Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

Week 1:

Exercise 1: Implementing the Singleton Pattern

Code Implementation:

i. Singleton.java

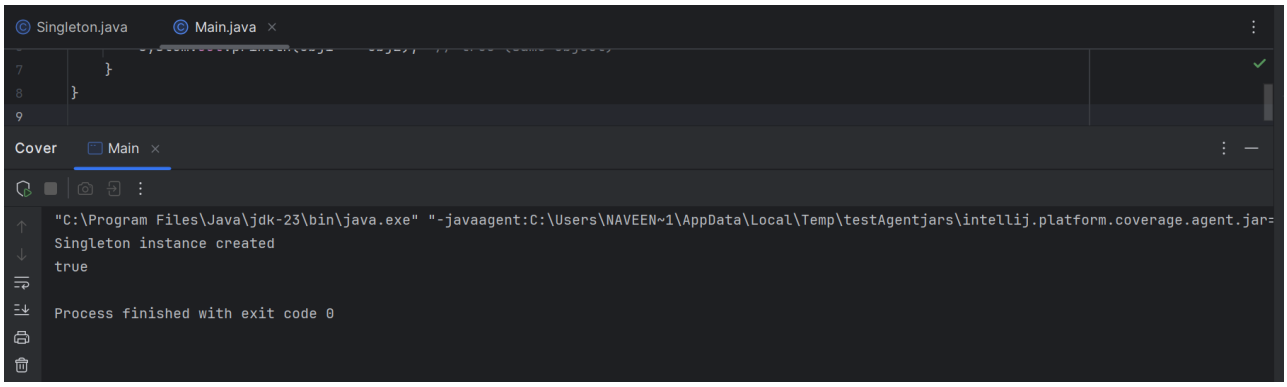
```
public class Singleton {
    private static Singleton instance;

    // Private constructor
    private Singleton() {
        System.out.println("Singleton instance created");
    }

    // Public method to provide access
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

ii. Main.java

```
public class Main {
    public static void main(String[] args) {
        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();

        System.out.println(obj1 == obj2);  // true (same object)
    }
}
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

OUTPUT:



Exercise 2: Implementing the Factory Method Pattern

Code Implementation:

i.Interface Shape

```
public interface Shape {
   void draw();
}
```

ii.Circle.java

```
public class Circle implements Shape {

   public void draw() {

      System.out.println("Drawing a Circle");

   }

}
```

iii.Square.java

```
public class Square implements Shape {

   public void draw() {

      System.out.println("Drawing a Square");

   }

}
```

iv.ShapeFactory.java

```
public class ShapeFactory {

   public static Shape getShape(String type) {

      if (type.equalsIgnoreCase("circle")) {
```

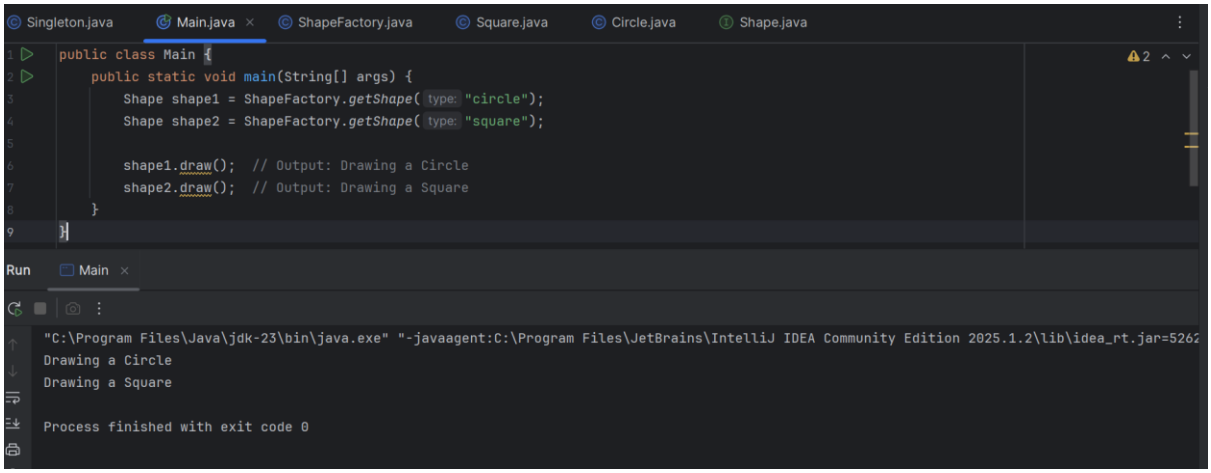
Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
            return new Circle();
        } else if (type.equalsIgnoreCase("square")) {
            return new Square();
        } else {
            return null;
        }
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Shape shape1 = ShapeFactory.getShape("circle");
        Shape shape2 = ShapeFactory.getShape("square");

        shape1.draw();
        shape2.draw();
    }
}
```

OUTPUT:



Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

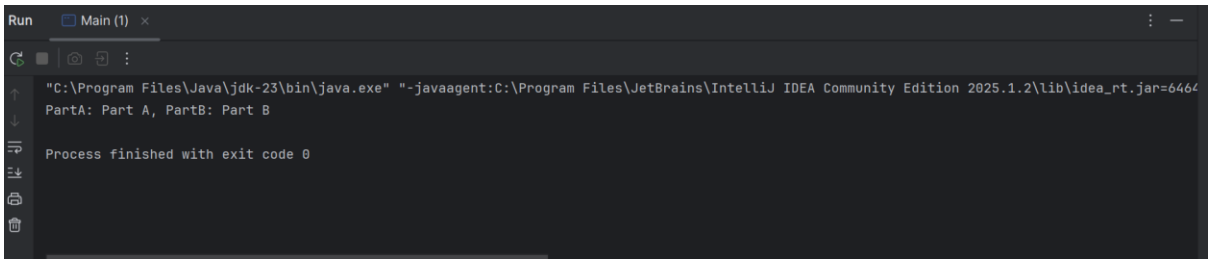
Exercise 3: Implementing the Builder Pattern

```
class Product {
    private String partA;
    private String partB;

    public void setPartA(String partA) {
        this.partA = partA;
    }

    public void setPartB(String partB) {
        this.partB = partB;
    }

    public void show() {
        System.out.println("PartA: " + partA + ", PartB: " + partB);
    }
}

interface Builder {
    void buildPartA();
    void buildPartB();
    Product getResult();
}

class ConcreteBuilder implements Builder {
    private Product product = new Product();

    @Override
    public void buildPartA() {
        product.setPartA("Part A");
```

```
    }

    @Override
    public void buildPartB() {
        product.setPartB("Part B");
    }

    @Override
    public Product getResult() {
        return product;
    }
}

class Director {
    public void construct(Builder builder) {
        builder.buildPartA();
        builder.buildPartB();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Director director = new Director();
        Builder builder = new ConcreteBuilder();
        director.construct(builder);
        Product product = builder.getResult();
        product.show();
    }
}
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

output:



Exercise 4: Implementing the Adapter Pattern

```
interface Target { void request(); }

class Adaptee { public void specificRequest() { System.out.println("Specific Request Executed"); } }

class Adapter implements Target {

   private Adaptee adaptee = new Adaptee();

   public void request() { adaptee.specificRequest(); }

}

class AdapterTest {

   public static void main(String[] args) {

      Target adapter = new Adapter();

      adapter.request();

   }

}


// Decorator Pattern Example with Main Method

interface Component { void operation(); }

class ConcreteComponent implements Component {

   public void operation() { System.out.println("Base Operation"); }

}

class Decorator implements Component {

   protected Component component;

   public Decorator(Component component) { this.component = component; }

   public void operation() { component.operation(); }

}
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
class ConcreteDecorator extends Decorator {
    public ConcreteDecorator(Component component) { super(component); }
    public void operation() {
        super.operation();
        System.out.println("Extra Behavior Added");
    }
}
class DecoratorTest {
    public static void main(String[] args) {
        Component decorated = new ConcreteDecorator(new ConcreteComponent());
        decorated.operation();
    }
}
```

OUTPUT:

Exercise 5: Implementing the Decorator Pattern

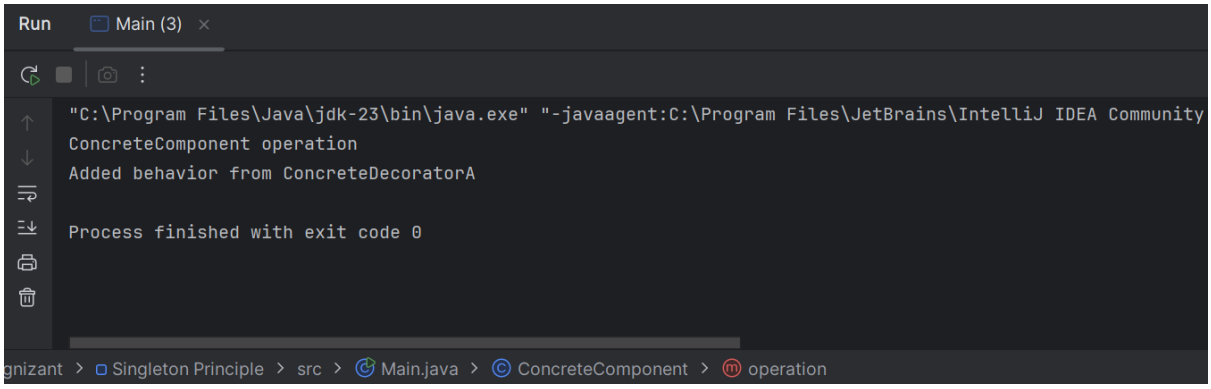
```
interface Component {
    void operation();
}

class ConcreteComponent implements Component {
    @Override
    public void operation() {
        System.out.println("ConcreteComponent operation");
```

```
    }
}

abstract class Decorator implements Component {
    protected Component component;

    public Decorator(Component component) {
        this.component = component;
    }

    @Override
    public void operation() {
        component.operation();
    }
}

class ConcreteDecoratorA extends Decorator {
    public ConcreteDecoratorA(Component component) {
        super(component);
    }

    @Override
    public void operation() {
        super.operation();
        System.out.println("Added behavior from ConcreteDecoratorA");
    }
}

// Usage
public class Main {
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
    public static void main(String[] args) {
        Component component = new ConcreteComponent();
        Component decorated = new ConcreteDecoratorA(component);
        decorated.operation();
    }
}
```

OUTPUT:



Exercise 6: Implementing the Proxy Pattern

```
interface Subject {
    void request();
}

class RealSubject implements Subject {
    @Override
    public void request() {
        System.out.println("RealSubject request");
    }
}

class Proxy implements Subject {
    private RealSubject realSubject;

    @Override
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
    public void request() {
        if (realSubject == null) {
            realSubject = new RealSubject();
        }
        realSubject.request();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Proxy proxy = new Proxy();
        proxy.request();
    }
}
```

OUTPUT:

Exercise 7: Implementing the Observer Pattern

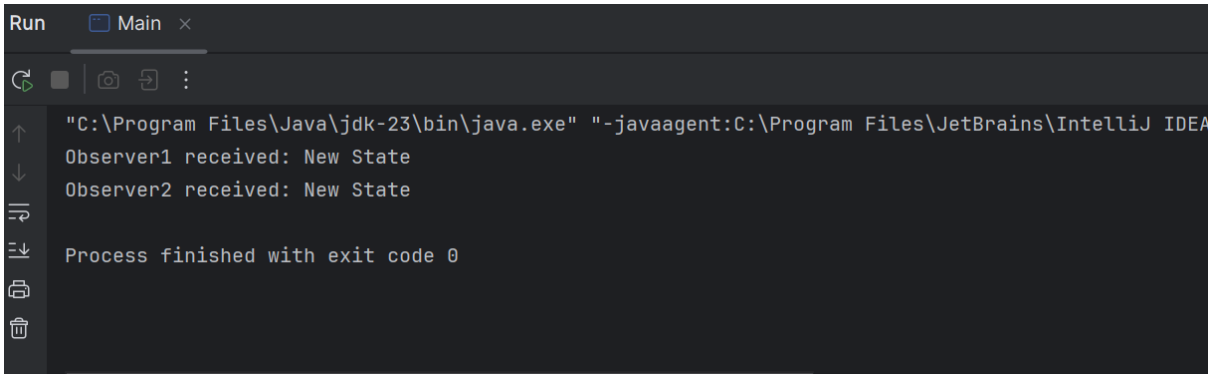
```
import java.util.ArrayList;
import java.util.List;

interface Observer {
    void update(String message);
}
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
class ConcreteObserver implements Observer {
    private String name;

    public ConcreteObserver(String name) {
        this.name = name;
    }

    @Override
    public void update(String message) {
        System.out.println(name + " received: " + message);
    }
}

class Subject {
    private List<Observer> observers = new ArrayList<>();
    private String state;

    public void attach(Observer observer) {
        observers.add(observer);
    }

    public void setState(String state) {
        this.state = state;
        notifyAllObservers();
    }

    private void notifyAllObservers() {
        for (Observer observer : observers) {
            observer.update(state);
```

```
        }
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Subject subject = new Subject();
        Observer observer1 = new ConcreteObserver("Observer1");
        Observer observer2 = new ConcreteObserver("Observer2");

        subject.attach(observer1);
        subject.attach(observer2);
        subject.setState("New State");
    }
}
```

OUTPUT:



Exercise 8: Implementing the Strategy Pattern

```
interface Strategy {
    void execute();
}

class ConcreteStrategyA implements Strategy {
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
    @Override
    public void execute() {
        System.out.println("Strategy A executed");
    }
}

class ConcreteStrategyB implements Strategy {
    @Override
    public void execute() {
        System.out.println("Strategy B executed");
    }
}

class Context {
    private Strategy strategy;

    public void setStrategy(Strategy strategy) {
        this.strategy = strategy;
    }

    public void executeStrategy() {
        strategy.execute();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Context context = new Context();
        context.setStrategy(new ConcreteStrategyA());
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
        context.executeStrategy();


        context.setStrategy(new ConcreteStrategyB());
        context.executeStrategy();
    }
}
```

OUTPUT:

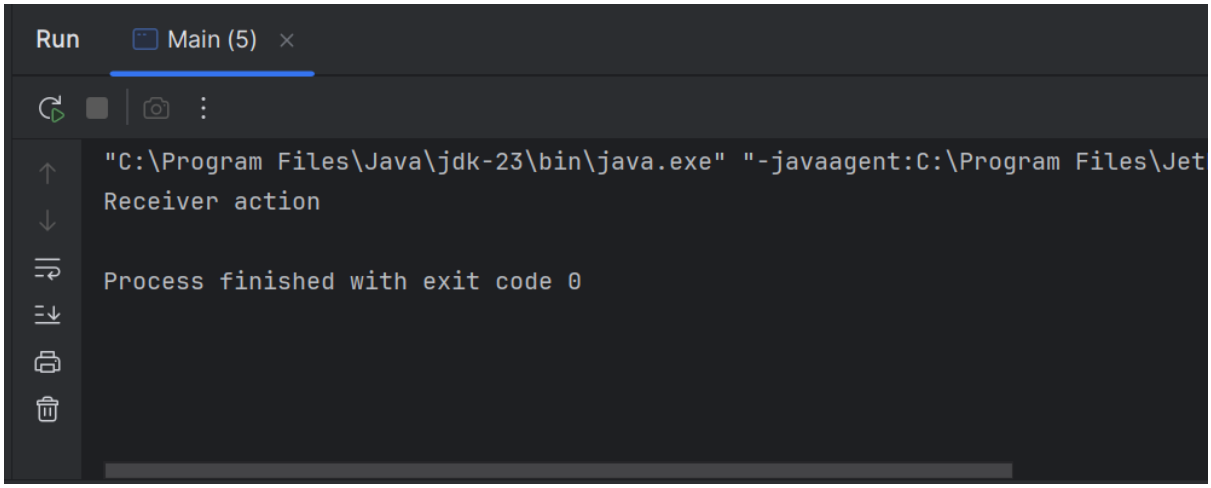
Exercise 9: Implementing the Command Pattern

```
interface Command {
    void execute();
}


class Receiver {
    public void action() {
        System.out.println("Receiver action");
    }
}


class ConcreteCommand implements Command {
    private Receiver receiver;


    public ConcreteCommand(Receiver receiver) {
```

```
        this.receiver = receiver;
    }

    @Override
    public void execute() {
        receiver.action();
    }
}

class Invoker {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void executeCommand() {
        command.execute();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Receiver receiver = new Receiver();
        Command command = new ConcreteCommand(receiver);
        Invoker invoker = new Invoker();

        invoker.setCommand(command);
        invoker.executeCommand();
```

```
    }
}
```

OUTPUT:



Exercise 10: Implementing the MVC Pattern

```
// Model
class Student {
    private String name;
    private int rollNo;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getRollNo() {
        return rollNo;
    }
```

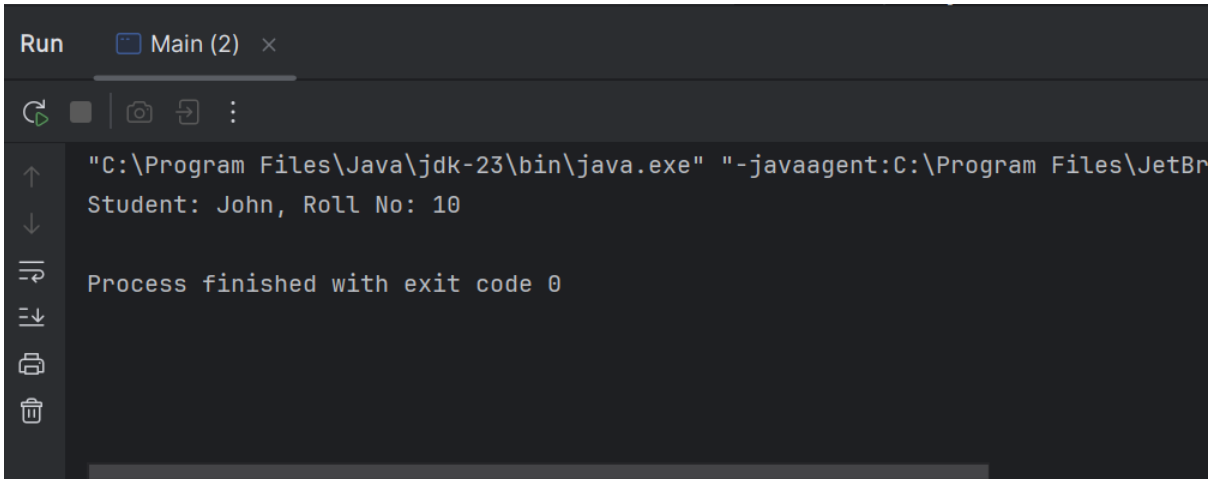
Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }
}

// View
class StudentView {
    public void printStudentDetails(String name, int rollNo) {
        System.out.println("Student: " + name + ", Roll No: " + rollNo);
    }
}

// Controller
class StudentController {
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    public void updateView() {
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
        Student model = new Student();
        model.setName("John");
        model.setRollNo(10);

        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);
        controller.updateView();
    }
}
```

OUTPUT:



Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

Exercise 11: Implementing Dependency Injection

```
interface Service {
   void serve();
}

class ServiceA implements Service {
   @Override
   public void serve() {
      System.out.println("ServiceA serving");
   }
}

class Client {
   private Service service;

   public Client(Service service) {
      this.service = service;
   }

   public void doSomething() {
      service.serve();
   }
}

// Usage
public class Main {
   public static void main(String[] args) {
      Service service = new ServiceA();
      Client client = new Client(service);
      client.doSomething();
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: naveenpr4/superset.git

```
    }
}
```

OUTPUT: