

Laiba Binte Tahir

FA21-BSE-019

Lab Assignment-01

Question 1: How does the following code violates “Single responsibility Principle”? Also Write the correct code.

<pre>class Marker{ String name; String color; int year; int price; public Marker(String name, String color, int year, int price) { this.name = name; this.color = color; this.year = year; this.price = price; } }</pre>	<pre>class Invoice{ private Marker marker; private int quantity; public Invoice(Marker marker, int quantity) { this.marker = marker; this.quantity = quantity; } public int calculateTotal(){ int price = ((marker.price)*this.quantity); return price; } public void printInvoice(){ // print the invoice } public void saveToDB(){ //save the data into DB } }</pre>
---	---

Violation:

A class only handles one responsibility in a single responsibility pattern so in above code we should make:

- InvoiceClass to handle only calculations.
- InvoicePrinter Class to handle only printing work.
- InvoiceRepository To handle database thus saving invoices to DB.

1. Invoice Class

```
1 package org.example;
2
3 class Invoice {
4     private Marker marker;
5     private int quantity;
6
7     public Invoice(Marker marker, int quantity) {
8         this.marker = marker;
9         this.quantity = quantity;
10    }
11
12    public int calculateTotal() {
13        int price = (marker.price * quantity);
14        return price;
15    }
16 }
```

2. MarkerClass

```
package org.example;

class Marker {
    String name;
    String color;
    int year;
    int price;

    public Marker(String name, String color, int year, int price) {
        this.name = name;
        this.color = color;
        this.year = year;
        this.price = price;
    }
}
```

3. InvoicePrinter Class

```
package org.example;

class InvoicePrinter {
    public void printInvoice(Invoice invoice) {
        System.out.println("Invoice total: " + invoice.calculateTotal());
    }
}
```

4. InvoiceRepository

```
package org.example;

class InvoiceRepository {
    public void saveToDB(Invoice invoice) {
        System.out.println("Invoice saved to database.");
    }
}
```

5. MainClass

```
package org.example;

To Run code, press Shift F10 or click the  icon in the gutter.

public class Main {
    public static void main(String[] args) {
        Marker marker = new Marker(name: "MarkerBrand", color: "Black", year: 2024, price: 50);

        Invoice invoice = new Invoice(marker, quantity: 5);
        InvoicePrinter printer = new InvoicePrinter();
        printer.printInvoice(invoice);

        InvoiceRepository repository = new InvoiceRepository();
        repository.saveToDB(invoice);
    }
}
```

6. Output



```
.gitignore
Run Main x
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA C
Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.
Binta Tahir\IdeaProjects\SingleResponsibiltyPrinciple\target\classes" org.example.Main
Invoice total: 250
Invoice saved to database.
Process finished with exit code 0
```

Question 2: Write the correct code by applying Liskov substitution principle.

```
public class Bird {
    public void fly() {
        // Flying logic
    }
}

public class Ostrich extends Bird {
    // Ostriches cannot fly, but they inherit the
    fly method
}
```

The superclass can easily substitute with other classes without effecting implantation. In above scenario we must make:

- BirdClass without Fly() method
- Make another flyingbird and orchid classes with appropriate methods.

BirdClass

```
Main.java  Bird.java  FlyingBird.java
1 package org.example;
2
3 public abstract class Bird {
4     public abstract void move();
5 }
6
```

FlyingBirdClass

```
Main.java  Bird.java  FlyingBird.java
1 package org.example;
2
3 class FlyingBird extends Bird {
4     @Override
5     public void move() {
6         fly();
7     }
8
9     public void fly() {
10        System.out.println("Flying");
11    }
12 }
```

OrchidClass



The screenshot shows an IDE with three tabs: Main.java, Bird.java, and FlyingBird.java. The Ostrich class is defined in Bird.java, extending Bird. It has two methods: move() and run(). The move() method calls run(). The run() method prints "Running" to the console. A lightbulb icon is visible next to the run() method, indicating a suggestion or tip.

```
1 package org.example;
2
3 // Ostrich Class (Non-Flying Bird)
4 class Ostrich extends Bird {
5     @Override
6     public void move() {
7         run();
8     }
9
10    public void run() {
11        System.out.println("Running");
12    }
13 }
```

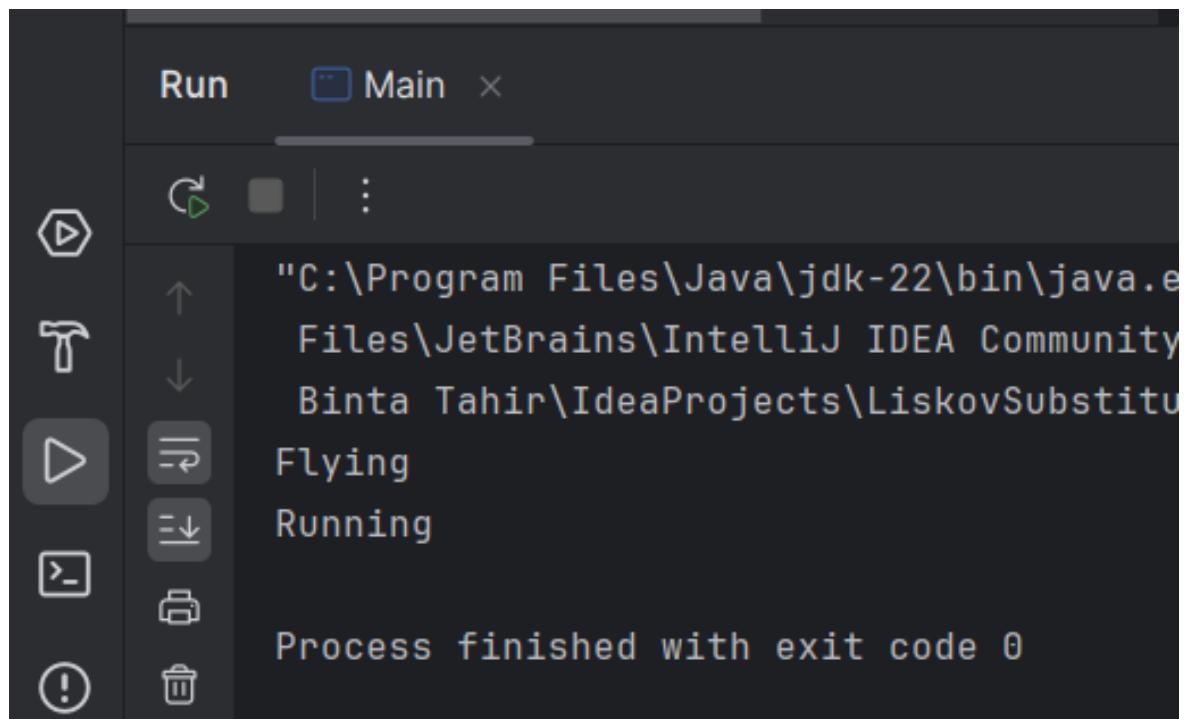
MainClass



The screenshot shows an IDE with four tabs: Main.java, Bird.java, FlyingBird.java, and Ostrich.java. The Main class is defined in Main.java. It has a main method that creates two Bird objects: a FlyingBird object named sparrow and an Ostrich object named ostrich. Both objects have their move() method called. A red circle icon is visible next to the closing brace of the main method, indicating an error or warning.

```
1 package org.example;
2 public class Main {
3     public static void main(String[] args) {
4         Bird sparrow = new FlyingBird();
5         sparrow.move();
6
7         Bird ostrich = new Ostrich();
8         ostrich.move();
9     }
10 }
```

Output:



```
Run    Main x
[C]
"C:\Program Files\Java\jdk-22\bin\java.exe"
Files\JetBrains\IntelliJ IDEA Community
Binta Tahir\IdeaProjects\LiskovSubstitu
Flying
Running
Process finished with exit code 0
```