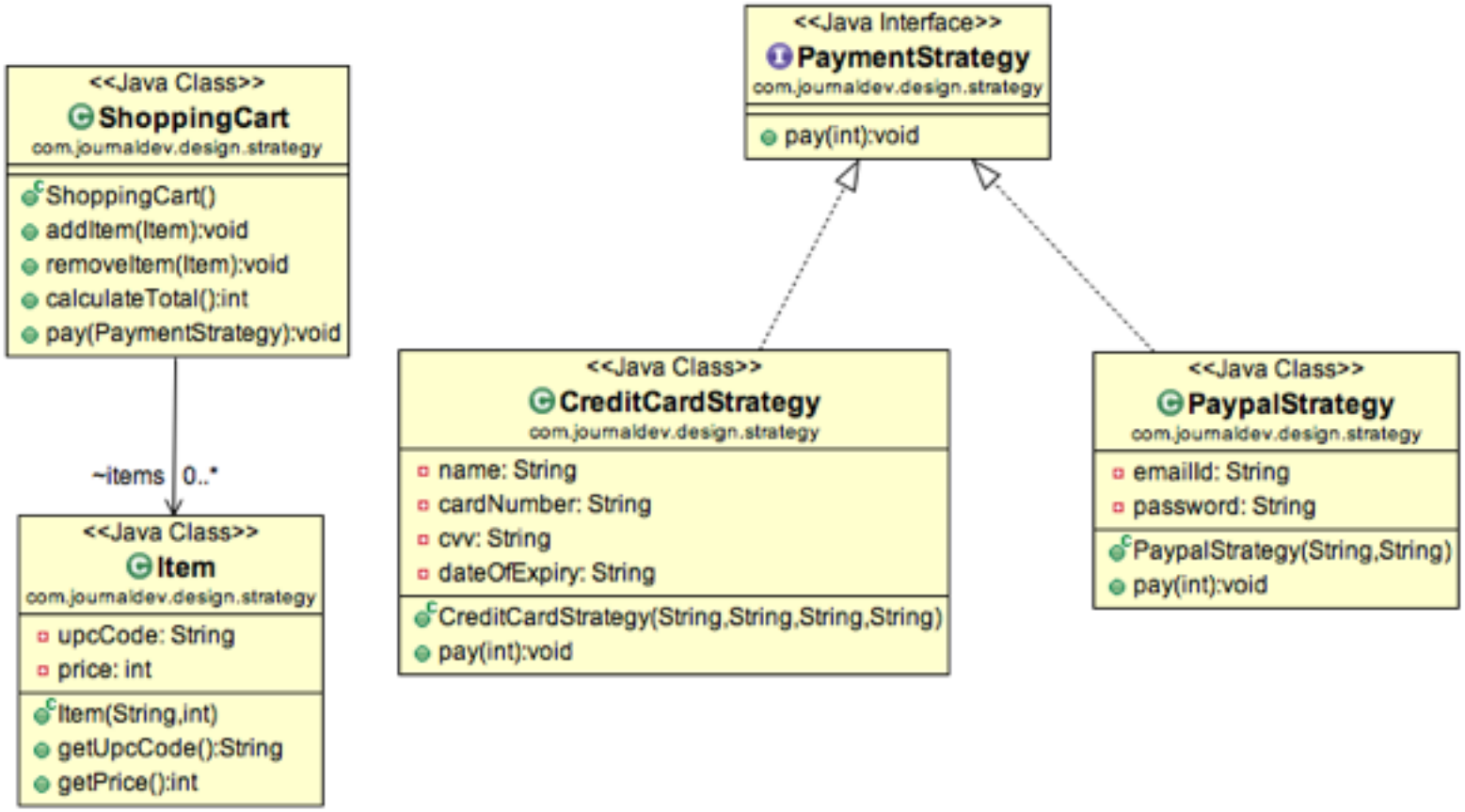
Strategy [2]

- “Seleccionar un algoritmo” podría ser resuelto fácilmente con una serie de estructuras *if-then-else*
- ¿Por qué usar un patrón de diseño para resolver este problema de selección de algoritmos?
  - Porque así los algoritmos quedan encapsulados en clases distintas y pueden variar libremente
    - Cada desarrollador puede encargarse de una clase “estrategia” sin entorpecer el trabajo de los demás desarrolladores


Client
context
-strategy
Interface
+algorithm()
ImplementationOne
+algorithm()
ImplementationTwo
+algorithm()


TransportationToAirport
Strategy
City bus
Personal car
Taxi

Ejemplo: CiberMonday [1]



Ejemplo: CiberMonday [2]

```
package com.journaldev.design.strategy;
public interface PaymentStrategy {
    public void pay(int amount);
}
```

Ejemplo: CiberMonday [3]

```
package com.journaldev.design.strategy;

public class CreditCardStrategy implements PaymentStrategy {
    private String name;
    private String cardNumber;
    private String cvv;
    private String dateOfExpiry;

    public CreditCardStrategy(String nm, String ccNum, String cvv, String
expiryDate){
        this.name=nm;
        this.cardNumber=ccNum;
        this.cvv=cvv;
        this.dateOfExpiry=expiryDate;
    }
    @Override
    public void pay(int amount) {
        System.out.println(amount +" paid with credit/debit card");
    }
}
```

Ejemplo: CiberMonday [4]

```
package com.journaldev.design.strategy;

public class PaypalStrategy implements PaymentStrategy {
    private String emailId;

    private String password;

    public PaypalStrategy(String email, String pwd){
        this.emailId=email;
        this.password=pwd;
    }
    @Override
    public void pay(int amount) {
        System.out.println(amount + " paid using Paypal.");
    }
}
```

Ejemplo: CiberMonday [5]

```
public class Item {
    private String upcCode;
    private int price;

    public Item(String upc, int cost){
        this.upcCode=upc;
        this.price=cost;
    }

    public String getUpcCode() {
        return upcCode;
    }

    public int getPrice() {
        return price;
    }
```

Ejemplo: CiberMonday [6]

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
    List<Item> items;
    public ShoppingCart(){
        this.items=new ArrayList<Item>();
    }

    public void addItem(Item item){
        this.items.add(item);
    }

    public void removeItem(Item item){
        this.items.remove(item);
    }

    public int calculateTotal(){
        int sum = 0;
        for(Item item : items){
            sum += item.getPrice();
        }
        return sum;
    }

    public void pay(PaymentStrategy paymentMethod){
        int amount = calculateTotal();
        paymentMethod.pay(amount);
    }
}
```

Ejemplo: CiberMonday [7]

```
public class ShoppingCartTest {
    public static void main(String[] args) {
        ShoppingCart cart = new ShoppingCart();
        Item item1 = new Item("1234",10);
        Item item2 = new Item("5678",40);
        cart.addItem(item1);
        cart.addItem(item2);
        cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));
        cart.pay(new CreditCardStrategy("Pankaj Kumar", "1234567890123456",
"786", "12/15"));
    }
}
```

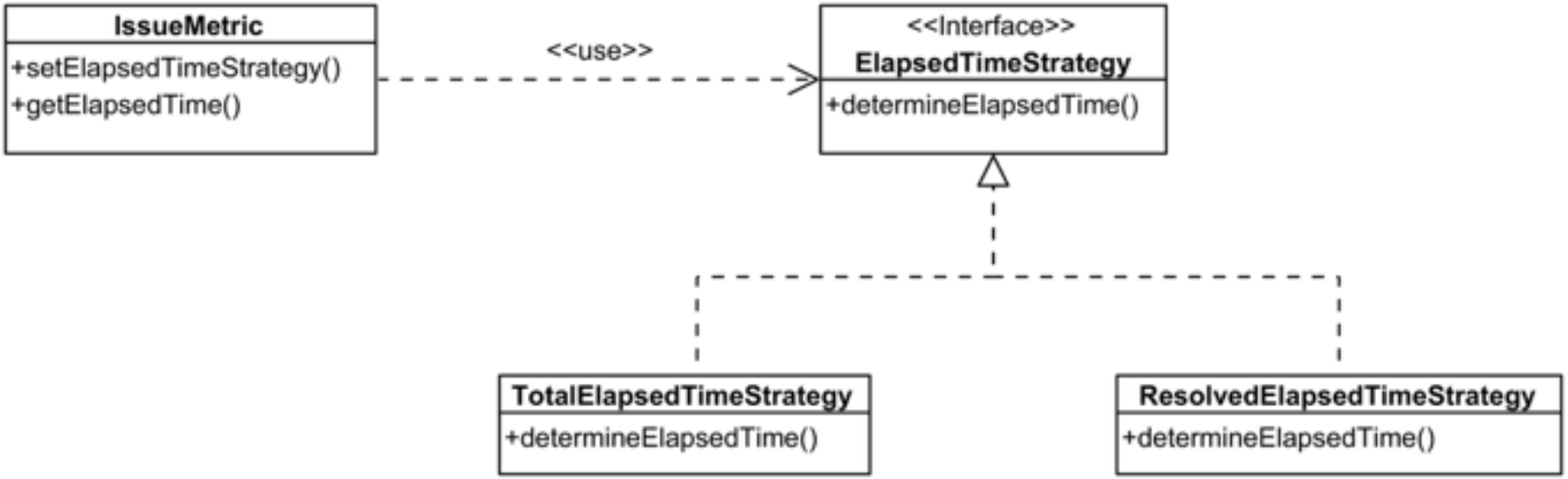
Ejemplo: Strategy con JAVA [1]

- En muchos ejemplos hemos visto el concepto de tiempo transcurrido *(elapsed time)* para los *issues* (incidencias)
- En la realidad, medir el tiempo transcurrido de un *issue* es un asunto un poco más complejo porque existen muchos tipos de *"elapsed time"*
  - ElapsedTime completo
    - Desde que el *issue* se creó hasta que se cerró
  - ElapsedTime de resolución
    - Desde que el *issue* se creó hasta que fue marcado como resuelto (sin probar)
  - ElapsedTime de asignación
    - Desde que el *issue* se creó hasta que fue asignado hacia un equipo de desarrollo

Ejemplo: Strategy con JAVA [2]

- ¿Cómo nos ayuda Strategy en este problema?
  - Encapsulando los distintos algoritmos en clases distintas
- Para el ejemplo, supondremos que un sistema debe medir el tiempo transcurrido total y el tiempo transcurrido para la última resolución
  - Tendremos dos clases que encapsulan los distintos algoritmos:
    - TotalElapsedTimeStrategy
    - ResolvedElapsedTimeStrategy
  - Ambas clases implementan una interfaz que provee el método determineElapsedTime()

Ejemplo: Strategy con JAVA [3]



Ejemplo: Strategy con JAVA [4]

```
// creamos una estrategia general para encontrar el tiempo
// transcurrido
public interface ElapsedTimeStrategy {
        public void determineElapsedTime(Issue issue);
}

// caso sencillo: estrategia 'tiempo total' es lo que hemos
// visto siempre en los ejemplos
public class TotalElapsedTimeStrategy implements
        ElapsedTimeStrategy {
        public void determineElapsedTime(Issue issue) {
                // obtener openDate desde issue
                // obtener closeDate desde issue
                // obtener closeDate - openDate
                return elapsedTime;
        }
}
```

Ejemplo: Strategy con JAVA [5]

```
// caso complejo: estrategia 'tiempo de resolución'
public class ResolvedElapsedTimeStrategy implements
                ElapsedTimeStrategy {
        public void determineElapsedTime(Issue issue) {
                // obtener openDate desde issue
                // obtener de alguna forma la última vez
                // que el issue fue marcado como status = resolved
                // obtener resolvedDate - openDate
                return resolvedElapsedTime;
        }
}
```

Ejemplo: Strategy con JAVA [6]

```
public class IssueMetric {
        private ElapsedTimeStrategy elapsedTimeStrategy;
        private Issue issue;
        // constructor puede ser usado para establecer
        // estrategia por defecto (si aplica) y para
        // recibir un 'issue'

        public void setElapsedTimeStrategy(ElapsedTimeStrategy
                                    elapsedTimeStrategy) {
                this.elapsedTimeStrategy = elapsedTimeStrategy;
        }

        public int getElapsedTime() {
                return elapsedTimeStrategy.
                        determineElapsedTime(issue);
        }
}
```

Ejemplo: Strategy con JAVA [7]

```
// hacemos uso de las estrategias
IssueMetric issueMetric = new IssueMetric();

issueMetric.setElapedTimeStrategy(new
                                    TotalElapsedTimeStrategy());
issueMetric.getElapsedTime();

// de manera análoga, usamos estrategia para tiempo transcurrido
// de resolución
```

FIN