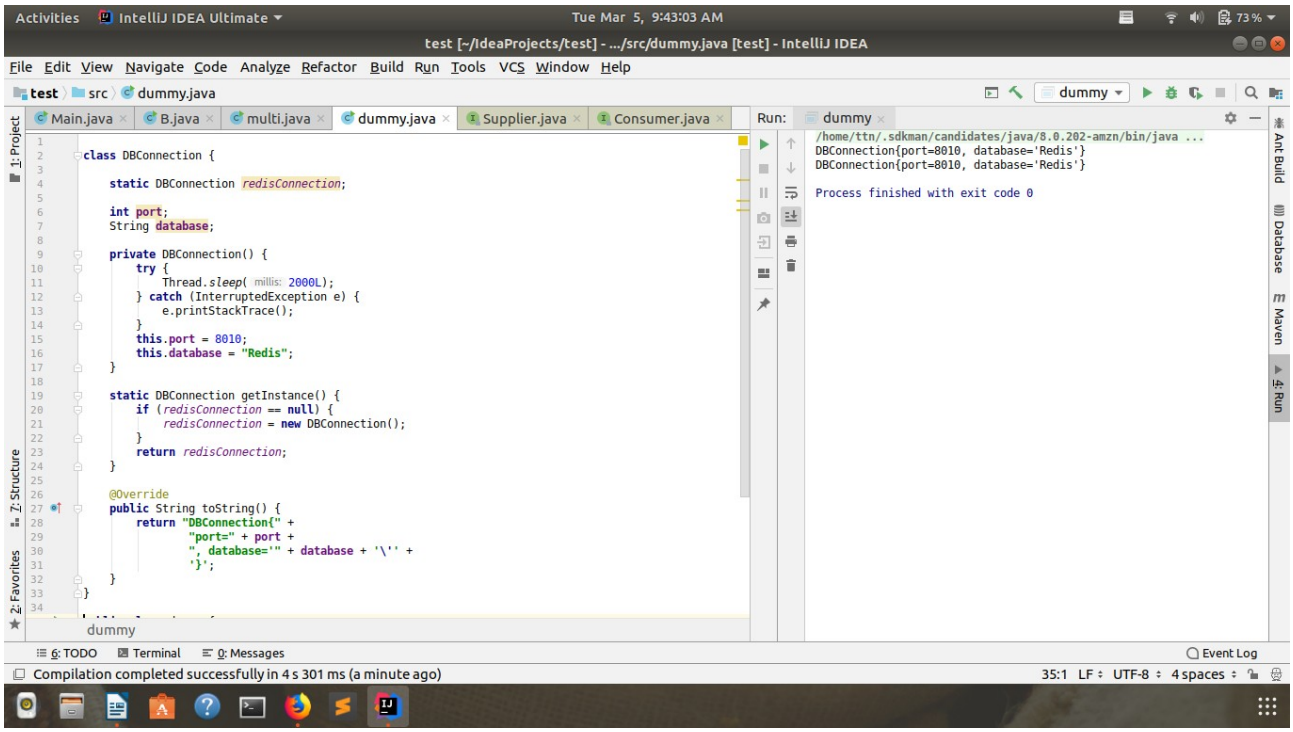
JAVA DESIGN PATTERNS

1. Implement Singleton Design Pattern on a dummy class.
2. Implement Factory Pattern to get the Polygon of differnt type.
3. Implement Abstract Factory Pattern to create cars of different categories from different countries.
4. Implement Builder pattern to create a student object with more than 6 fields.
5. Implement Bridge Design Pattern for Color and Shape such that Shape and Color can be combined together e.g BlueSquare, RedSquare, PinkTriangle etc.
6. Implement Decorator pattern to decorate the Pizza with topings.
7. Implement Composite Design Pattern to maintaing the directories of employees on the basis of departments.
8. Implement proxy design for accessing Record of a student and allow the access only to Admin.

ANSWERS

Q1.Implement Singleton Design Pattern on a dummy class.

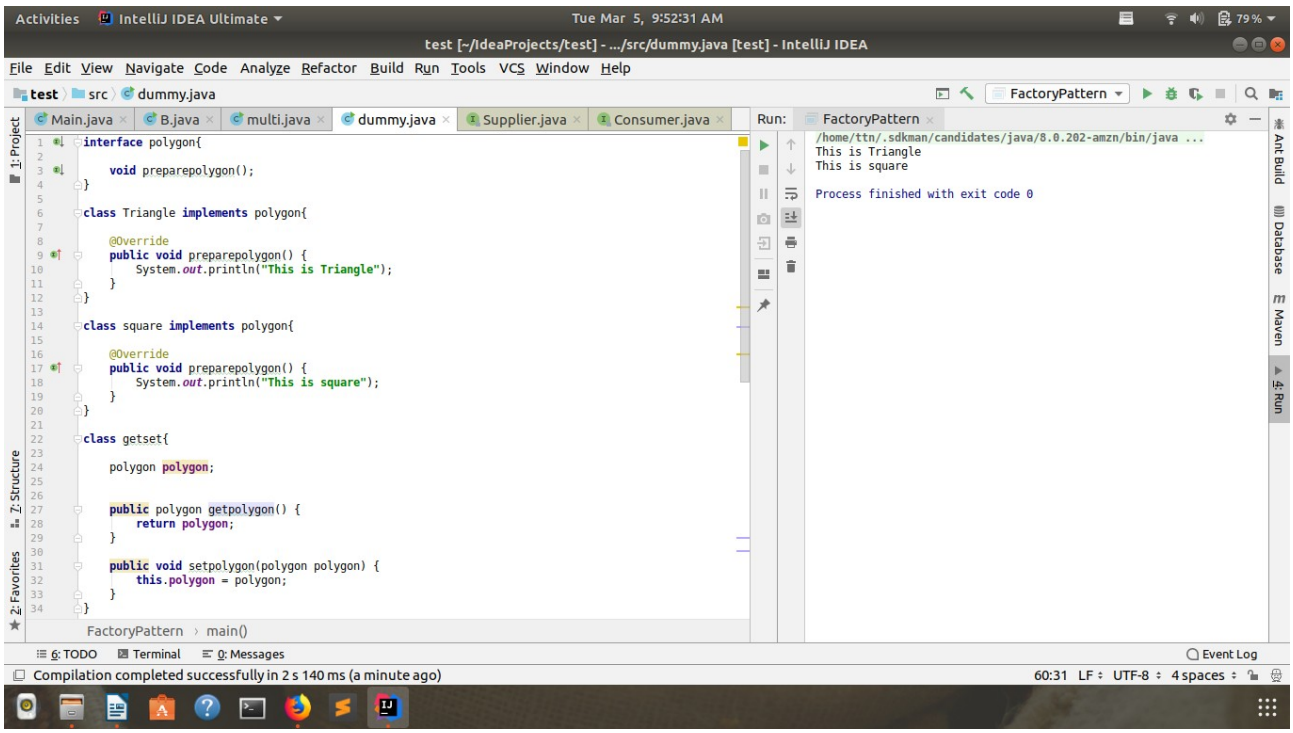
ANS.




```
class DBConnection {
    static DBConnection redisConnection;
    int port;
    String database;
    private DBConnection() {
        try {
            Thread.sleep(2000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }
        this.port = 8010;
        this.database = "Redis";
    }
    static DBConnection getInstance() {
        if (redisConnection == null) {
            redisConnection = new DBConnection();
        }
        return redisConnection;
    }
    @Override
    public String toString() {
        return "DBConnection{" +
                "port=" + port +
                ", database='" + database + '\'' +
                '}';
    }
}
public class dummy {
    public static void main(String[] args) {
        System.out.println(DBConnection.getInstance());
        System.out.println(DBConnection.getInstance());
    }
}
```

Q2.Implement Factory Pattern to get the Polygon of differnt type.
ANS.

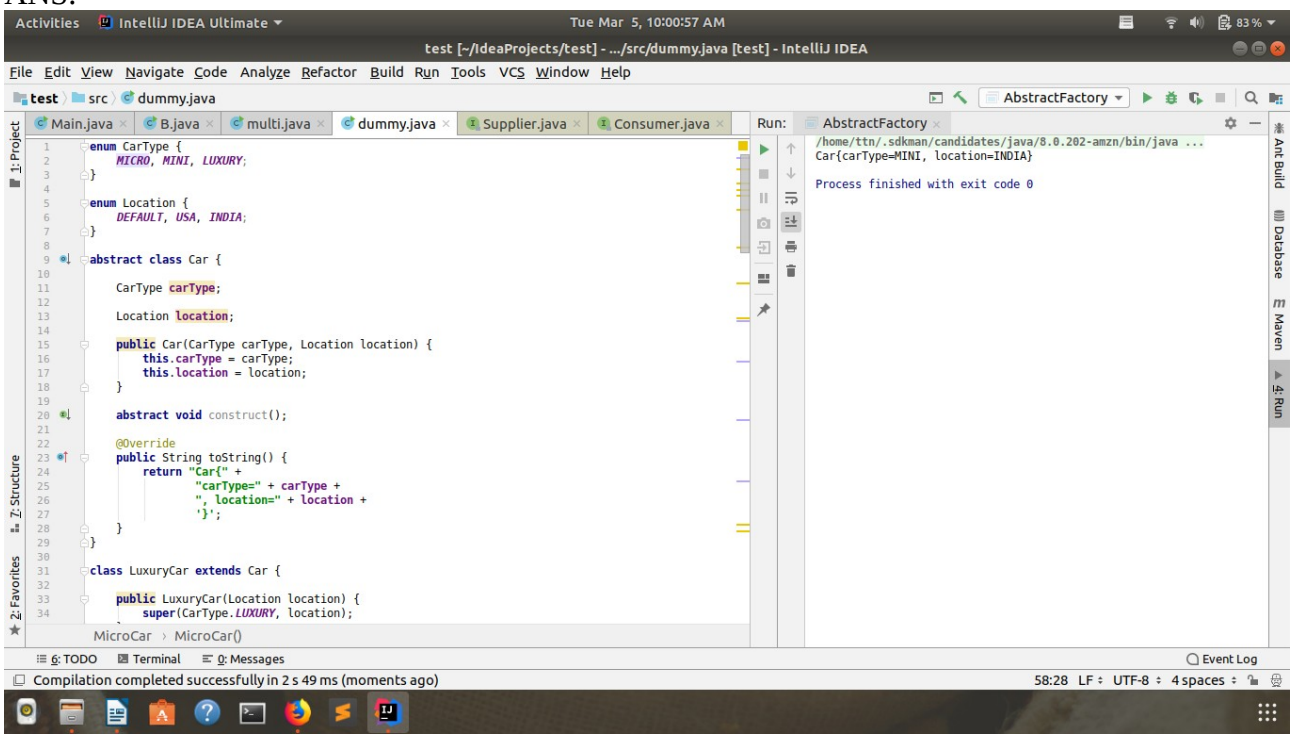



```
interface polygon{
    void preparepolygon();
}
class Triangle implements polygon{
    @Override
    public void preparepolygon() {
        System.out.println("This is Triangle");
    }
}
class square implements polygon{
    @Override
    public void preparepolygon() {
```

```
            System.out.println("This is square");
        }
}
class getset{
        polygon polygon;
        public polygon getpolygon() {
            return polygon;
        }
        public void setpolygon(polygon polygon) {
            this.polygon = polygon;
        }
}
class getsetFactory{
        static getset getgetsetObject(String name){
            getset getset= new getset();
            switch (name){
                case "getsetWithTriangle":
                    getset.setpolygon(new Triangle());
                    break;
                case "getsetWithsquare" :
                    getset.setpolygon(new square());
                    break;
            }
            return getset;
        }
}
class FactoryPattern {
        public static void main(String[] args) {
            getset getsetWithTriangle = getsetFactory.getgetsetObject("getsetWithTriangle");
            getsetWithTriangle.getpolygon().preparepolygon();
            getset getsetWithsquare = getsetFactory.getgetsetObject("getsetWithsquare");
            getsetWithsquare.getpolygon().preparepolygon();
        }
}
```

Q3.Implement Abstract Factory Pattern to create cars of different categories from different countries.

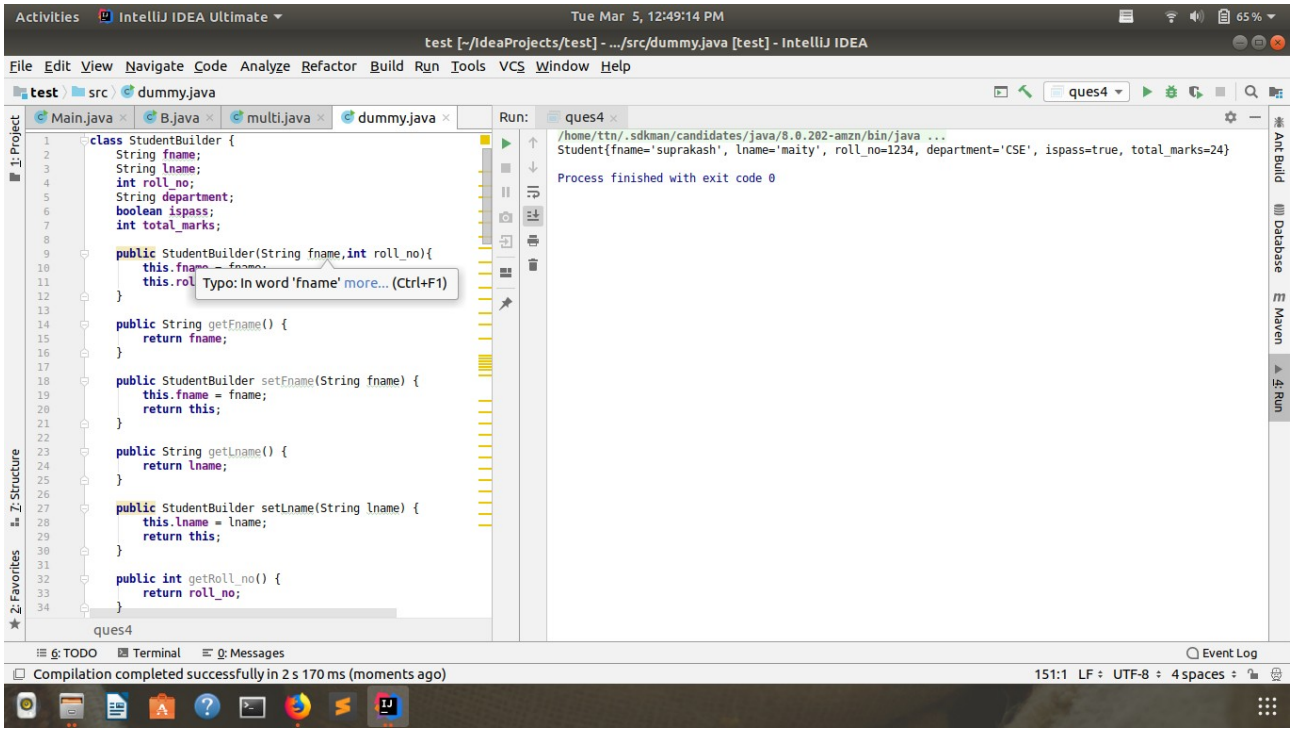
ANS.



```
enum CarType {
     MICRO, MINI, LUXURY;
}
enum Location {
     DEFAULT, USA, INDIA;
}
abstract class Car {
     CarType carType;
     Location location;
     public Car(CarType carType, Location location) {
          this.carType = carType;
          this.location = location;
     }
     abstract void construct();
     @Override
     public String toString() {
          return "Car{" +
                    "carType=" + carType +
                    ", location=" + location +
                    '}';
     }
}
class LuxuryCar extends Car {
     public LuxuryCar(Location location) {
          super(CarType.LUXURY, location);
     }
     @Override
     void construct() {
          System.out.println("connecting to Luxury Car");
     }
}
class MiniCar extends Car {
     public MiniCar(Location location) {
          super(CarType.MINI, location);
     }
     @Override
     void construct() {
          System.out.println("connecting to Mini Car");
     }
}
class MicroCar extends Car {
     public MicroCar(Location location) {
          super(CarType.MICRO, location);
     }
     @Override
     void construct() {
          System.out.println("connecting to Micro Car");
     }
}
class IndianCarFactory {
     static Car buildCar(CarType carType) {
          Car car = null;
          switch (carType) {
               case MICRO:
                    car = new MicroCar(Location.INDIA);
                    break;
               case MINI:
                    car = new MiniCar(Location.INDIA);
                    break;
               case LUXURY:
                    car = new LuxuryCar(Location.INDIA);
                    break;
          }
          return car;
     }
```

```
}
class DefaultCarFactory {
     static Car buildCar(CarType carType) {
          Car car = null;
          switch (carType) {
               case MICRO:
                    car = new MicroCar(Location.DEFAULT);
                    break;
               case MINI:
                    car = new MiniCar(Location.DEFAULT);
                    break;
               case LUXURY:
                    car = new LuxuryCar(Location.DEFAULT);
                    break;
          }
          return car;
     }
}
class USACarFactory {
     static Car buildCar(CarType carType) {
          Car car = null;
          switch (carType) {
               case MICRO:
                    car = new MicroCar(Location.USA);
                    break;
               case MINI:
                    car = new MiniCar(Location.USA);
                    break;
               case LUXURY:
                    car = new LuxuryCar(Location.USA);
                    break;
          }
          return car;
     }
}
class CarFactory {
     Car car = null;
     static Car buildCar(CarType carType, Location location) {
          Car car = null;
          switch (location) {
               case INDIA:
                    car = IndianCarFactory.buildCar(carType);
                    break;
               case USA:
                    car = USACarFactory.buildCar(carType);
                    break;
               case DEFAULT:
                    car = DefaultCarFactory.buildCar(carType);
                    break;
          }
          return car;
     }
}
class AbstractFactory {
     public static void main(String[] args) {
          System.out.println(
                    CarFactory.buildCar(CarType.MINI, Location.INDIA)
          );
     }
}
```

Q4.Implement Builder pattern to create a student object with more than 6 fields.
ANS.

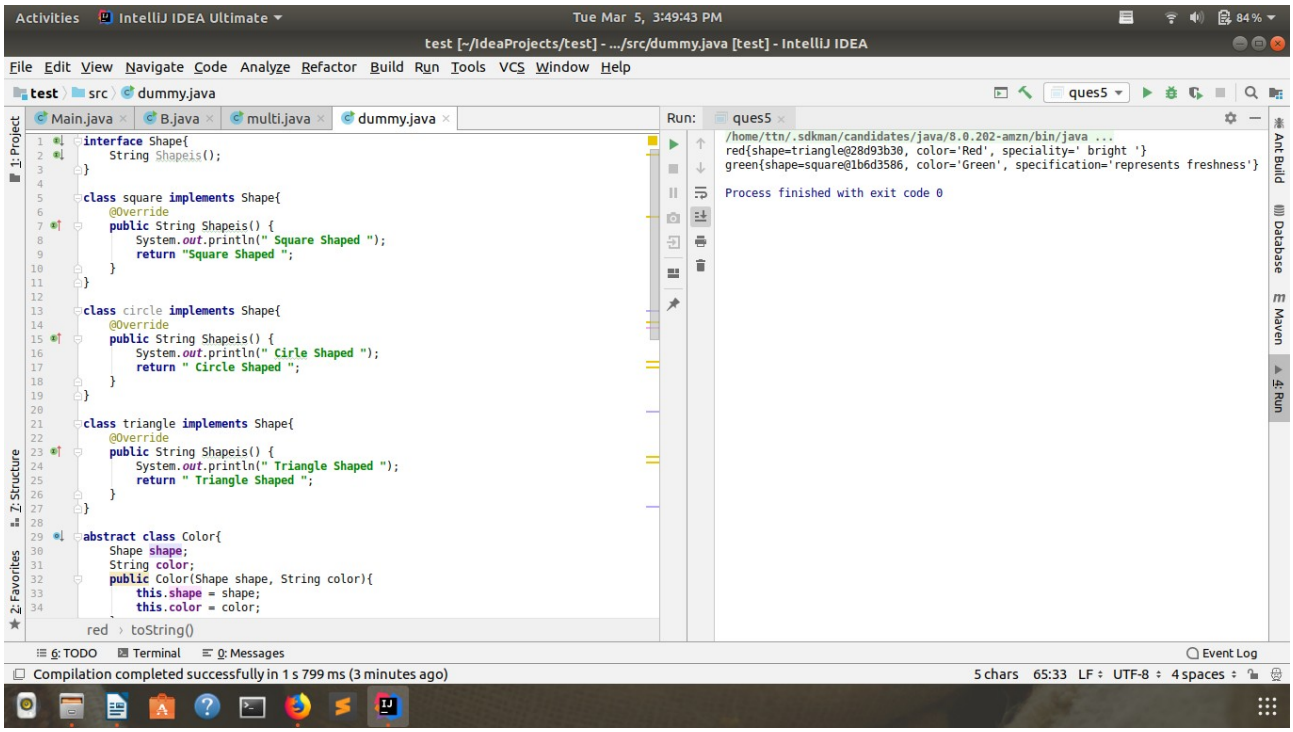



```java
class StudentBuilder {
     String fname;
     String lname;
     int roll_no;
     String department;
     boolean ispass;
     int total_marks;
     public StudentBuilder(String fname,int roll_no){
          this.fname = fname;
          this.roll_no = roll_no;
     }
     public String getFname() {
          return fname;
     }
     public StudentBuilder setFname(String fname) {
          this.fname = fname;
          return this;
     }
     public String getLname() {
          return lname;
     }
     public StudentBuilder setLname(String lname) {
          this.lname = lname;
          return this;
     }
     public int getRoll_no() {
          return roll_no;
     }
     public StudentBuilder setRoll_no(int roll_no) {
          this.roll_no = roll_no;
          return this;
     }
     public String getDepartment() {
          return department;
     }
```

```
    public StudentBuilder setDepartment(String department) {
        this.department = department;
        return this;
    }
    public boolean isIspass() {
        return ispass;
    }
    public StudentBuilder wasIspass(boolean ispass) {
        this.ispass = ispass;
        return this;
    }
    public int getTotal_marks() {
        return total_marks;
    }
    public StudentBuilder setTotal_marks(int total_marks) {
        this.total_marks = total_marks;
        return this;
    }
    public Student build(){
        return new Student(this);
    }
}
class Student{
    String fname;
    String lname;
    int roll_no;
    String department;
    boolean ispass;
    int total_marks;
    public Student(StudentBuilder sb){
        this.fname = sb.fname;
        this.lname = sb.lname;
        this.roll_no = sb.roll_no;
        this.department = sb.department;
        this.ispass = sb.ispass;
        this.total_marks = sb.total_marks;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public int getRoll_no() {
        return roll_no;
    }
    public void setRoll_no(int roll_no) {
        this.roll_no = roll_no;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
    public boolean isIspass() {
        return ispass;
    }
    }
```

```
    public void setIspass(boolean ispass) {
        this.ispass = ispass;
    }
    public int getTotal_marks() {
        return total_marks;
    }
    public void setTotal_marks(int total_marks) {
        this.total_marks = total_marks;
    }
    @Override
    public String toString() {
        return "Student{" +
                "fname='" + fname + '\'' +
                ", lname='" + lname + '\'' +
                ", roll_no=" + roll_no +
                ", department='" + department + '\'' +
                ", ispass=" + ispass +
                ", total_marks=" + total_marks +
                '}';
    }
}
class ques4 {
    public static void main(String[] args) {
        Student s = new StudentBuilder("suprakash",1234).
                setLname("maity").
                setTotal_marks(24).
                setDepartment("CSE").
                wasIspass(true).build();
        System.out.println(s);
    }
}
```

Q5.Implement Bridge Design Pattern for Color and Shape such that Shape and Color can be combined together e.g BlueSquare, RedSquare, PinkTriangle etc.
ANS.

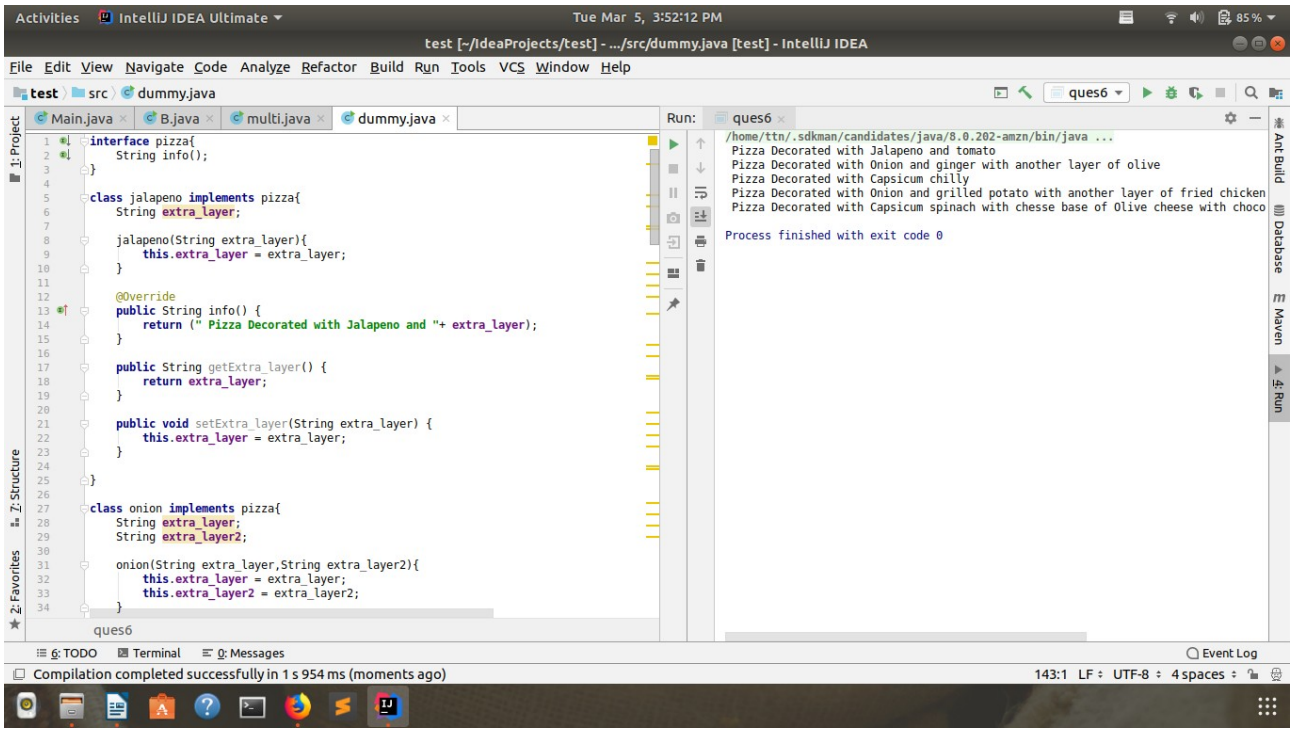



```
interface Shape{
```

```
    String Shapeis();
}
class square implements Shape{
    @Override
    public String Shapeis() {
        System.out.println(" Square Shaped ");
        return "Square Shaped ";
    }
}
class circle implements Shape{
    @Override
    public String Shapeis() {
        System.out.println(" Cirle Shaped ");
        return " Circle Shaped ";
    }
}
class triangle implements Shape{
    @Override
    public String Shapeis() {
        System.out.println(" Triangle Shaped ");
        return " Triangle Shaped ";
    }
}
abstract class Color{
    Shape shape;
    String color;
    public Color(Shape shape, String color){
        this.shape = shape;
        this.color = color;
    }
}
class green extends Color{
    String specification;
    public green(Shape shape, String specification) {
        super(shape, "Green");
        this.specification = specification;
    }
    @Override
    public String toString() {
        return "green{" +
                "shape=" + shape +
                ", color='" + color + '\'' +
                ", specification='" + specification + '\'' +
                '}';
    }
}
class red extends Color{
    String speciality;
    public red(Shape shape,String speciality){
        super(shape,"Red");
        this.speciality = speciality;
    }
    @Override
    public String toString() {
        return "red{" +
                "shape=" + shape +
                ", color='" + color + '\'' +
                ", speciality='" + speciality + '\'' +
                '}';
    }
}
class ques5 {
    public static void main(String[] args) {
        red r = new red(new triangle()," bright ");
        green g = new green(new square(),"represents freshness");
```

```
        System.out.println(r);
        System.out.println(g);
    }
}
```

Q6.Implement Decorator pattern to decorate the Pizza with topings.
ANS



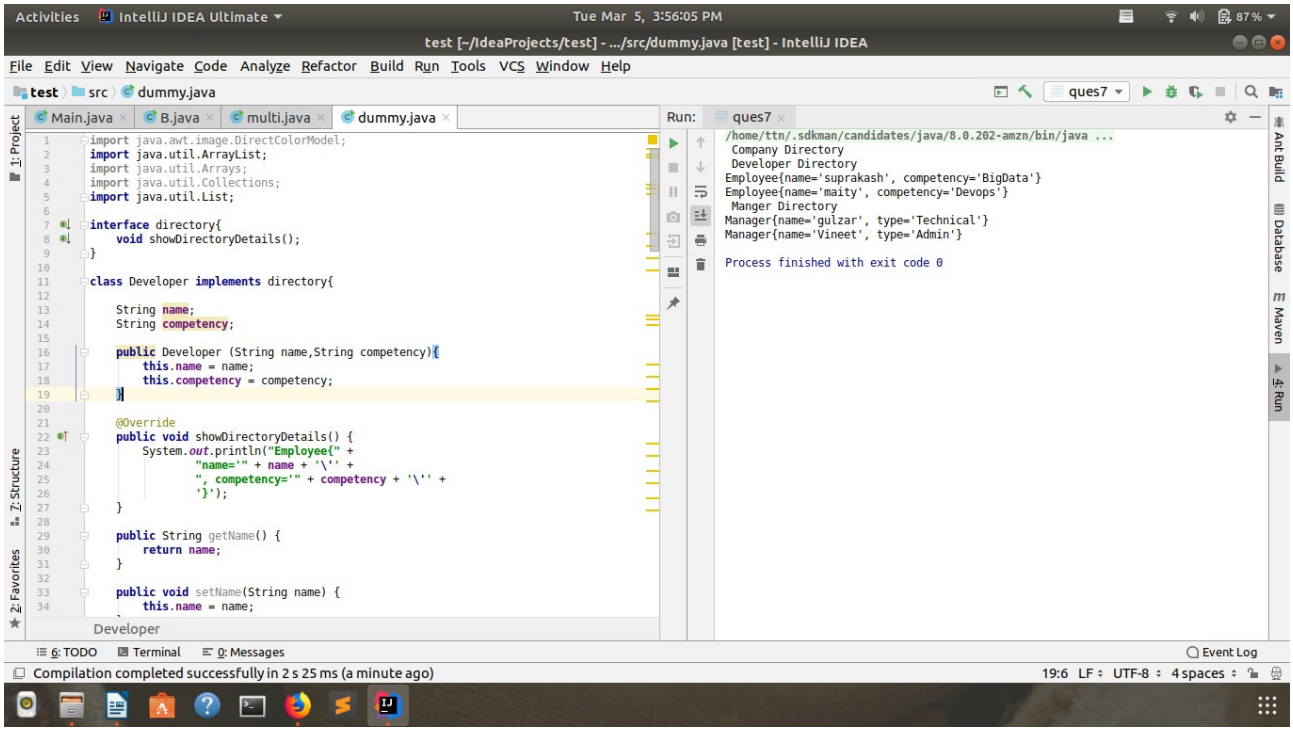

```
interface pizza{
     String info();
}
class jalapeno implements pizza{
     String extra_layer;
     jalapeno(String extra_layer){
          this.extra_layer = extra_layer;
     }
     @Override
     public String info() {
          return (" Pizza Decorated with Jalapeno and "+ extra_layer);
     }
     public String getExtra_layer() {
          return extra_layer;
     }
     public void setExtra_layer(String extra_layer) {
          this.extra_layer = extra_layer;
     }
}
class onion implements pizza{
     String extra_layer;
     String extra_layer2;
     onion(String extra_layer,String extra_layer2){
          this.extra_layer = extra_layer;
          this.extra_layer2 = extra_layer2;
     }
     @Override
     public String info() {
          return (" Pizza Decorated with Onion and "+extra_layer+" with another layer of
"+extra_layer2) ;
```

```
    }
    public String getExtra_layer() {
        return extra_layer;
    }
    public void setExtra_layer(String extra_layer) {
        this.extra_layer = extra_layer;
    }
    public String getExtra_layer2() {
        return extra_layer2;
    }
    public void setExtra_layer2(String extra_layer2) {
        this.extra_layer2 = extra_layer2;
    }
}
class capsicum implements pizza{
    String extra_layer;
    capsicum(String extra_layer){
        this.extra_layer = extra_layer;
    }
    @Override
    public String info() {
        return (" Pizza Decorated with Capsicum "+extra_layer);
    }
    public String getExtra_layer() {
        return extra_layer;
    }
    public void setExtra_layer(String extra_layer) {
        this.extra_layer = extra_layer;
    }
}
class cheeseBasePizza implements pizza{
    pizza pizzas;
    String cheesyBase;
    cheeseBasePizza(pizza pizzas,String cheesyBase){
        this.pizzas = pizzas;
        this.cheesyBase = cheesyBase;
    }
    @Override
    public String info() {
        return pizzas.info()+" with chesse base of "+cheesyBase;
    }
    public pizza getPizzas() {
        return pizzas;
    }
    public void setPizzas(pizza pizzas) {
        this.pizzas = pizzas;
    }
    public String getCheesyBase() {
        return cheesyBase;
    }
    public void setCheesyBase(String cheesyBase) {
        this.cheesyBase = cheesyBase;
    }
}
class chocolateBasePizza implements pizza{
    pizza pizzas;
    String chocoBase;
    chocolateBasePizza(pizza pizzas,String chocoBase){
        this.pizzas = pizzas;
        this.chocoBase = chocoBase;
    }
    @Override
    public String info() {
        return pizzas.info()+" with choco base of "+chocoBase;
```

```
    }
    public pizza getPizzas() {
        return pizzas;
    }
    public void setPizzas(pizza pizzas) {
        this.pizzas = pizzas;
    }
    public String getChocoBase() {
        return chocoBase;
    }
    public void setChocoBase(String chocoBase) {
        this.chocoBase = chocoBase;
    }
}
class ques6 {
    public static void main(String[] args) {
        jalapeno j = new jalapeno("tomato");
        onion o = new onion("ginger","olive");
        capsicum c = new capsicum("chilly");
        System.out.println(j.info());
        System.out.println(o.info());
        System.out.println(c.info());
        cheeseBasePizza cz = new cheeseBasePizza(new onion(
                "grilled potato","fried chicken"),
                "Margretta Cheese");
        System.out.println(cz.info());
        chocolateBasePizza ch = new chocolateBasePizza(
                new cheeseBasePizza(
                        new capsicum("spinach"), "Olive cheese"), "Almond choco");
        System.out.println(ch.info());
    }
}
```

Q7.Implement Composite Design Pattern to maintaing the directories of employees on the basis of departments.

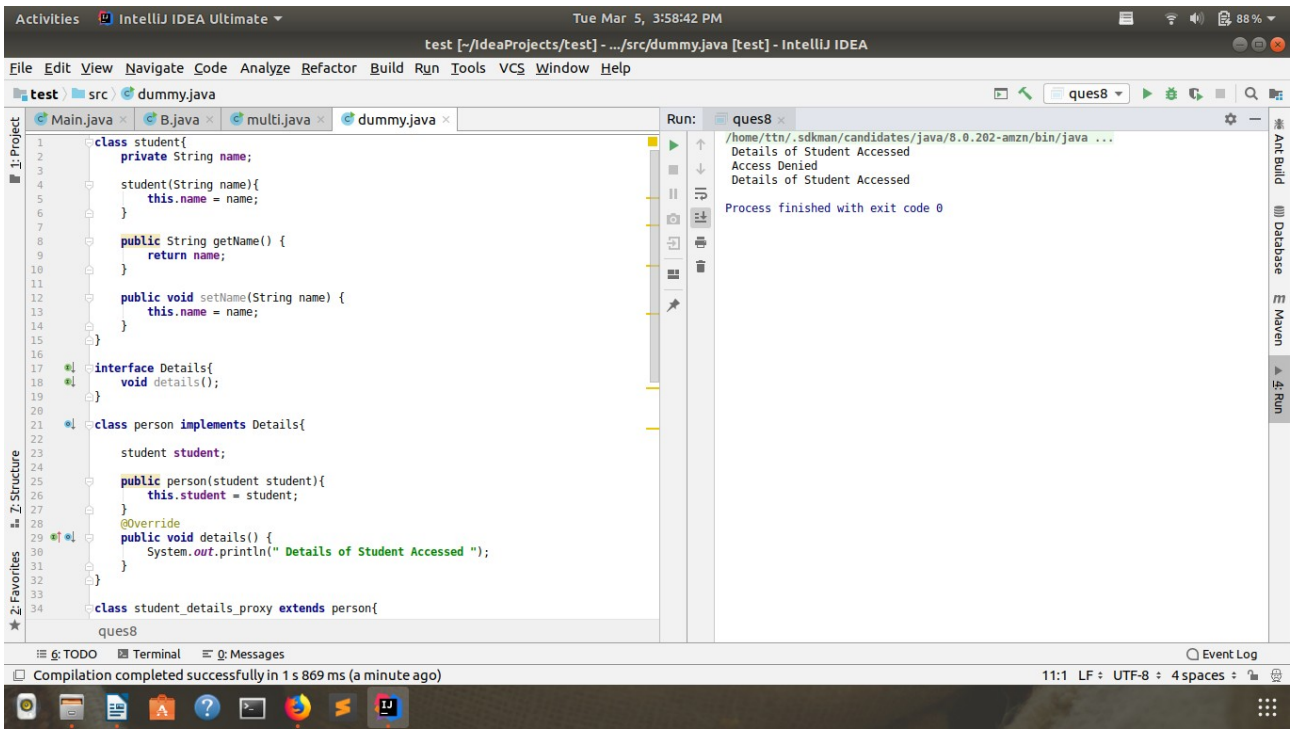
ANS.



```java
import java.awt.image.DirectColorModel;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
interface directory{
    void showDirectoryDetails();
}
class Developer implements directory{
    String name;
    String competency;
    public Developer (String name,String competency){
        this.name = name;
        this.competency = competency;
    }
    @Override
    public void showDirectoryDetails() {
        System.out.println("Employee{" +
                "name='" + name + '\'' +
                ", competency='" + competency + '\'' +
                '}');
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCompetency() {
        return competency;
    }
    public void setCompetency(String competency) {
        this.competency = competency;
    }
    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", competency='" + competency + '\'' +
                '}';
    }
}
class Manager implements directory{
    String name;
    String type;
    public Manager(String name,String type){
        this.name = name;
        this.type = type;
    }
    @Override
    public void showDirectoryDetails() {
        System.out.println("Manager{" +
                "name='" + name + '\'' +
                ", type='" + type + '\'' +
                '}');
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getType() {
        return type;
```

```
    }
    public void setType(String type) {
        this.type = type;
    }
    @Override
    public String toString() {
        return "Manager{" +
                "name='" + name + '\'' +
                ", type='" + type + '\'' +
                '}';
    }
}
class developerDirectory implements directory{
    List<directory> d = new ArrayList<directory>();
    @Override
    public void showDirectoryDetails() {
        System.out.println(" Developer Directory ");
        d.forEach(e-> e.showDirectoryDetails());
    }
}
class managerDirectory implements directory{
    List<directory> d = new ArrayList<directory>();
    @Override
    public void showDirectoryDetails() {
        System.out.println(" Manger Directory ");
        d.forEach(e-> e.showDirectoryDetails());
    }
}
class companyDirectory implements directory{
    List<directory> d = new ArrayList<directory>();
    @Override
    public void showDirectoryDetails() {
        System.out.println(" Company Directory ");
        d.forEach(e->e.showDirectoryDetails());
    }
}
class ques7 {
    public static void main(String[] args) {
        Developer dev1 = new Developer("suprakash","BigData");
        Developer dev2 = new Developer("maity","Devops");
        Manager man1 = new Manager("gulzar","Technical");
        Manager man2 = new Manager("Vineet","Admin");
        developerDirectory devDir = new developerDirectory();
        devDir.d.add(dev1);
        devDir.d.add(dev2);
        managerDirectory manDir = new managerDirectory();
        manDir.d.add(man1);
        manDir.d.add(man2);
        companyDirectory comDir = new companyDirectory();
        comDir.d.add(devDir);
        comDir.d.add(manDir);
        comDir.showDirectoryDetails();
    }
}
```

Q8.Implement proxy design for accessing Record of a student and allow the access only to Admin.
ANS.




```java
class student{
     private String name;
     student(String name){
          this.name = name;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
}
interface Details{
     void details();
}
class person implements Details{
     student student;
     public person(student student){
          this.student = student;
     }
     @Override
     public void details() {
          System.out.println(" Details of Student Accessed ");
     }
}
class student_details_proxy extends person{
     public student_details_proxy(student student) {
          super(student);
     }
     @Override
     public void details() {
          if(student.getName() =="Admin"){
               super.details();
          }
          else{
               System.out.println(" Access Denied ");
          }
     }
}
```

```
class ques8 {
    public static void main(String[] args) {
        student s = new student("Admin");
        person p = new student_details_proxy(s);
        p.details();
        student s2 = new student("suprakash");
        person p2 = new student_details_proxy(s2);
        p2.details();
        person p3 = new person(s2);
        p3.details();
    }
}
```