

Week 1:

Exercise 1: Implementing the Singleton Pattern

Code Implementation:

i. Singleton.java

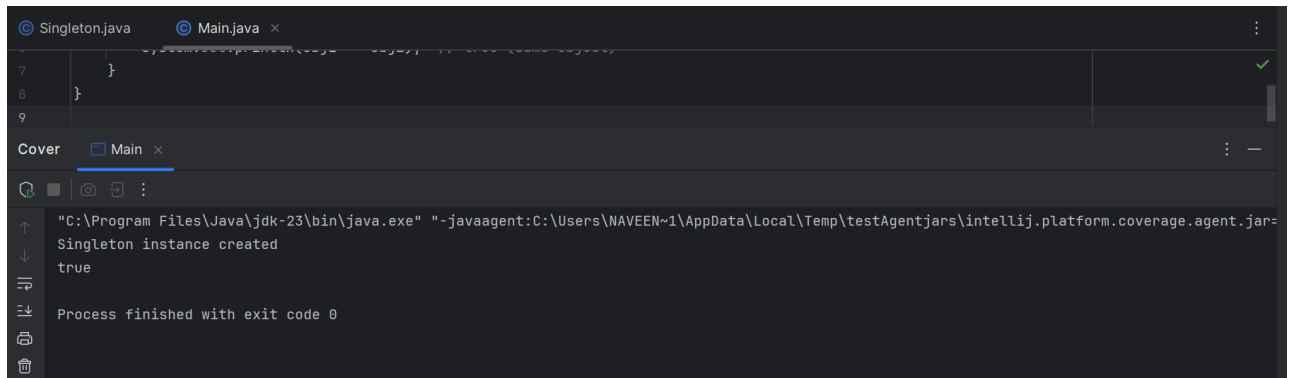
```
➤ public class Singleton {  
    private static Singleton instance;  
  
    // Private constructor  
    private Singleton() {  
        System.out.println("Singleton instance created");  
    }  
  
    // Public method to provide access  
    public static Singleton getInstance() {  
        if (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: [navveenpr4/superset.git](https://github.com/navveenpr4/superset.git)

OUTPUT:



```
7      }
8  }
9
Cover  Main x
"\"C:\Program Files\Java\jdk-23\bin\java.exe\" \"-javaagent:C:\Users\NAVEEN~1\AppData\Local\Temp\testAgentjars\intelliplatform.coverage.agent.jar=
Singleton instance created
true
Process finished with exit code 0
```

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](https://github.com/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");

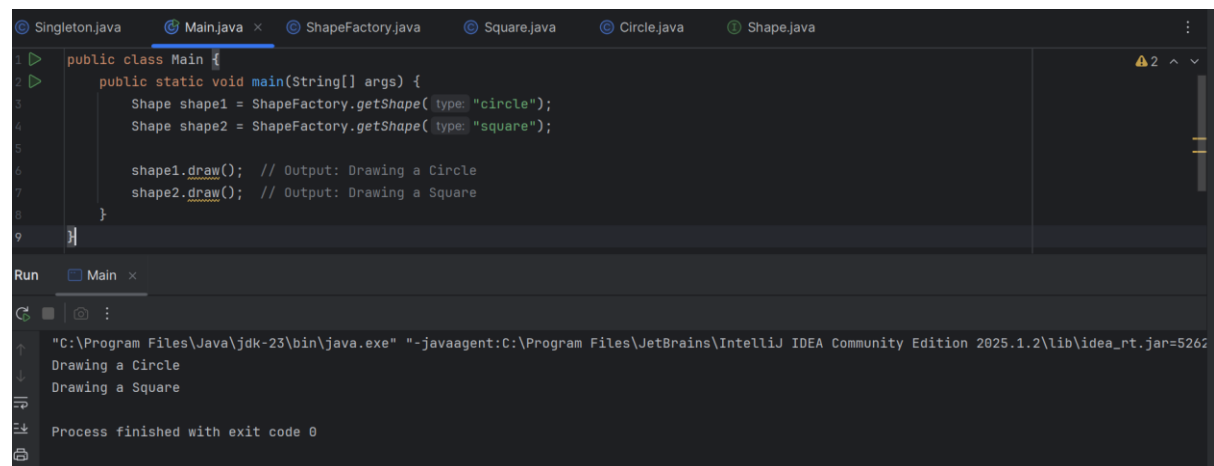

        shape1.draw();

        shape2.draw();

    }

}
```

OUTPUT:

The screenshot shows an IDE window with several tabs: Singleton.java, Main.java (active), ShapeFactory.java, Square.java, Circle.java, and Shape.java. The Main.java file contains the following code:

```
1 public class Main {
2     public static void main(String[] args) {
3         Shape shape1 = ShapeFactory.getShape("circle");
4         Shape shape2 = ShapeFactory.getShape("square");
5
6         shape1.draw(); // Output: Drawing a Circle
7         shape2.draw(); // Output: Drawing a Square
8     }
9 }
```

Below the code editor, the 'Run' tab is active, showing the command used to execute the program: `"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=5262"`. The output of the program is displayed below the command: `Drawing a Circle` and `Drawing a Square`. At the bottom, it states `Process finished with exit code 0`.

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");  
    }  
}
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/naveenpr4/superset.git)

```
}
```

```
@Override
```

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

```
@Override
```

```
public Product getResult() {  
    return product;  
}  
}
```

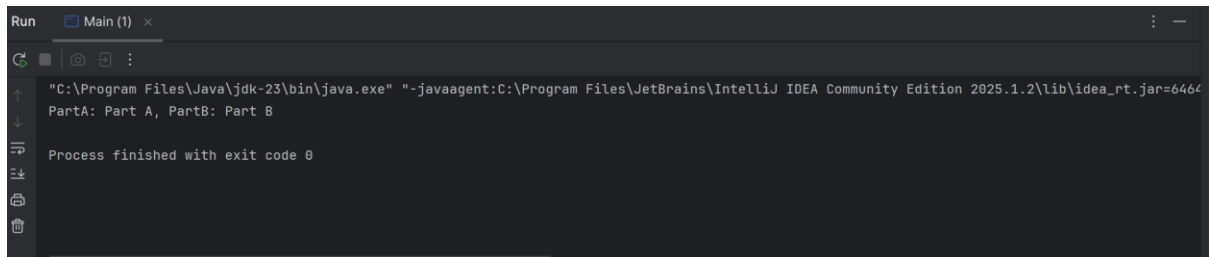
```
class Director {  
    public void construct(Builder builder) {  
        builder.buildPartA();  
        builder.buildPartB();  
    }  
}
```

```
// 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](https://github.com/naveenpr4/superset.git)

output:



```
Run Main (1) x
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=6464"
PartA: Part A, PartB: Part B
Process finished with exit code 0
```

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: [navpr4/superset.git](https://github.com/navpr4/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:

A screenshot of an IDE's Run console window. The window title is "Run" with a sub-tab "AdapterTest". The console shows the execution of a Java program. The first line of output is the full command used to run the program: "C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA...". The second line of output is "Specific Request Executed". The third line of output is "Process finished with exit code 0". The console has a dark background with light-colored text. On the left side of the console, there are several icons for running, debugging, and other IDE functions.

Exercise 5: Implementing the Decorator Pattern

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

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/naveenpr4/superset.git)

```
}  
}
```

```
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](https://github.com/naveenpr4/superset.git)

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

OUTPUT:



The screenshot shows the 'Run' window of IntelliJ IDEA. The title bar says 'Run' and 'Main (3)'. The output console displays the following text:
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
ConcreteComponent operation
Added behavior from ConcreteDecoratorA
Process finished with exit code 0
The breadcrumb at the bottom of the window reads: 'gnizant > Singleton Principle > src > Main.java > ConcreteComponent > operation'.

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](https://github.com/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:



```
Run Main (4) ×  
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA\bin\idea_rt.jar=12174:C:\Program Files\Java\jdk-23\bin" RealSubject request  
Process finished with exit code 0
```

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](https://github.com/naveenpr4/superset)

```
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);  
        }  
    }  
}
```

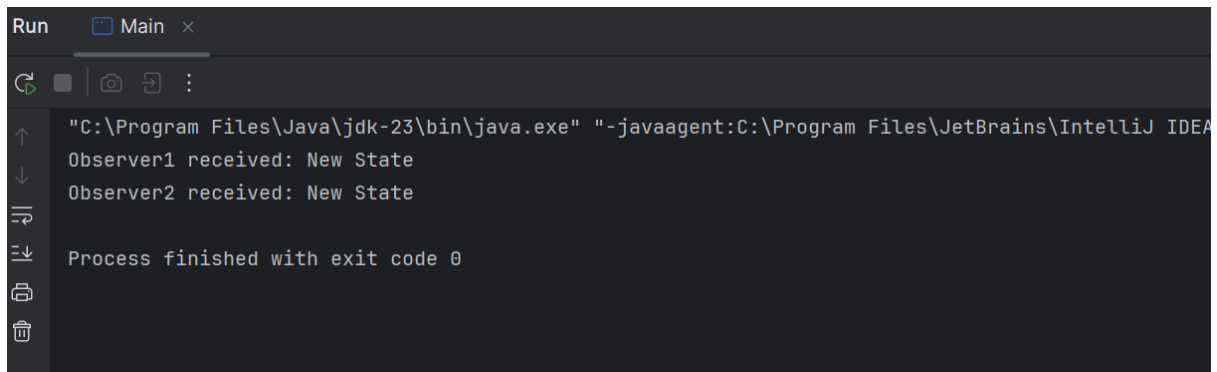
Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/naveenpr4/superset.git)

```
    }  
}  
}
```

// 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:

A screenshot of a Java IDE's 'Run' window. The window title is 'Run' with a sub-tab 'Main'. The output text is as follows:
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Observer1 received: New State
Observer2 received: New State
Process finished with exit code 0
The window includes standard IDE icons for running, debugging, and viewing logs.

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: [navpr4/superset.git](https://github.com/navpr4/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](https://github.com/naveenpr4/superset.git)

```
        context.executeStrategy();

        context.setStrategy(new ConcreteStrategyB());

        context.executeStrategy();
    }
}
```

OUTPUT:



```
Run  Main (1) x
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\
Strategy A executed
Strategy B executed
Process finished with exit code 0
```

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) {
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/navpr4/superset)

```
        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();
    }
}

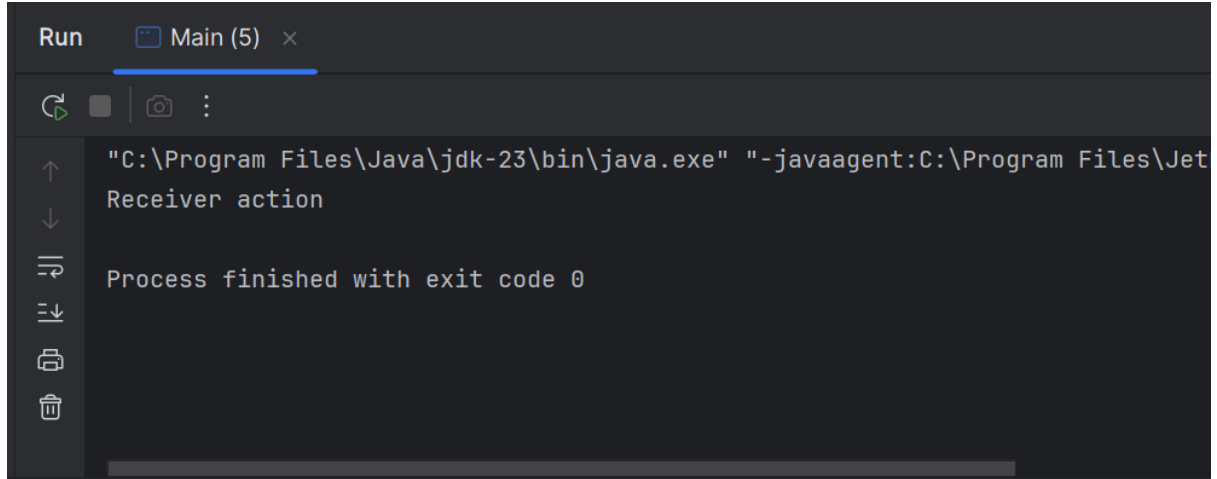
// Usage
public class Main {
    public static void main(String[] args) {
        Receiver receiver = new Receiver();
        Command command = new ConcreteCommand(receiver);
        Invoker invoker = new Invoker();

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

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [navpr4/superset.git](https://github.com/navpr4/superset.git)

```
}  
}
```

OUTPUT:

A screenshot of an IDE's Run console. The window title is 'Run' with a sub-tab 'Main (5)'. The console shows the command: "C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Jet" followed by "Receiver action". Below the command, it says "Process finished with exit code 0". On the left side of the console, there is a vertical toolbar with icons for running, stepping through, and other debugging actions.

```
Run   Main (5) ×  
↻   ■   📷   ⋮  
↑   "C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\Jet  
↓   Receiver action  
⏮   Process finished with exit code 0  
⏭  
🖨  
🗑
```

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;  
    }  
}
```

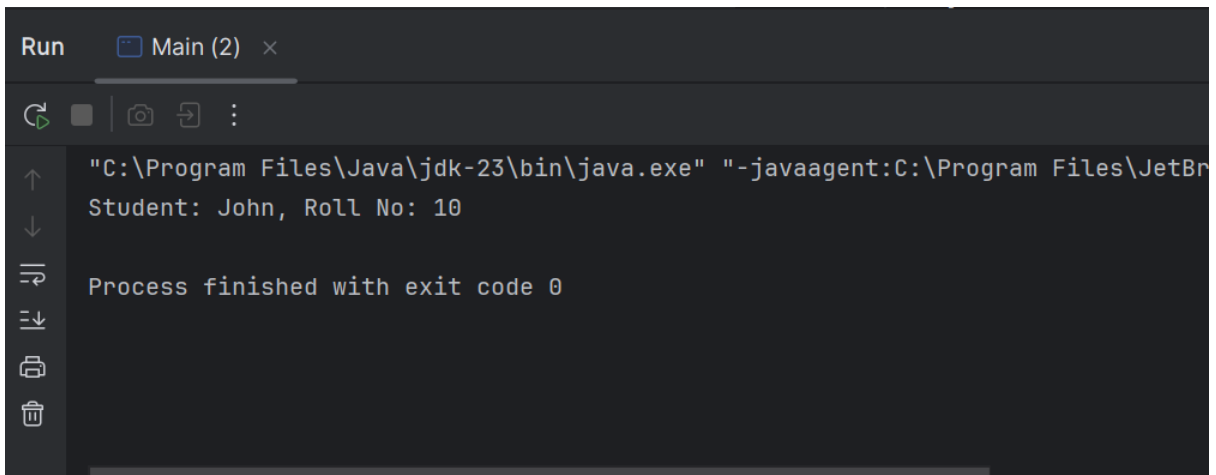

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/naveenpr4/superset.git)

```
    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](https://github.com/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:



```
Run Main (2) ×  
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBr  
Student: John, Roll No: 10  
Process finished with exit code 0
```

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();
```

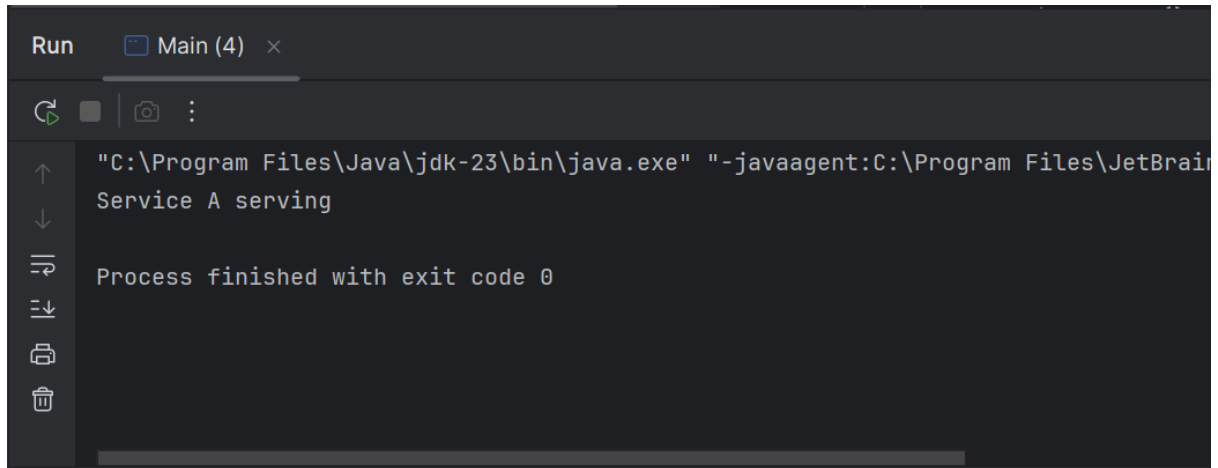
```
        Client client = new Client(service);
```

```
        client.doSomething();
```

Name: Naveen Gowda P R Superset ID: 6418880 Mail:navpr7@gmail.com
GitHub Repo: [naveenpr4/superset.git](https://github.com/naveenpr4/superset.git)

```
}  
}
```

OUTPUT:



The screenshot shows an IDE's Run console window. The title bar reads "Run" and "Main (4) x". The console output is as follows:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrain  
Service A serving  
  
Process finished with exit code 0
```

On the left side of the console, there is a vertical toolbar with icons for running, stepping through code, and other debugging actions.