Ingegneria del Software
Esercitazione

29 Novembre 2023

Davide Yi Xian Hu

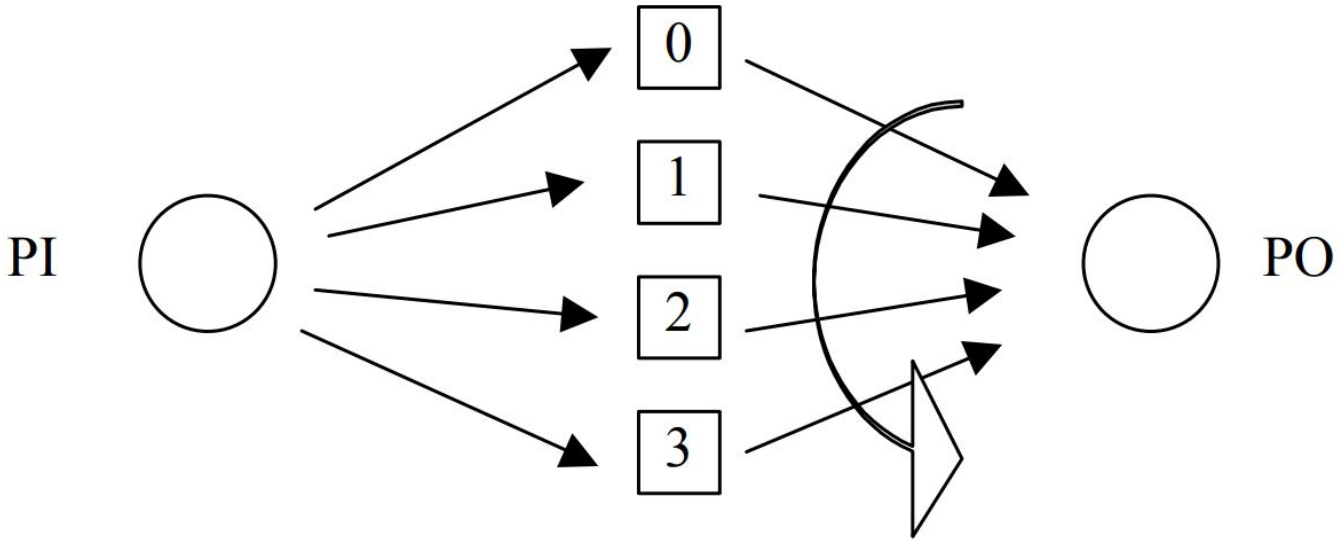
Email: davideyi.hu@polimi.it

POLITECNICO
MILANO 1863

Esercizio 1 / IO Buffer

- 👉 Dato un sistema con quattro buffer di caratteri.
- 👉 Il modulo PI esegue ripetutamente le seguenti operazioni: legge da tastiera una coppia di valori , dove **i** è un numero tra 0 e 3, **ch** un carattere, e inserisce il carattere **ch** nel buffer **i** (ognuno dei quattro buffer contiene al più un carattere).
- 👉 Il modulo PO considera a turno in modo circolare i quattro buffer e preleva il carattere in esso contenuto, scrivendo in uscita la coppia di valori se ha appena prelevato il carattere **ch** dal buffer **i**.
- 👉 L'accesso a ognuno dei buffer è in mutua esclusione.
- 👉 PI rimane bloccato se il buffer a cui accede è pieno, PO se è vuoto.

Esercizio 1 / IO Buffer



Esercizio 1 / IO Buffer

Esempio:

- 👉 Sequenza di input: <1, c> <0, b> <2, m> <0, f> <1, h> <3, n>
- 👉 Sequenza in output: <0, b> <1, c> <2, m> <3, n> <0, f> <1, h>

Considerare in quali casi si puo' avere un deadlock e implementare il sistema.

Esercizio 2 / Parrot Server

Implementare un client server TCP.

- 👉 Il server e’ in ascolto su una porta.
- 👉 Il client si connette al server.
- 👉 L’utente può’ inserire testo dal terminale client, e il client lo spedira’ al server.
- 👉 Il server risponde con lo stesso messaggio ricevuto dal client.
- 👉 Se l’utente digita “exit”, il client si disconnette.

Esercizio 3 / Parrot Server+

Estendere Parrot Server.

- 👉 Rendere non bloccante il server utilizzando i thread.
- 👉 Collegare diversi client al server e verificare il corretto funzionamento del server.
- 👉 Implementare un protocollo per gestire quattro tipi di messaggio:
  - Login Message: Il client invia il proprio nome utente.
  - Text Message: Il server risponde con il nome utente del mittente e il contenuto del messaggio ricevuto.
  - Reversed Message: Il server risponde con il nome utente del mittente e il contenuto al contrario del messaggio ricevuto.
  - Exit Message: Il server disconnette il client.

Esercizio 4 / Singleton

Il pattern Singleton garantisce che una classa ha una sola istanza e permette un punto di accesso globale a quella istanza.

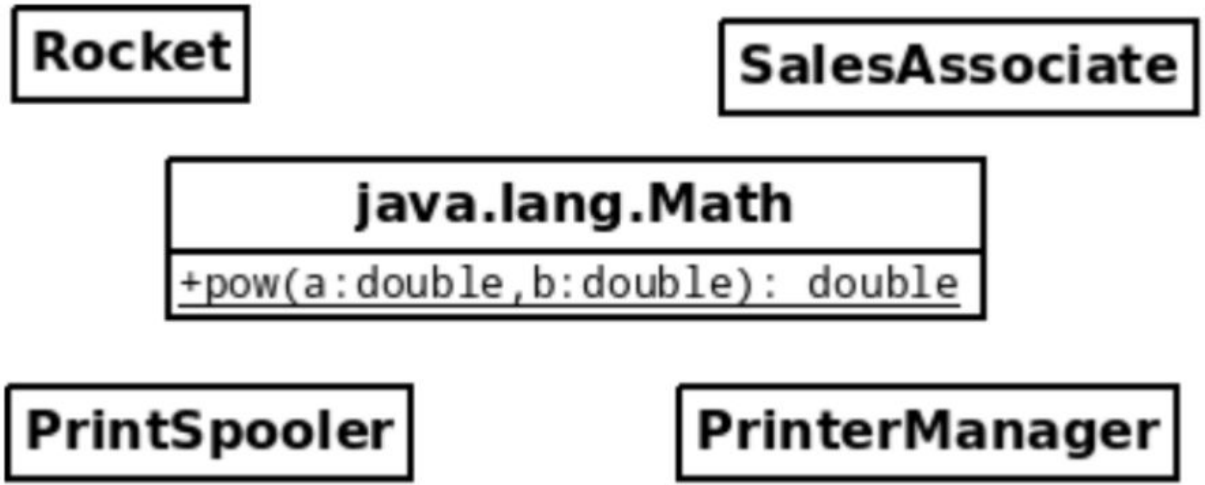
```
public class Singleton {
    private static Singleton instance;

    private Singleton() {
        // private constructor to prevent instantiation
    }

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Esercizio 4 / Singleton

Date le seguenti classi dire dove si puo' applicare il pattern Singleton



Esercizio 5 / Strategy (parte 1)

Il pattern Strategy definisce una famiglia di algoritmi, li incapsula e li rende intercambiabili.

```
public interface PaymentStrategy {
    void pay(int amount);
}

public class CreditCardPayment implements PaymentStrategy {
    private String cardNumber;

    public CreditCardPayment(String cardNumber) {
        this.cardNumber = cardNumber;
    }

    @Override
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using credit card: " + cardNumber);
    }
}
```

Esercizio 5 / Strategy (parte 2)

Il pattern Strategy definisce una famiglia di algoritmi, li incapsula e li rende intercambiabili.

```
public class PayPalPayment implements PaymentStrategy {
    private String email;

    public PayPalPayment(String email) {
        this.email = email;
    }

    @Override
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using PayPal with email: " + email);
    }
}
```

Esercizio 5 / Strategy (parte 3)

Il pattern Strategy definisce una famiglia di algoritmi, li incapsula e li rende intercambiabili.

```
public class ShoppingCart {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }

    public void checkout(int amount) {
        paymentStrategy.pay(amount);
    }
}
```

Esercizio 5 / Strategy

Considera un'applicazione di elaborazione del testo che deve eseguire diverse operazioni di formattazione del testo.

- 👉 Definisci un'interfaccia **TextFormatter** con un metodo formatText(String text).

- 👉 Implementa tre classi che implementano l'interfaccia **TextFormatter**:
  - **UpperCaseFormatter**: Converte il testo di input in maiuscolo.
  - **LowerCaseFormatter**: Converte il testo di input in minuscolo.
  - **TitleCaseFormatter**: Converte il testo di input in maiuscolo all'inizio di ogni parola.
- 👉 Crea una classe **TextEditor** che ha un campo **TextFormatter**. Questa classe dovrebbe avere un metodo **string formatText(String text)** che delega la formattazione alla strategia.

- 👉 Permetti agli utenti di impostare dinamicamente la strategia di formattazione.

Esercizio 6 / Factory

Il pattern Factory definisce una interfaccia per creare un oggetto ma lascia la scelta del tipo alla sottoclasse.

```
public interface Product {
    void create();
}

public class ConcreteProductA implements Product {
    @Override
    public void create() {
        System.out.println("Product A created");
    }
}

public class ConcreteProductB implements Product {
    @Override
    public void create() {
        System.out.println("Product B created");
    }
}
```

Esercizio 6 / Factory

Il pattern Factory definisce una interfaccia per creare un oggetto ma lascia la scelta del tipo alla sottoclasse.

```
public interface Creator {
    Product factoryMethod();
}

public class ConcreteCreatorA implements Creator {
    @Override
    public Product factoryMethod() {
        return new ConcreteProductA();
    }
}

public class ConcreteCreatorB implements Creator {
    @Override
    public Product factoryMethod() {
        return new ConcreteProductB();
    }
}
```

Esercizio 6 / Factory

- 👉 Rappresentare in UML ed implementare in Java un insieme di classi che permettano la creazione di una PetSchool.
- 👉 Una PetSchool è una scuola composta da animali invece che da essere umani.
- 👉 Ogni PetSchool è composta da animali dello stesso tipo (solo gatti, solo cani, etc.) in numero pari a 50.
- 👉 Il costruttore di PetSchool deve riempire la scuola. Usare un adeguato design pattern.

Esercizio 6 / Factory