**Q1. Implement Singleton Design Pattern on a dummy class.**

**Solution**

**class ApplicationConfigurations**

**{**

**String dataSource;**

**int connectionPool;**

**private static ApplicationConfigurations *applicationConfigurations*;**

**private ApplicationConfigurations(String dataSource,int connectionPool)**

**{**

**this.dataSource=dataSource;**

**this.connectionPool=connectionPool;**

**}**

**public static ApplicationConfigurations getInstance()**

**{**

**if(*applicationConfigurations*==null)**

***applicationConfigurations*=new ApplicationConfigurations("reap",10);**

**return *applicationConfigurations*;**

**}**

**}**

**public class SingletonDemo {**

**public static void main(String[] args) {**

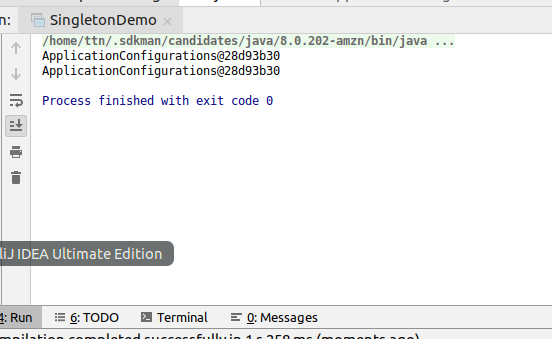
**System.*out*.println(ApplicationConfigurations.*getInstance*());**

**System.*out*.println(ApplicationConfigurations.*getInstance*());**

**}**

**}**

**Output**



**Q2. Implement Factory Pattern to get the Polygon of different type.**

**Solution**

**interface Polygon**

**{**

**public void paint();**

**}**

**class Square implements Polygon**

**{**

**@Override**

**public void paint() {**

**System.*out*.println("Painting a square");**

**}**

**}**

**class Rectangle implements Polygon**

**{**

**@Override**

**public void paint()**

**{**

**System.*out*.println("Painting a rectangle");**

**}**

**}**

**class Pentagon implements Polygon**

**{**

**@Override**

**public void paint() {**

**System.*out*.println("Painting a pentagon");**

**}**

**}**

**class PolygonFactory**

**{**

**static Polygon getPolygonInstance(int numberOfSides)**

**{**

**Polygon polygon=null;**

**switch(numberOfSides)**

**{**

**case 1:polygon=new Square();**

**break;**

**case 2:polygon=new Rectangle();**

**break;**

**case 5:polygon=new Pentagon();**

**break;**

**}**

**return polygon;**

**}**

**}**

**public class PolygonFactoryDemo {**

**public static void main(String[] args) {**

**Polygon square=PolygonFactory.*getPolygonInstance*(1);**

**square.paint();**

**Polygon rectangle=PolygonFactory.*getPolygonInstance*(2);**

**rectangle.paint();**

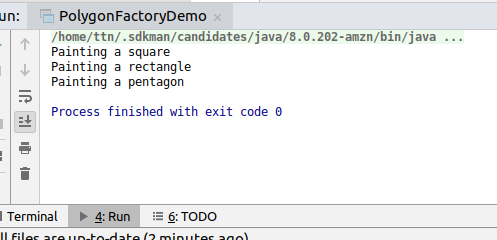
**Polygon pentagon=PolygonFactory.*getPolygonInstance*(5);**

**pentagon.paint();**

**}**

**}**

**Output**



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

**Solution**

**enum CarType {**

***MINI*, *MACRO*, *LUXURY***

**}**

**enum Location {**

***INDIA*, *CANADA*, *USA***

**}**

**abstract class Car {**

**private CarType carType;**

**private Location location;**

**private String color;**

**public Car(CarType carType, Location location, String color) {**

**this.carType = carType;**

**this.location = location;**

**this.color = color;**

**}**

**public CarType getCarType() {**

**return carType;**

**}**

**public void setCarType(CarType carType) {**

**this.carType = carType;**

**}**

**public Location getLocation() {**

**return location;**

**}**

**public void setLocation(Location location) {**

**this.location = location;**

**}**

**public String getColor() {**

**return color;**

**}**

**public void setColor(String color) {**

**this.color = color;**

**}**

**@Override**

**public String toString() {**

**return "Car{" +**

**"carType=" + carType +**

**", location=" + location +**

**", color='" + color + '\'' +**

**'}';**

**}**

**abstract void manufactureCar();**

**}**

**class MiniCar extends Car {**

**public MiniCar(Location location, String color) {**

**super(CarType.*MINI*, location, color);**

**}**

**@Override**

**void manufactureCar() {**

**System.*out*.println("Creating Mini car with color" + getColor());**

**}**

**}**

**class MacroCar extends Car {**

**public MacroCar(Location location, String color) {**

**super(CarType.*MACRO*, location, color);**

**}**

**@Override**

**void manufactureCar() {**

**System.*out*.println("Creatng Macro car with color " + getColor());**

**}**

**}**

**class LuxuryCar extends Car {**

**public LuxuryCar(Location location, String color) {**

**super(CarType.*LUXURY*, location, color);**

**}**

**@Override**

**void manufactureCar() {**

**System.*out*.println("Creating luxury car with color: " + getColor());**

**}**

**}**

**class IndianCarFactory {**

**static Car buildCar(CarType carType, String color) {**

**Car car = null;**

**switch (carType) {**

**case *MINI*:**

**car = new MiniCar(Location.*INDIA*, color);**

**break;**

**case *MACRO*:**

**car = new MacroCar(Location.*INDIA*, color);**

**break;**

**case *LUXURY*:**

**car = new LuxuryCar(Location.*INDIA*, color);**

**break;**

**}**

**return car;**

**}**

**}**

**class CanadaCarFactory {**

**static Car buildCar(CarType carType, String color) {**

**Car car = null;**

**switch (carType) {**

**case *MINI*:**

**car = new MiniCar(Location.*CANADA*, color);**

**break;**

**case *MACRO*:**

**car = new MacroCar(Location.*CANADA*, color);**

**break;**

**case *LUXURY*:**

**car = new LuxuryCar(Location.*CANADA*, color);**

**break;**

**}**

**return car;**

**}**

**}**

**class USACarFactory {**

**static Car buildCar(CarType carType, String color) {**

**Car car = null;**

**switch (carType) {**

**case *MINI*:**

**car = new MiniCar(Location.*USA*, color);**

**break;**

**case *MACRO*:**

**car = new MacroCar(Location.*USA*, color);**

**break;**

**case *LUXURY*:**

**car = new LuxuryCar(Location.*USA*, color);**

**break;**

**}**

**return car;**

**}**

**}**

**class CarFactory {**

**static Car buildCar(CarType carType, Location location, String color) {**

**Car car = null;**

**switch (location) {**

**case *INDIA*:**

**car = IndianCarFactory.*buildCar*(carType, color);**

**break;**

**case *USA*:**

**car = USACarFactory.*buildCar*(carType, color);**

**break;**

**case *CANADA*:**

**car=CanadaCarFactory.*buildCar*(carType, color);**

**break;**

**}**

**return car;**

**}**

**}**

**public class AbstractCarFactoryDemo {**

**public static void main(String[] args) {**

**Car miniIndianCar=CarFactory.*buildCar*(CarType.*MINI*,Location.*INDIA*,"red");**

**Car macroIndianCar=CarFactory.*buildCar*(CarType.*MACRO*,Location.*INDIA*,"blue");**

**Car luxuryIndanCar=CarFactory.*buildCar*(CarType.*LUXURY*,Location.*INDIA*,"black");**

**System.*out*.println("Indian cars");**

**System.*out*.println(miniIndianCar);**

**System.*out*.println(macroIndianCar);**

**System.*out*.println(luxuryIndanCar);**

**Car miniCanadianCar=CarFactory.*buildCar*(CarType.*MINI*,Location.*CANADA*,"red");**

**Car macroCanadianCar=CarFactory.*buildCar*(CarType.*MACRO*,Location.*CANADA*,"blue");**

**Car luxuryCanadianCar=CarFactory.*buildCar*(CarType.*LUXURY*,Location.*CANADA*,"black");**

**System.*out*.println("Canada Cars");**

**System.*out*.println(miniCanadianCar);**

**System.*out*.println(macroCanadianCar);**

**System.*out*.println(luxuryCanadianCar);**

**Car miniUSACar=CarFactory.*buildCar*(CarType.*MINI*,Location.*USA*,"red");**

**Car macroUSACar=CarFactory.*buildCar*(CarType.*MACRO*,Location.*USA*,"blue");**

**Car luxuryUSACar=CarFactory.*buildCar*(CarType.*LUXURY*,Location.*USA*,"black");**

**System.*out*.println("USA Cars");**

**System.*out*.println(miniUSACar);**

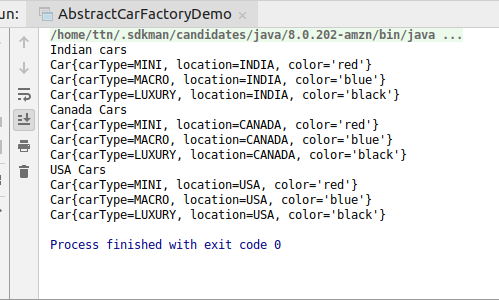
**System.*out*.println(macroUSACar);**

**System.*out*.println(luxuryUSACar);**

**}**

**}**

**Output**



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

**Solution**

**class Student**

**{**

**private int rollNumber;**

**private String name;**

**private String course;**

**private int yearOfEnrollment;**

**private int yearOfPassing;**

**private String fatherName;**

**private String motherName;**

**public Student(StudentBuilder studentBuilder)**

**{**

**this.rollNumber=studentBuilder.getRollNumber();**

**this.name=studentBuilder.getName();**

**this.course=studentBuilder.getCourse();**

**this.yearOfEnrollment=studentBuilder.getYearOfEnrollment();**

**this.yearOfPassing=studentBuilder.getYearOfPassing();**

**this.fatherName=studentBuilder.getFatherName();**

**this.motherName=studentBuilder.getMotherName();**

**}**

**public int getRollNumber() {**

**return rollNumber;**

**}**

**public void setRollNumber(int rollNumber) {**

**this.rollNumber = rollNumber;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getCourse() {**

**return course;**

**}**

**public void setCourse(String course) {**

**this.course = course;**

**}**

**public int getYearOfEnrollment() {**

**return yearOfEnrollment;**

**}**

**public void setYearOfEnrollment(int yearOfEnrollment) {**

**this.yearOfEnrollment = yearOfEnrollment;**

**}**

**public int getYearOfPassing() {**

**return yearOfPassing;**

**}**

**public void setYearOfPassing(int yearOfPassing) {**

**this.yearOfPassing = yearOfPassing;**

**}**

**public String getFatherName() {**

**return fatherName;**

**}**

**public void setFatherName(String fatherName) {**

**this.fatherName = fatherName;**

**}**

**public String getMotherName() {**

**return motherName;**

**}**

**public void setMotherName(String motherName) {**

**this.motherName = motherName;**

**}**

**@Override**

**public String toString() {**

**return "Student{" +**

**"rollNumber=" + rollNumber +**

**", name='" + name + '\'' +**

**", course='" + course + '\'' +**

**", yearOfEnrollment=" + yearOfEnrollment +**

**", yearOfPassing=" + yearOfPassing +**

**", fatherName='" + fatherName + '\'' +**

**", motherName='" + motherName + '\'' +**

**'}';**

**}**

**}**

**class StudentBuilder**

**{**

**private int rollNumber;**

**private String name;**

**private String course;**

**private int yearOfEnrollment;**

**private int yearOfPassing;**

**private String fatherName;**

**private String motherName;**

**public int getRollNumber() {**

**return rollNumber;**

**}**

**public StudentBuilder hasRollNumber(int rollNumber) {**

**this.rollNumber = rollNumber;**

**return this;**

**}**

**public String getName() {**

**return name;**

**}**

**public StudentBuilder hasName(String name) {**

**this.name = name;**

**return this;**

**}**

**public String getCourse() {**

**return course;**

**}**

**public StudentBuilder studiesInCourse(String course) {**

**this.course = course;**

**return this;**

**}**

**public int getYearOfEnrollment() {**

**return yearOfEnrollment;**

**}**

**public StudentBuilder hasEnrolledInYear(int yearOfEnrollment) {**

**this.yearOfEnrollment = yearOfEnrollment;**

**return this;**

**}**

**public int getYearOfPassing() {**

**return yearOfPassing;**

**}**

**public StudentBuilder hasPassingYear(int yearOfPassing) {**

**this.yearOfPassing = yearOfPassing;**

**return this;**

**}**

**public String getFatherName() {**

**return fatherName;**

**}**

**public StudentBuilder setFatherName(String fatherName) {**

**this.fatherName = fatherName;**

**return this;**

**}**

**public String getMotherName() {**

**return motherName;**

**}**

**public StudentBuilder setMotherName(String motherName) {**

**this.motherName = motherName;**

**return this;**

**}**

**public Student createStudent()**

**{**

**return new Student(this);**

**}**

**}**

**public class BuilderDemo {**

**public static void main(String[] args) {**

**Student student=new StudentBuilder()**

**.hasRollNumber(1)**

**.hasName("Surbhi Garg")**

**.studiesInCourse("MCA")**

**.setFatherName("Umesh Chand Garg")**

**.setMotherName("Rama Garg")**

**.hasEnrolledInYear(2016)**

**.hasPassingYear(2019)**

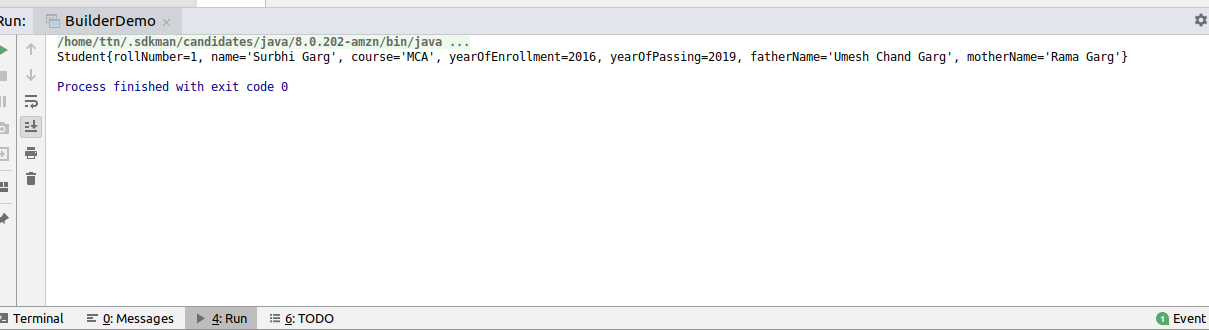
**.createStudent();**

**System.*out*.println(student);**

**}**

**}**

**Output**



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

**Solution**

**package com.design.bridge;**

***//implementor***

**interface Color {**

**public String fillColor();**

**}**

***//Concrete implementor***

**class RedColor implements Color {**

**@Override**

**public String fillColor() {**

**return "Filling red color in object";**

**}**

**}**

**class BlueColor implements Color {**

**@Override**

**public String fillColor() {**

**return "Filling blue color in object";**

**}**

**}**

**class PinkColor implements Color {**

**@Override**

**public String fillColor() {**

**return "Filling pink color in object";**

**}**

**}**

***//Abstraction***

**abstract class Shape {**

**private static int *numberOfDimensions*;**

**private Color color;**

**public Shape(int numberOfDimensions, Color color) {**

**this.*numberOfDimensions* = numberOfDimensions;**

**this.color = color;**

**}**

**public int getNumberOfDimensions() {**

**return *numberOfDimensions*;**

**}**

**public void setNumberOfDimensions(int numberOfDimensions) {**

**this.*numberOfDimensions* = numberOfDimensions;**

**}**

**public Color getColor() {**

**return color;**

**}**

**public void setColor(Color color) {**

**this.color = color;**

**}**

**}**

***//Refined Abstraction***

**class Square extends Shape {**

**private int side;**

**public Square(Color color, int side) {**

**super(1, color);**

**this.side = side;**

**}**

**@Override**

**public String toString() {**

**return "Square{" +**

**"side=" + side +**

**"color=" + getColor().fillColor() +**

**'}';**

**}**

**}**

**class Triangle extends Shape {**

**private int base, height;**

**public Triangle(Color color, int base, int height) {**

**super(2, color);**

**this.base = base;**

**this.height = height;**

**}**

**@Override**

**public String toString() {**

**return "Triangle{" +**

**"base=" + base +**

**", height=" + height +**

**"color=" + getColor().fillColor() +**

**'}';**

**}**

**}**

**public class BridgePatternDemo {**

**public static void main(String[] args) {**

**Square blueSquare = new Square(new BlueColor(), 4);**

**System.*out*.println(blueSquare);**

**Square redSquare = new Square(new RedColor(), 5);**

**System.*out*.println(redSquare);**

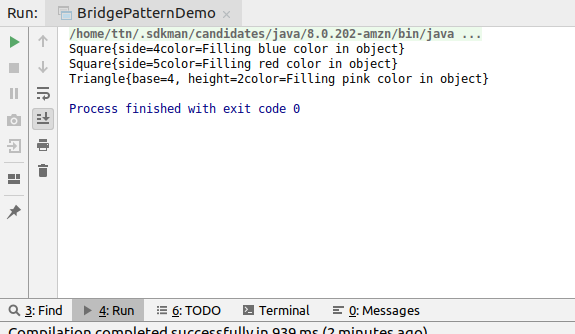
**Triangle pinkTriangle = new Triangle(new PinkColor(), 4, 2);**

**System.*out*.println(pinkTriangle);**

**}**

**}**

**Output**



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

**Solution**

**import java.util.Arrays;**

**import java.util.List;**

***/\*Assuming the case of Domino's where there are some predefined***

***pizzas on which we can have some additional toppings and also considering***

***a customizable pizza which has no toppings***

***\*/***

**interface Pizza {**

**void makePizza();**

**}**

**class CustomizablePizza implements Pizza {**

**@Override**

**public void makePizza() {**

**System.*out*.println("Making customizable pizza");**

**}**

**}**

**class PeppyPaneerPizza implements Pizza {**

**@Override**

**public void makePizza() {**

**System.*out*.println("Making peppy panner pizza");**

**}**

**}**

**abstract class PizzaDecorator implements Pizza {**

**Pizza pizza;**

**public PizzaDecorator(Pizza pizza) {**

**this.pizza = pizza;**

**}**

**}**

**class ExtraCheeseDecorator extends PizzaDecorator {**

**private String cheeseType;**

**public ExtraCheeseDecorator(Pizza pizza, String cheeseType) {**

**super(pizza);**

**this.cheeseType = cheeseType;**

**}**

**@Override**

**public void makePizza() {**

**pizza.makePizza();**

**System.*out*.println(" with extra cheese of type " + cheeseType);**

**}**

**}**

**class VeggiesDecorator extends PizzaDecorator {**

**List<String> vegetableList;**

**public VeggiesDecorator(Pizza pizza, List<String> vegetableList) {**

**super(pizza);**

**this.vegetableList = vegetableList;**

**}**

**@Override**

**public void makePizza() {**

**pizza.makePizza();**

**System.*out*.println(" with" + vegetableList);**

**}**

**}**

**public class DecoratorDemo {**

**public static void main(String[] args) {**

**Pizza capsicumAndCheesePizza = new VeggiesDecorator(**

**new ExtraCheeseDecorator(new CustomizablePizza(), "margharita")**

**, Arrays.*asList*("capsicum"));**

**capsicumAndCheesePizza.makePizza();**

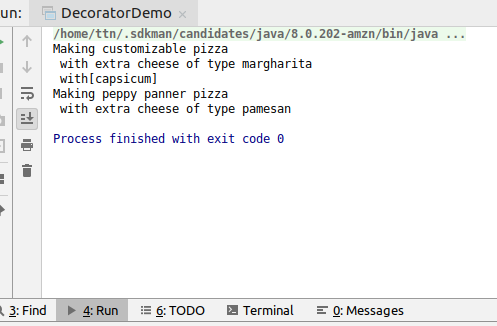
**Pizza peppyPaneerWithExtraCheese = new ExtraCheeseDecorator(new PeppyPaneerPizza(), "pamesan");**

**peppyPaneerWithExtraCheese.makePizza();**

**}**

**}**

**Output**



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

**Solution**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.List;**

***/\****

***\* Employee***

***\* |***

***\* ---------------***

***\* | |***

***\* Technical Non Technical***

***\* | |***

***\* ----------- --------------***

***\* | | | |***

***\* Surbhi Gagan Ajay Shreya***

***\****

***\* component--->Directory (All are of type directory)***

***\* Comppsite---->EmployeeDirectory,TechnicalEmployeeDirectory,Non-TechnicalEmployeeDirectory***

***\* Leaf-->TechnicalEmployee,NonTechnicalEmployee***

***\****

***\* \*/***

***//Component***

**interface Directory {**

**void showDirectoryInfo();**

**}**

***//Leaf***

**class TechnicalEmployee implements Directory {**

**private String name;**

**private String specialization;**

**public TechnicalEmployee(String name, String specialization) {**

**this.name = name;**

**this.specialization = specialization;**

**}**

**@Override**

**public void showDirectoryInfo() {**

**System.*out*.println("TechnicalEmployee{" +**

**"name='" + name + '\'' +**

**", specialization='" + specialization + '\'' +**

**'}');**

**}**

**}**

**class NonTechnicalEmployee implements Directory {**

**String name;**

**public NonTechnicalEmployee(String name) {**

**this.name = name;**

**}**

**@Override**

**public void showDirectoryInfo() {**

**System.*out*.println("NonTechnicalEmployee{" +**

**"name='" + name + '\'' +**

**'}');**

**}**

**}**

***//Composite***

**class TechnicalEmployeeDirectory implements Directory {**

**List<TechnicalEmployee> technicalEmployees;**

**public TechnicalEmployeeDirectory() {**

**technicalEmployees = new ArrayList<>();**

**}**

**public List<TechnicalEmployee> getTechnicalEmployees() {**

**return technicalEmployees;**

**}**

**public void setTechnicalEmployees(List<TechnicalEmployee> technicalEmployees) {**

**this.technicalEmployees = technicalEmployees;**

**}**

**@Override**

**public void showDirectoryInfo() {**

**technicalEmployees.forEach(e -> e.showDirectoryInfo());**

**}**

**}**

**class NonTechnicalEmployeeDirectory implements Directory {**

**List<NonTechnicalEmployee> nonTechnicalEmployees;**

**public NonTechnicalEmployeeDirectory() {**

**nonTechnicalEmployees = new ArrayList<>();**

**}**

**public List<NonTechnicalEmployee> getNonTechnicalEmployees() {**

**return nonTechnicalEmployees;**

**}**

**public void setNonTechnicalEmployees(List<NonTechnicalEmployee> nonTechnicalEmployees) {**

**this.nonTechnicalEmployees = nonTechnicalEmployees;**

**}**

**@Override**

**public void showDirectoryInfo() {**

**nonTechnicalEmployees.forEach(e -> e.showDirectoryInfo());**

**}**

**}**

**class EmployeeDirectory implements Directory {**

**List<Directory> employeeDirectories;**

**public EmployeeDirectory() {**

**employeeDirectories = new ArrayList<>();**

**}**

**public List<Directory> getEmployeeDirectories() {**

**return employeeDirectories;**

**}**

**public void setEmployeeDirectories(List<Directory> employeeDirectories) {**

**this.employeeDirectories = employeeDirectories;**

**}**

**@Override**

**public void showDirectoryInfo() {**

**employeeDirectories.forEach(e -> e.showDirectoryInfo());**

**}**

**}**

**public class CompositeDesignDemo {**

***//Client***

**public static void main(String[] args) {**

**TechnicalEmployee technicalEmployee1 = new TechnicalEmployee("Surbhi", "JVM");**

**TechnicalEmployee technicalEmployee2 = new TechnicalEmployee("Gagan", "JVM");**

**NonTechnicalEmployee nonTechnicalEmployee1 = new NonTechnicalEmployee("Ajay");**

**NonTechnicalEmployee nonTechnicalEmployee2 = new NonTechnicalEmployee("Shreya");**

**TechnicalEmployeeDirectory technicalEmployeeDirectory = new TechnicalEmployeeDirectory();**

**technicalEmployeeDirectory.getTechnicalEmployees().addAll(Arrays.*asList*(technicalEmployee1, technicalEmployee2));**

**NonTechnicalEmployeeDirectory nonTechnicalEmployeeDirectory = new NonTechnicalEmployeeDirectory();**

**nonTechnicalEmployeeDirectory.getNonTechnicalEmployees().addAll(Arrays.*asList*(nonTechnicalEmployee1, nonTechnicalEmployee2));**

**EmployeeDirectory employeeDirectory = new EmployeeDirectory();**

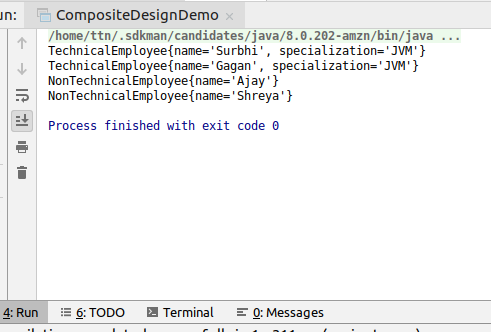
**employeeDirectory.getEmployeeDirectories().addAll(Arrays.*asList*(technicalEmployeeDirectory, nonTechnicalEmployeeDirectory));**

**employeeDirectory.showDirectoryInfo();**

**}**

**}**

**Output**



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

**Solution**

**package com.design.proxy;**

**interface Accessable**

**{**

**void accessRecord();**

**}**

**class Student implements Accessable**

**{**

**int rollNo;**

**String name;**

**public Student(int rollNo, String name) {**

**this.rollNo = rollNo;**

**this.name = name;**

**}**

**@Override**

**public void accessRecord() {**

**System.*out*.println("Record is: "+rollNo+" "+name);**

**}**

**}**

**enum UserRole{*ADMIN*,*USER*}**

**class User**

**{**

**private String name;**

**private UserRole role;**

**public User(String name,UserRole role) {**

**this.name = name;**

**this.role=role;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public UserRole getRole() {**

**return role;**

**}**

**public void setRole(UserRole role) {**

**this.role = role;**

**}**

**}**

**class StudentProxy extends Student implements Accessable**

**{**

**private User user;**

**public StudentProxy(int rollNo,String name,User user) {**

**super(rollNo,name);**

**this.user = user;**

**}**

**@Override**

**public void accessRecord() {**

**if(user.getRole()==UserRole.*ADMIN*)**

**super.accessRecord();**

**else**

**System.*out*.println("Access Denied! you're not admin");**

**}**

**}**

**public class ProxyDemo {**

**public static void main(String[] args) {**

**Accessable student1=**

**new StudentProxy(1,"Surbhi Garg",new User("Abhinav",UserRole.*ADMIN*));**

**Accessable student2=new StudentProxy(2,"AditiWalia",new User("Surbhi",UserRole.*USER*));**

**student1.accessRecord();**

**student2.accessRecord();**

**}**

**}**

**Output**

