e: office@sirma.bg m: +359 2 9768310

SOLID - Exercises

1. Single Responsibility Principle (SRP)

Task: Separate the database operations from the Book class, ensuring that each class has a single responsibility.

```
public class Book {
   private String title;
   private String author;
   // ... other properties

public void saveToDatabase() {
      // Save book to the database
   }

public String getBookSummary() {
    return title + " by " + author;
   }
}
```

Task: Split the Invoice class so that the database and printing operations are segregated into their respective classes.

```
public class Invoice {
   private double amount;
   private String customerName;
   // ... other properties

   public void printInvoice() {
        // Print invoice
   }

   public void saveInvoice() {
        // Save invoice to database
   }
}
```

2. Open/Closed Principle (OCP)

Task: Refactor the Logger class to allow adding more logging methods in the future without changing the existing code.





e: office@sirma.bg m: +359 2 9768310

```
public class Logger {
   public void logToConsole(String message) {
      System.out.println(message);
   }
   public void logToFile(String message, String filename) {
      // Code to write message to a file
   }
}
```

Task: Refactor the DiscountCalculator class so that you can introduce new discount types without modifying existing code.

```
public class DiscountCalculator {
   public double calculateDiscount(String type, double price) {
     if ("STUDENT".equals(type)) {
        return price * 0.1;
     } else if ("SENIOR".equals(type)) {
        return price * 0.2;
     }
     return price;
   }
}
```

3. Liskov Substitution Principle (LSP)

Task: Address the violation of LSP in the above inheritance hierarchy.

```
public class Engine {
   public void start() {
      // Start the engine
   }
}

public class ElectricEngine extends Engine {
   @Override
   public void start() {
      throw new UnsupportedOperationException("Electric engines don't start traditionally.");
   }
}
```

Task: Penguins don't fly! Modify the class design to adhere to LSP.

public class Bird {





e: office@sirma.bg m: +359 2 9768310

```
public void fly() {
    //...
}
}
public class Penguin extends Bird { }
```

4. Interface Segregation Principle (ISP)

Task: Not all machines can print, fax, and scan. Separate these capabilities into individual interfaces.

```
public interface Machine {
   void print();
   void fax();
   void scan();
}
```

Task: Some media players might only support play and pause. Refactor the interface to ensure that no player class implements unnecessary methods.

```
public interface Player {
   void play();
   void pause();
   void next();
   void previous();
   void shuffle();
}
```

5. Dependency Inversion Principle (DIP)

Task: Decouple LightSwitch from the concrete Bulb class using an appropriate abstraction.

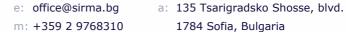
```
public class LightSwitch {
    private Bulb bulb;

    public void operate() {
        // Toggle bulb state
    }
}

public class Bulb {
    public void turnOn() { /*...*/ }
    public void turnOff() { /*...*/ }
}
```









Task: Refactor the WeatherReporter class so that it doesn't depend on a specific temperature sensor implementation.

```
public class WeatherReporter {
  private TemperatureSensor sensor;
  public String report() {
     return "Current temperature: " + sensor.getTemperature();
  }
}
public class TemperatureSensor {
  public double getTemperature() {
     // Return temperature from sensor
     return 25.0; // dummy value
}
```

