**Exam 2 Instructions**

**OBJECT-ORIENTED PROG**

* This is a take-home exam. You can use any resources that are available for you to finish this exam, except
  + Outsourcing the exam to any person or to any third party websites
  + Copying from other students work
  + Copying direct quotes from the books or internet
* Do not lose your opportunity to learn while working on the exam. Understand the concept and write answers on your own.
* Usually, in life, we have several choices. Unfortunately, you don’t have any choice on this exam. You have to answer all the questions and each part of the problem.
* All the topics on this exam were discussed in class before week 13. So, you cannot claim that the questions are out of the syllabus!
* Refer to Microsoft Word tutorials for proper formatting
* Points will be deducted for grammatical and spelling mistakes
* No two brains think alike unless you are soulmates. Definitely your answers will not be same as other students.
* Read the code of academic integrity before you start the exam. <https://www.nwmissouri.edu/policies/academics/Academic-Integrity.pdf>
* Push your source code to GitHub and provide your GitHub link at the end of the document and in the comment section.
* Don’t use examples that already explained in class or worksheets.
* Provide the input and output screenshots for every program.

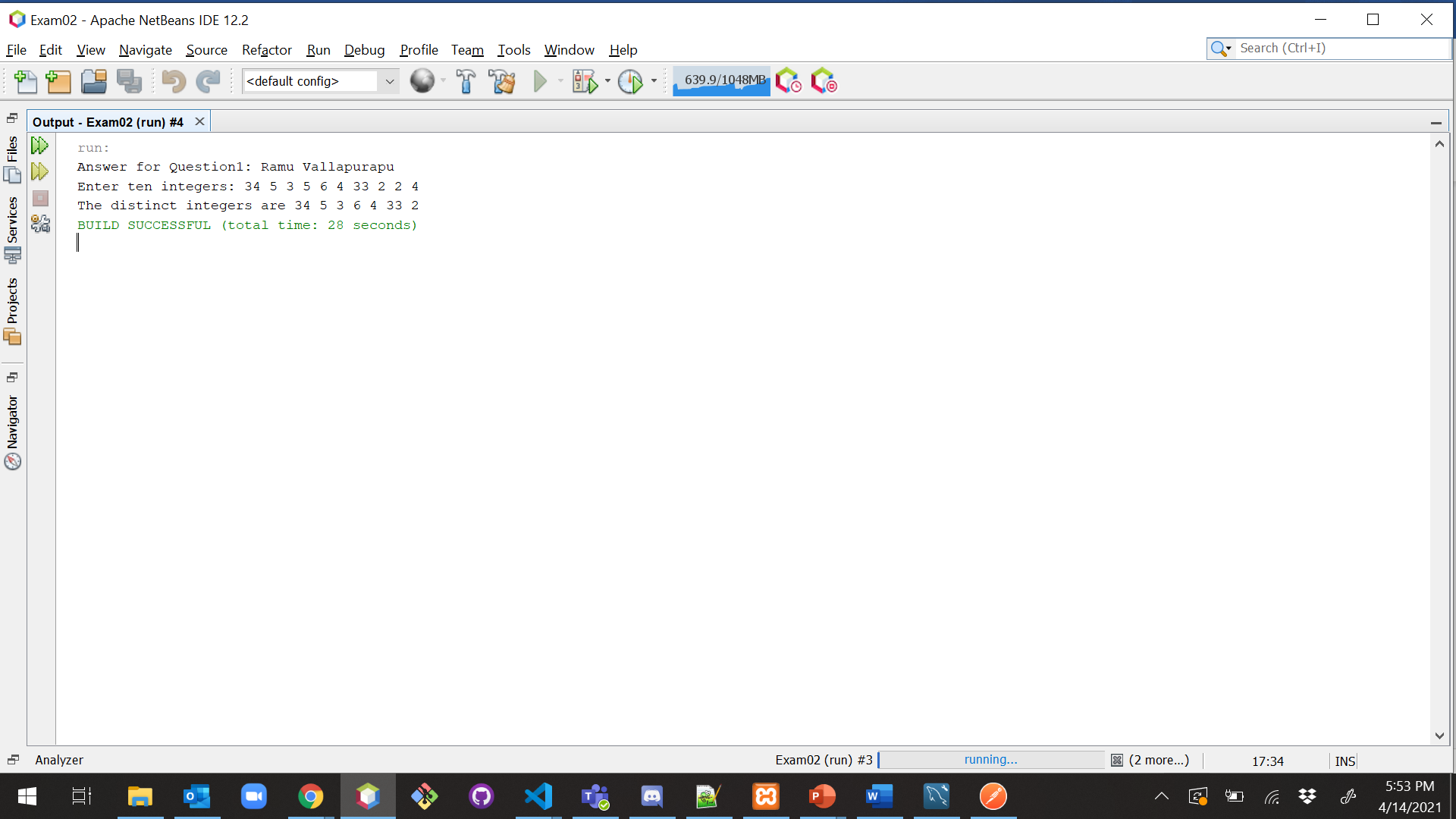
**Exam 2 OBJECT-ORIENTED PROG 01FA20 100 pts**

1. (5-Points) (1D-Array - )Write a method that removes the duplicate elements from an array list of integers using the following header:

Public static void removeDuplicate(ArrayList<Integer> list)

Write a test program that prompts the user to enter 10 integers to a list and displays the distinct integers separated by exactly one space. Provide screenshot of executable code with input and output. Here is a sample run:

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package questions1and2;  import java.util.ArrayList;  import java.util.Scanner;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class DuplicateElements01 {  public static void main(String args[]) {  System.out.println("Answer for Question1: Ramu Vallapurapu");  ArrayList<Integer> al = new ArrayList<>();  Scanner sc = new Scanner(System.in);  System.out.print("Enter ten integers: ");  for (int i = 0; i < 10; i++) {  al.add(sc.nextInt());  }  removeDuplicate(al);  }  public static void removeDuplicate(ArrayList<Integer> list) {  ArrayList<Integer> temp = new ArrayList<>();  for (Integer l : list) {  if (!temp.contains(l)) {  temp.add(l);  }  }  System.out.print("The distinct integers are ");  for (Integer i : temp) {  System.out.print(i + " ");  }  System.out.println();  }  } |



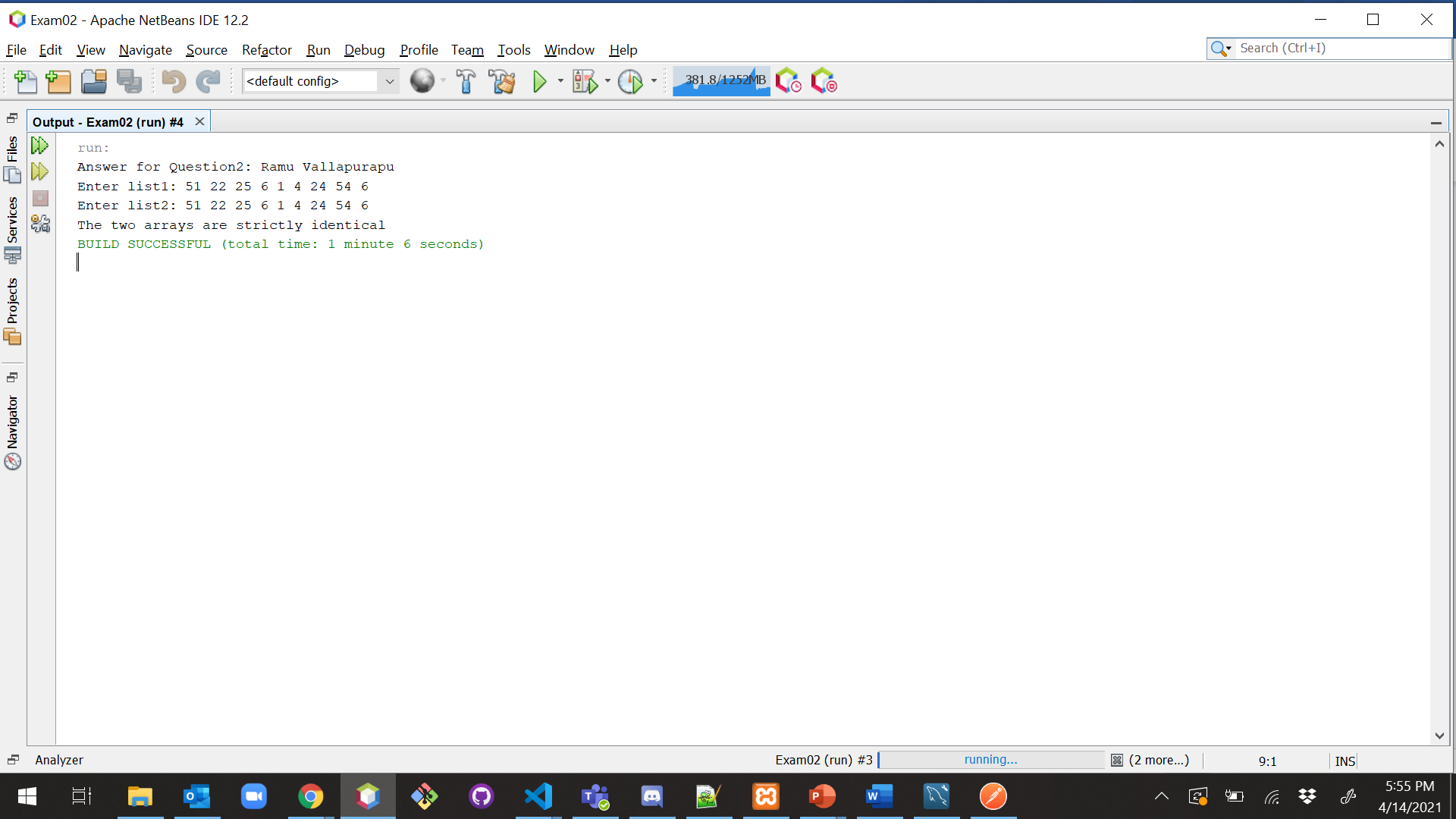
|  |
| --- |
| Enter ten integers: 34 5 3 5 6 4 33 2 2 4  The distinct integers are 34 5 3 6 4 33 2 |

1. (5-Points) (2D- Array) The two-dimensional arrays m1 and m2 are strictly identical if their corresponding elements are equal. Write a method that returns true if m1 and m2 are strictly identical, using the following header:

public static boolean equals(int[][] m1, int[][] m2)

Write a test program that prompts the user to enter two 3 \* 3 arrays of integers and displays whether the two are strictly identical. Provide screenshot of executable code with input and output. Here are the sample runs.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package exam02;  import java.util.Scanner;  /\*\*  \* @author Ramu Vallapurapu  \*/  public class TwoDimensionalArray02 {  public static void main(String[] args) {  System.out.println("Answer for Question2: Ramu Vallapurapu");  Scanner sc = new Scanner(System.in);  System.out.print("Enter list1: ");  int[][] m1 = new int[3][3];  for (int i = 0; i <= 2; i++) {  for (int j = 0; j <= 2; j++) {  m1[i][j] = sc.nextInt();  }  }  System.out.print("Enter list2: ");  int[][] m2 = new int[3][3];  for (int i = 0; i <= 2; i++) {  for (int j = 0; j <= 2; j++) {  m2[i][j] = sc.nextInt();  }  }  if (equals(m1, m2)) {  System.out.println("The two arrays are strictly identical");  } else {  System.out.println("The two arrays are not strictly identical");  }  }  public static boolean equals(int[][] m1, int[][] m2) {  int count = 0;  for (int i = 0; i <= 2; i++) {  for (int j = 0; j <= 2; j++) {  if (m1[i][j] != m2[i][j]) {  count++;  }  }  }  return count == 0;  }  } |



|  |
| --- |
| Enter list1: 51 22 25 6 1 4 24 54 6  Enter list2: 51 22 25 6 1 4 24 54 6  The two arrays are strictly identical |

|  |
| --- |
| Enter list1: 51 25 22 6 1 4 24 54 6  Enter list2: 51 22 25 6 1 4 24 54 6  The two arrays are not strictly identical |

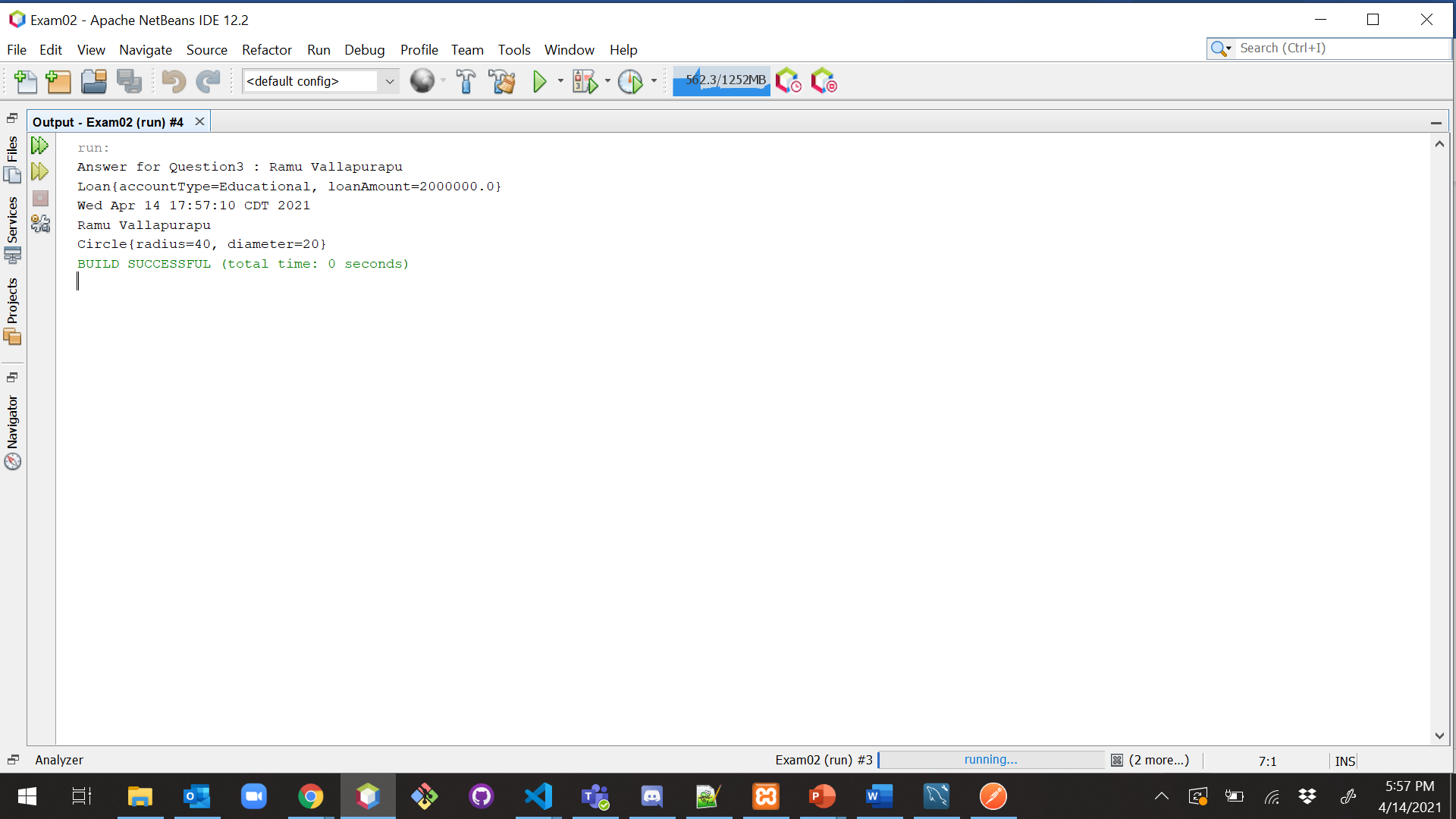
1. (10-Points) (Array List) Write a program that creates an ArrayList and adds a **Loan** object, a **Date** object (Use inbuilt method. No need to create separate class), a string, and a **Circle** object to the list, and use a loop to display all the elements in the list **by** invoking the object’s **toString**() method.

Note: For **Loan** and **Circle** you can use your own attributes and methods. **Constructor** and **tostring()** are mandatory requirements

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package arrayListQuestion03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Loan {  private String accountType;  private double loanAmount;  public String getAccountType() {  return accountType;  }  public void setAccountType(String accountType) {  this.accountType = accountType;  }  public double getLoanAmount() {  return loanAmount;  }  public void setLoanAmount(double loanAmount) {  this.loanAmount = loanAmount;  }  public Loan(String accountType, double loanAmount) {  this.accountType = accountType;  this.loanAmount = loanAmount;  }  @Override  public String toString() {  return "Loan{" + "accountType=" + accountType + ", loanAmount=" + loanAmount + '}';  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package arrayListQuestion03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Circle {  private int radius;  private int diameter;  public Circle(int radius, int diameter) {  this.radius = radius;  this.diameter = diameter;  }  public int getRadius() {  return radius;  }  public void setRadius(int radius) {  this.radius = radius;  }  public int getDiameter() {  return diameter;  }  public void setDiameter(int diameter) {  this.diameter = diameter;  }  @Override  public String toString() {  return "Circle{" + "radius=" + radius + ", diameter=" + diameter + '}';  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package arrayListQuestion03;  import java.util.ArrayList;  import java.util.Date;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ArrayListDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  ArrayList<Object> list = new ArrayList<>();  Date date = new Date();  Loan loan = new Loan("Educational", 2000000.00);  Circle circle = new Circle(40, 20);  String name = "Ramu Vallapurapu";  list.add(loan);  list.add(date);  list.add(name);  list.add(circle);  for (Object l : list) {  System.out.println(l.toString());  }  }  } |



1. (15-Points) What is Inheritance, Polymorphism and Late binding polymorphism? Explain and demonstrate with examples. Provide executable code screenshots for examples.

**Inheritance:**

When two different classes have common features among each other, then the concept of Inheritance will come into the picture. Inheritance is one of the key concepts under OOPs. It represents the IS-A relationship which is also known as a parent-child relationship, if a class wants to use commonly used state and behavior from other classes then the child class will inherit the parent class, before that it should pass the IS-A relationship where the instance of a child should be an instance of the parent class. Instead of creating new methods, we can reuse the methods/fields declared in the parent class/superclass.

Class inherited the methods/field == Parent Class/super class/base class

Class inherits the methods/fields == child Class/sub Class/derived class

**extends** is the keyword used to inherit the parent class in a child class.

**Polymorphic substitution:**

In general terminology Polymorphism means "many forms" where one thing can be performed in many ways, and it occurs when we have many classes that are related to each other by inheritance. As we know that Inheritance lets users to inherit fields and methods from the super class.

If a developer wants to override the implementation of one of the methods inherited from the super class and wants to invoke it without any issues then polymorphic substation will play a major role by storing the instance of subclass ref in the superclass ref variable, So compiler can decide to call which class and method to be executed.

**Late Binding:**

If we have a method declared in the superclass is overridden in the subclass, then deciding the class method to be executed based on the receiver side of the ref variable at runtime by the compiler is called Late binding.

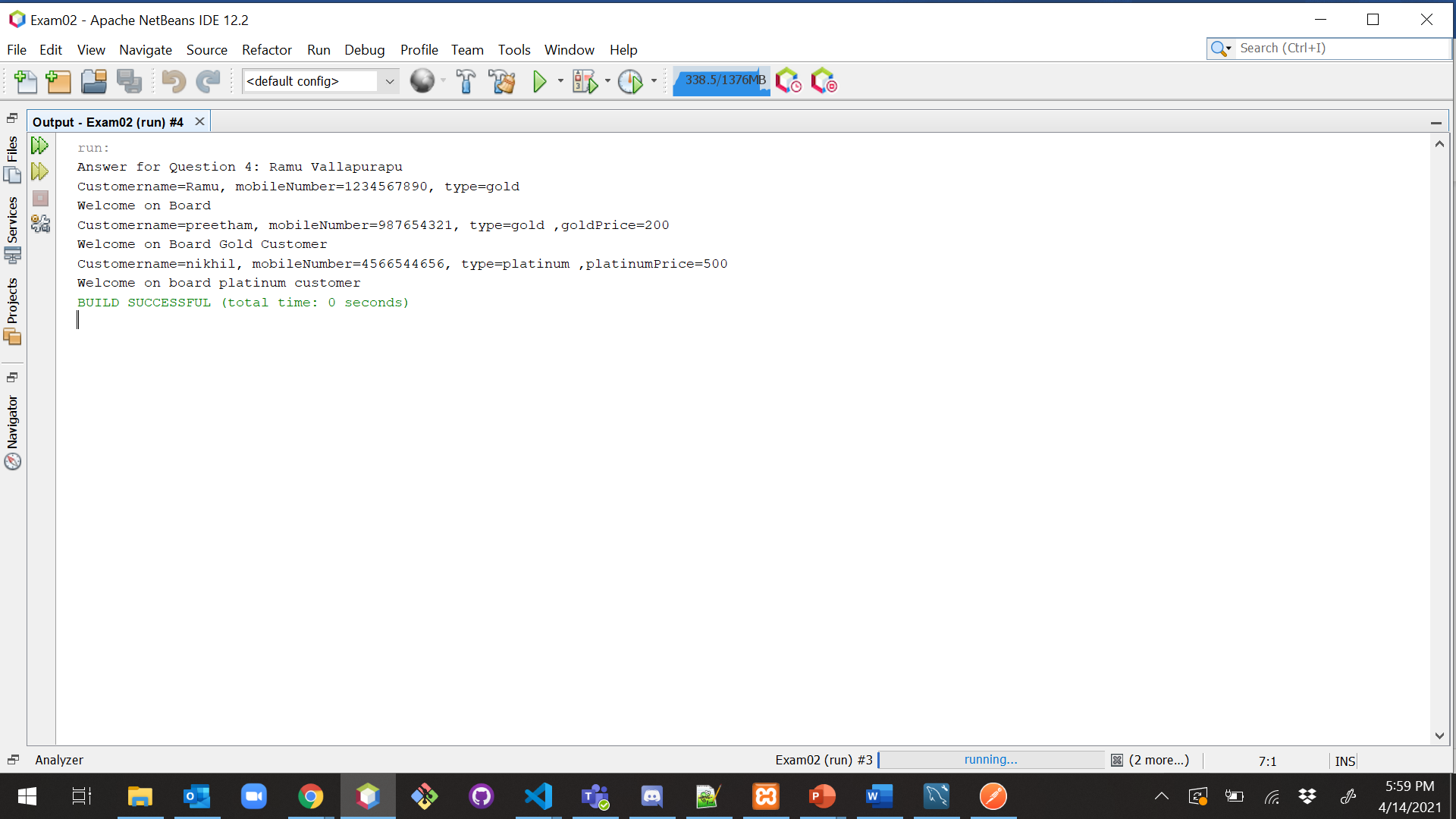
**Example:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Customer {  private String name;  private String mobileNumber;  private String type;  public Customer(String name, String mobileNumber, String type) {  this.name = name;  this.mobileNumber = mobileNumber;  this.type = type;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getMobileNumber() {  return mobileNumber;  }  public void setMobileNumber(String mobileNumber) {  this.mobileNumber = mobileNumber;  }  public String getType() {  return type;  }  public void setType(String type) {  this.type = type;  }  public void welcome() {  System.out.println("Welcome on Board");  }  @Override  public String toString() {  return "Customer" + "name=" + name + ", mobileNumber=" + mobileNumber + ", type=" + type;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class GoldCustomer extends Customer {  private int goldPrice;  public GoldCustomer(int goldPrice, String name, String mobileNumber, String type) {  super(name, mobileNumber, type);  this.goldPrice = goldPrice;  }  /\*\*  \*  \*/  @Override  public void welcome() {  System.out.println("Welcome on Board Gold Customer");  }  @Override  public String toString() {  return super.toString() + " " + ",goldPrice=" + goldPrice;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class PlatinumCustomer extends Customer {  private int platinumPrice;  public PlatinumCustomer(int platinumPrice, String name, String mobileNumber, String type) {  super(name, mobileNumber, type);  this.platinumPrice = platinumPrice;  }  /\*\*  \*  \*/  @Override  public void welcome() {  System.out.println("Welcome on board platinum customer");  }  @Override  public String toString() {  return super.toString() + " " + ",platinumPrice=" + platinumPrice;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class CustomerDriver {  public static void main(String[] args) {  System.out.println("Answer for Question 4: Ramu Vallapurapu");  Customer customer = new Customer("Ramu", "1234567890", "gold");  Customer goldCustomer = new GoldCustomer(200, "preetham", "987654321", "gold");  PlatinumCustomer platinumCustomer = new PlatinumCustomer(500, "nikhil", "4566544656", "platinum");  System.out.println(customer.toString());  customer.welcome();  System.out.println(goldCustomer.toString());  goldCustomer.welcome();  System.out.println(platinumCustomer.toString());  platinumCustomer.welcome();  }  } |



**Explanation:**

In the above example, I have defined a class called **Customer**, it has fields like name, mobileNumber, and the type and it has getter/setter methods along with that we have a **toString** method.

Now I have defined two other classes like **goldCustomer** and **platinumCustomer** where they common features as the customer class so with help of inheritance I have to **extend** the class customer so now the gold and platinum will have the state and behavior of the class customer.

Methods are extended into child class with a keyword called override which means these methods are rewritten on something that is existed (ParentClass) so now we can define our implementation which will help in code reusability.

In the above example, we have created an object for child class **GoldCustomer** that is stored in the reference variable of the super class which is called **Polymorphic Substitution**. when we call the ref variable it returns the methods declared in the subclass.

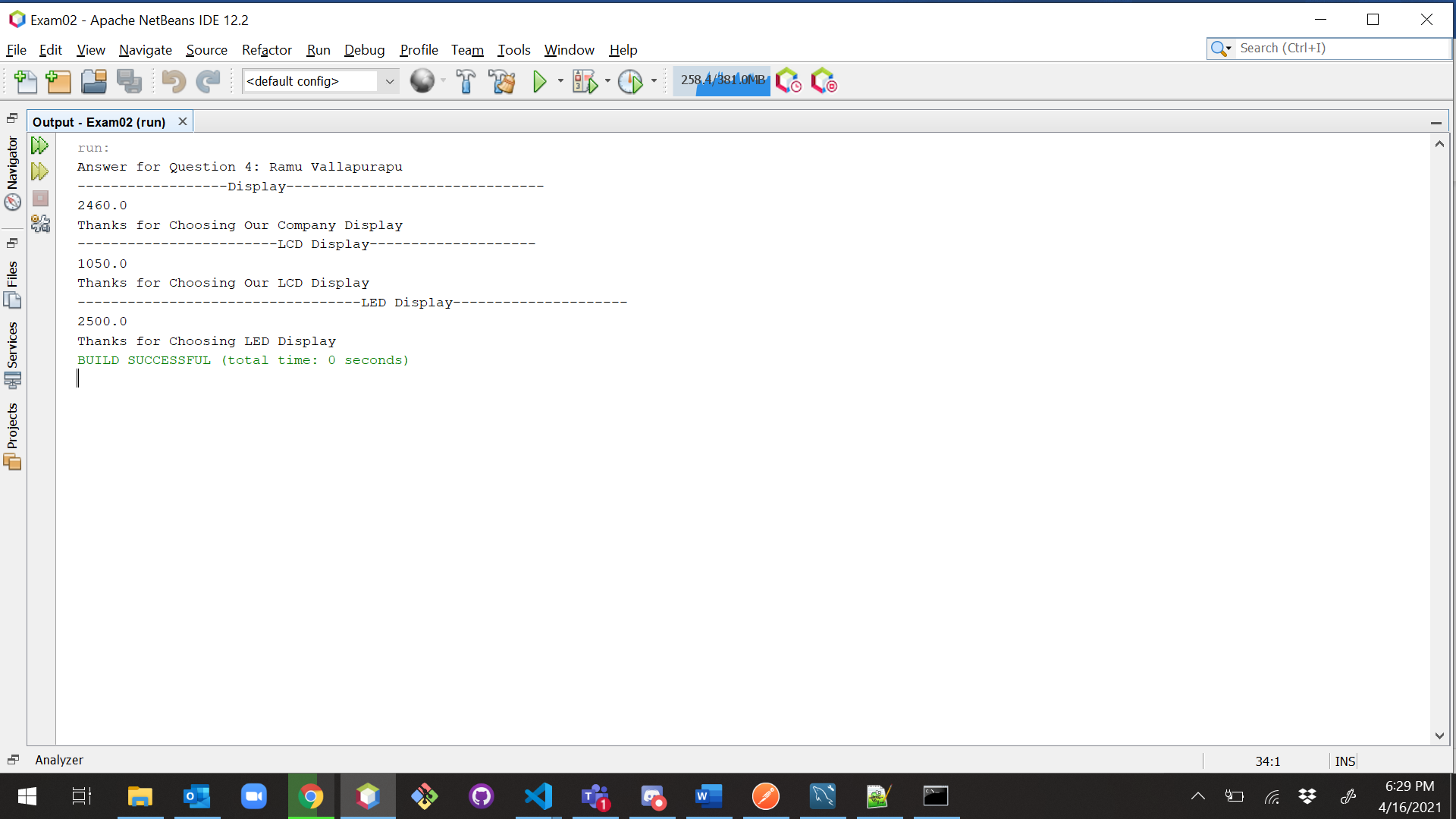
In the below statements we have **toString** method calls so at runtime it will check to which methods need to call and, in the process, child classes will get invoke first and check for **toString()** method if there is no implementation then the parent class will get invoke and will check for toString(), still if there is no **toString**() then object class will get invoke and will return the output and the process of invoking the valid **toString** is called **late binding .**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Display {  private int cost;  private double inches;  public Display(int cost, double inches) {  this.cost = cost;  this.inches = inches;  }  public int getCost() {  return cost;  }  public double getInches() {  return inches;  }  public void display() {  System.out.println("Thanks for Choosing Our Company Display");  }  public void cost() {  System.out.println(cost \* inches);  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class LCDDisplay extends Display {  public LCDDisplay(int cost, double inches) {  super(cost, inches);  }  @Override  public void display() {  System.out.println("Thanks for Choosing Our LCD Display");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class LEDDisplay extends Display {  public LEDDisplay(int cost, double inches) {  super(cost, inches);  }  @Override  public void display() {  System.out.println("Thanks for Choosing LED Display");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package inheritanceAndPolymorphism;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class DisplayDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for Question 4: Ramu Vallapurapu");  Display d = new Display(120, 20.5);  Display d1 = new LCDDisplay(100, 10.5);  Display d2 = new LEDDisplay(50, 50);  System.out.println("------------------Display-------------------------------");  d.cost();  d.display();  System.out.println("------------------------LCD Display--------------------");  d1.cost();  d1.display();  System.out.println("----------------------------------LED Display---------------------");  d2.cost();  d2.display();  }  } |



1. (10-Points) Design a class named **Person** and its two subclasses named **Student** and **Employee**. Make **Faculty** and **Staff** subclasses of **Employee**. A person has a name, address, phone number, and email address. A student has a grade and class status (Graduate). Define the status as a constant. An employee has an office, salary, and date hired. A faculty member has office hours and number of teaching subjects. A staff member has a title. Override the **toString** method in each class to display the class name and the person’s name.

Draw the UML diagram for the classes and implement them. Write a test program that creates a **Person**, **Student**, **Employee**, **Faculty**, and **Staff**, and invokes their **toString**() methods.

|  |
| --- |
|  |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class PersonDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for Question 5: Ramu Vallapurapu");  Person person = new Person("Ramu");  Student student = new Student("Krishna");  Employee employee = new Employee("Raja");  Faculty faculty = new Faculty("Charles Hoot");  Staff staff = new Staff("Dr Deva");  System.out.println(person.toString());  System.out.println("-------------------------");  System.out.println(student.toString());  System.out.println("-------------------------");  System.out.println(employee.toString());  System.out.println("-------------------------");  System.out.println(faculty.toString());  System.out.println("-------------------------");  System.out.println(staff.toString());  System.out.println("-------------------------");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Person {  private String name;  private String address;  private String phoneNumber;  private String emailAdress;  public Person(String name, String address, String phoneNumber, String emailAdress) {  this.name = name;  this.address = address;  this.phoneNumber = phoneNumber;  this.emailAdress = emailAdress;  }  public Person(String name) {  this.name = name;  }  public String getName() {  return name;  }  @Override  public String toString() {  return "Person Class \nName Of Person:" + name;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Student extends Person {  private static final String STATUS = "Graduate";  private double grade;    public Student(double grade, String name, String address, String phoneNumber, String emailAdress) {  super(name, address, phoneNumber, emailAdress);  this.grade = grade;  }  public Student(String name) {  super(name);  }  @Override  public String toString() {  return "Student Class \nName of Student: " + super.getName();  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  import java.util.Date;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Employee extends Person {  private String office;  private double salary;  private Date dateHired;  public Employee(String office, double salary, Date dateHired, String name, String address, String phoneNumber, String emailAdress) {  super(name, address, phoneNumber, emailAdress);  this.office = office;  this.salary = salary;  this.dateHired = dateHired;  }  public Employee(String name) {  super(name);  }  @Override  public String toString() {  return "Employee Class \nName of Employee: " + super.getName();  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  import java.util.Date;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Faculty extends Employee {  private int officeHours;  private int noOfTeachingSubjects;  public Faculty(int officeHours, int noOfTeachingSubjects, String office, double salary, Date dateHired, String name, String address, String phoneNumber, String emailAdress) {  super(office, salary, dateHired, name, address, phoneNumber, emailAdress);  this.officeHours = officeHours;  this.noOfTeachingSubjects = noOfTeachingSubjects;  }  public Faculty(String name) {  super(name);  }  @Override  public String toString() {  return "Faculty Class\nName of Faculty: " + super.getName();  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package individualDetails;  import java.util.Date;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Staff extends Employee {  private String title;  public Staff(String title, String office, double salary, Date dateHired, String name, String address, String phoneNumber, String emailAdress) {  super(office, salary, dateHired, name, address, phoneNumber, emailAdress);  this.title = title;  }  public Staff(String name) {  super(name);  }  @Override  public String toString() {  return "Staff Class \nName of Staff: " + super.getName();  }  } |

|  |
| --- |
|  |

Note: All classes should have **toString()** Method.

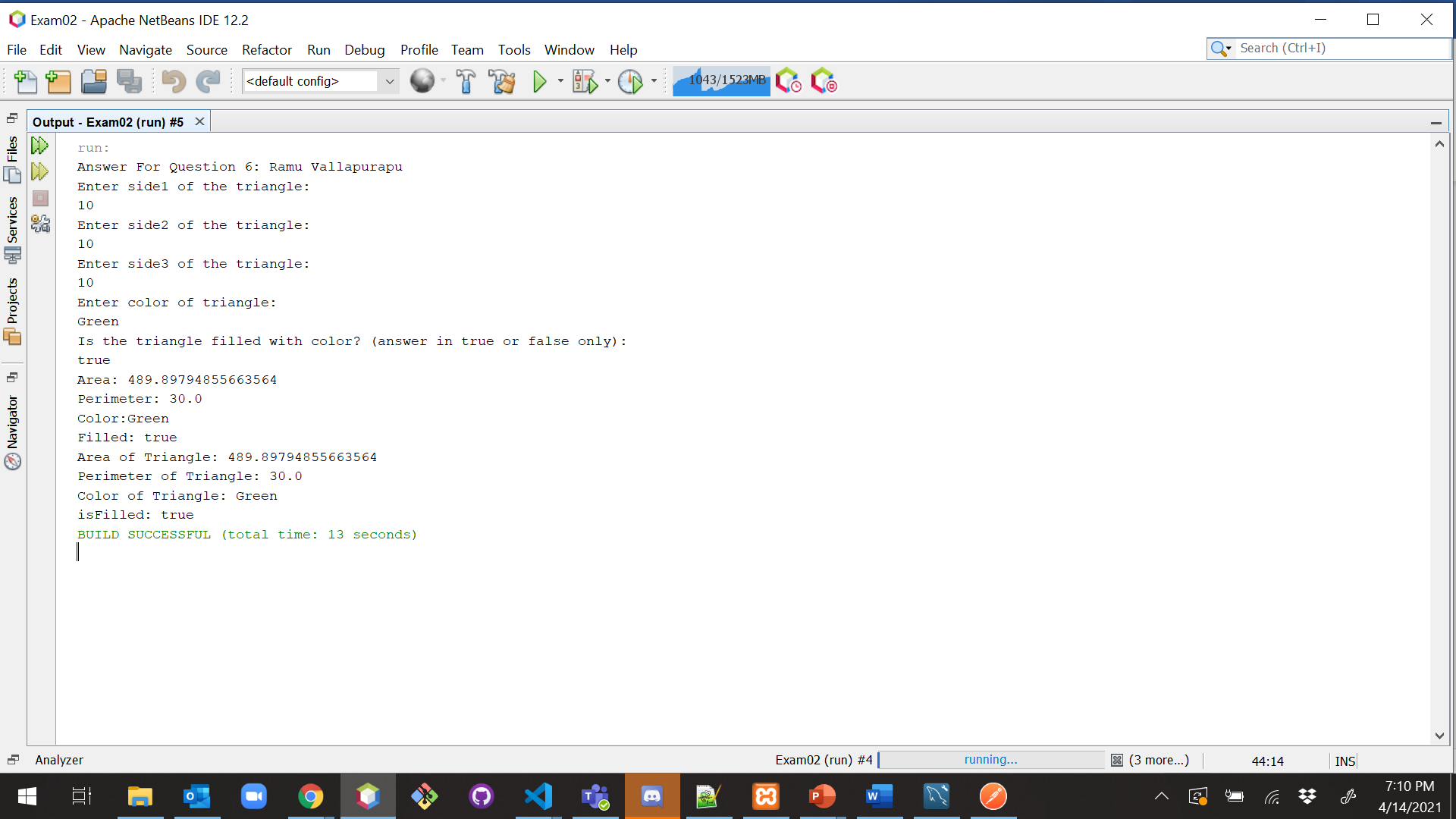
1. (10-Points) Design a new **Triangle** class that extends the abstract **GeometricObject** class. Draw the UML diagram for the classes **Triangle** and **GeometricObject** and then implement the **Triangle** class. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a **Triangle** object with these sides and set the color and filled properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not. Provide screenshot of executable code with input and output.

|  |
| --- |
|  |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package shapes;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  abstract class GeometricObject {  public abstract double getArea();  public abstract double getPerimeter();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package shapes;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Triangle extends GeometricObject {  private double side1;  private double side2;  private double side3;  private boolean isFilled;  private String color;  public Triangle(double side1, double side2, double side3, boolean isFilled, String color) {  this.side1 = side1;  this.side2 = side2;  this.side3 = side3;  this.isFilled = isFilled;  this.color = color;  }  public boolean isIsFilled() {  return isFilled;  }  public void setIsFilled(boolean isFilled) {  this.isFilled = isFilled;  }  public String getColor() {  return color;  }  public void setColor(String color) {  this.color = color;  }  @Override  public double getPerimeter() {  return (this.side1 + this.side2 + this.side3);  }  @Override  public double getArea() {  double p = this.getPerimeter();  return (Math.sqrt((p - this.side1) \* (p - this.side2)  \* (p - this.side3) \* p));  }  @Override  public String toString() {  return "Area: " + this.getArea() + "\nPerimeter: " + this.getPerimeter()  + "\nColor:" + this.getColor()  + "\nFilled: " + this.isIsFilled();  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package shapes;  import java.util.Scanner;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class TriangleDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer For Question 6: Ramu Vallapurapu");  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  System.out.println("Enter side1 of the triangle: ");  double side1 = scanner.nextDouble();  System.out.println("Enter side2 of the triangle: ");  double side2 = scanner.nextDouble();  System.out.println("Enter side3 of the triangle: ");  double side3 = scanner.nextDouble();  scanner.nextLine();  System.out.println("Enter color of triangle: ");  String color = scanner.nextLine();  System.out.println("Is the triangle filled with color?"  + " (answer in true or false only): ");  boolean isFilled = scanner.nextBoolean();  Triangle t1 = new Triangle(side1, side2, side3, isFilled, color);  System.out.println(t1.toString());  System.out.println("Area of Triangle: " + t1.getArea());  System.out.println("Perimeter of Triangle: " + t1.getPerimeter());  System.out.println("Color of Triangle: " + t1.getColor());  System.out.println("isFilled: " + t1.isIsFilled());  }  } |



1. (10-Points) What is an Enum in Java? Explain and demonstrate with some examples. Provide executable code screenshots for examples.

**ENUM:**

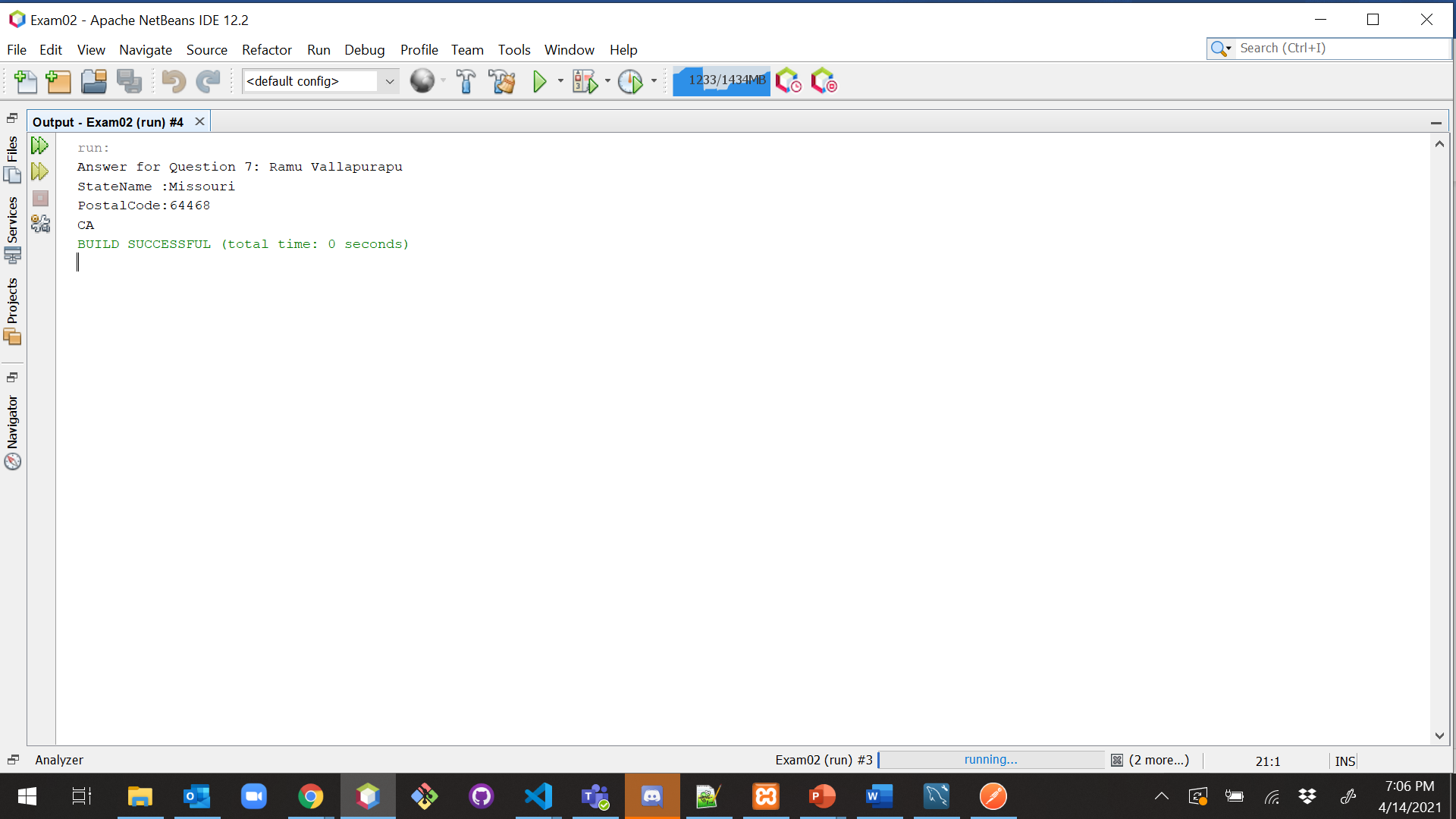
In general, we don’t have any datatype to declare constants in java but we can declare a constant value by using keyword Final in a class but with help of ENUM Java has given a flexibility to declare fixed set of constants in separate class.

Enum is basically a special data type and the Enum class will have fixed set of predefined constants and these constants are static and final implicitly and in ENUM class we can have methods and fields just like any other class. You can add methods which are abstract as well as concrete methods as well also. The very purpose of Enum is to enforce compile time type safety.

ENUM can be accessed by the class name.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public enum AreaZipCodes {  MO(64468), AR(72201), CA(94203);  private int zipcode;  private AreaZipCodes(int zipcode) {  this.zipcode = zipcode;  }  public int getZipcode() {  return zipcode;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class PostalCodeDriver {  private String stateName;  AreaZipCodes zipCodes;  public PostalCodeDriver(String stateName, AreaZipCodes zipCodes) {  this.stateName = stateName;  this.zipCodes = zipCodes;  }  public static void main(String[] args) {  PostalCodeDriver p = new PostalCodeDriver("Missouri", AreaZipCodes.MO);  System.out.println("StateName :" + p.stateName + "\nPostalCode:" + AreaZipCodes.MO.getZipcode());  System.out.println(AreaZipCodes.CA);  }  } |



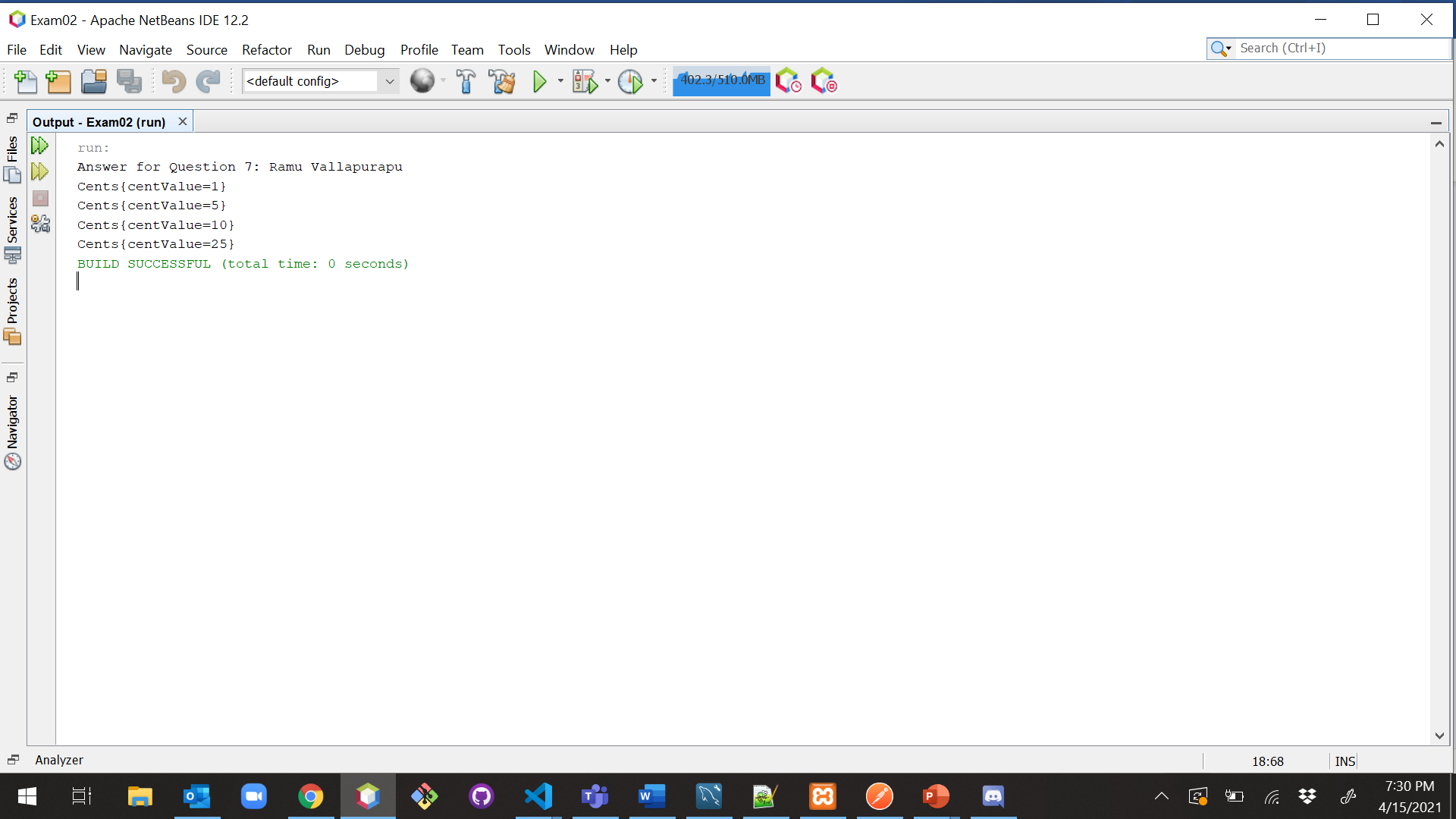
**Explanation:**

In the above example, we have created an Enum class and it contains the state names with their zip codes and they are constants, since it has a private constructor we cannot access that class and can modify the values in that class so the values are fixed, now in the driver class we can access the Enum class values directly with the help of class name.

**Example-2:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public enum Cents {  PENNY(1), NICKEL(5), DIME(10), QUARTER(25);  private final int centValue;  private Cents(int centValue) {  this.centValue = centValue;  }  public int getCentValue() {  return centValue;  }  @Override  public String toString() {  return "Cents{" + "centValue=" + centValue + '}';  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class CentsDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  for (Cents c : Cents.values()) {  System.out.println(c);  }  }  } |



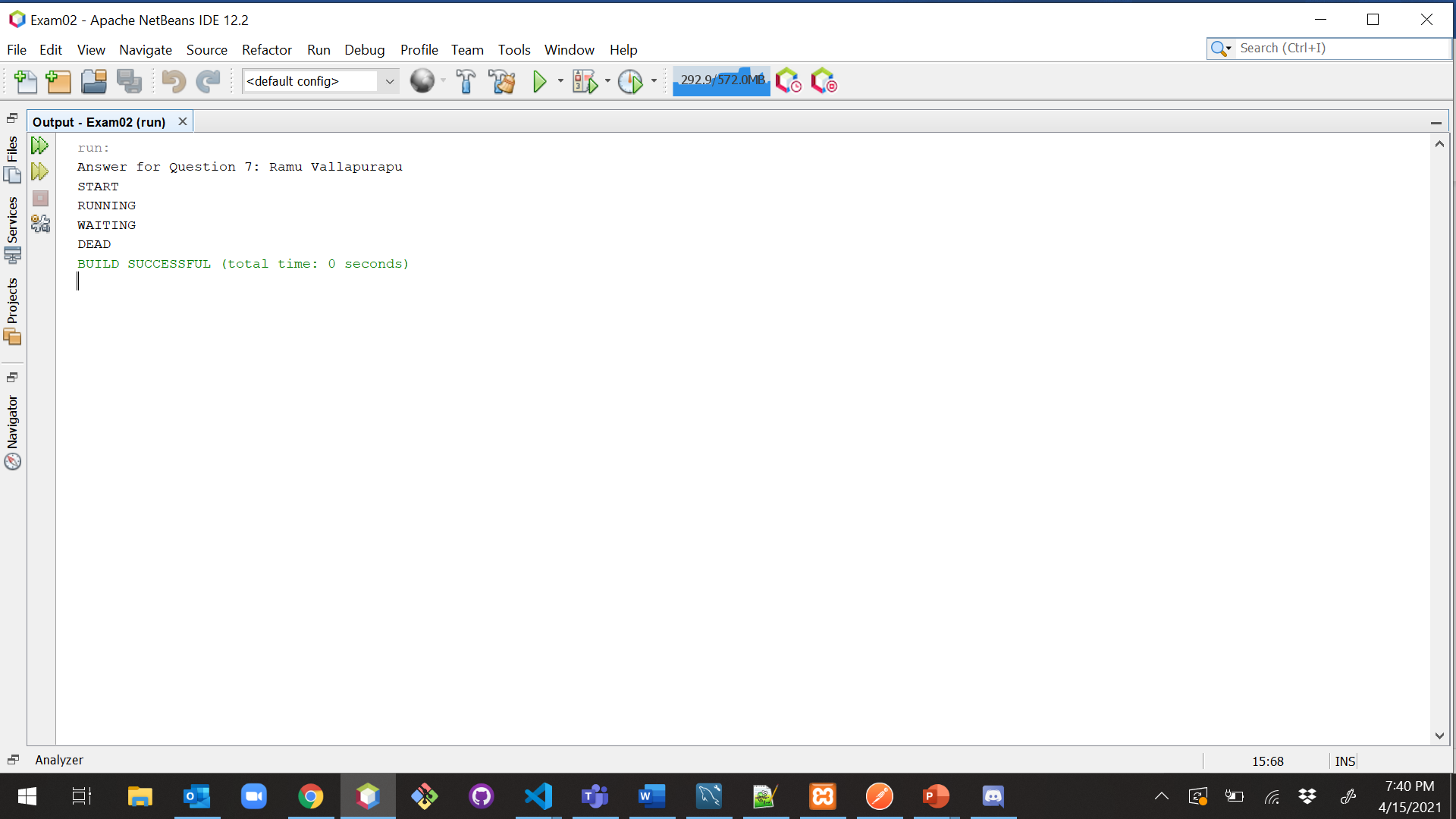
**Explanation:**

In the above example-2 we have declared a Enum called Cents and it contains coin name with cent values and we declared a variable called cent value with a **toString** method. Now in the driver class we are iterating the Enum class values with the help of a predefined method called **values()** so now it will iterate through each value in the Enum class.

**Example-3:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public enum ThreadStates {  START,  RUNNING,  WAITING,  DEAD;  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package enumExample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ThreadStatesDriver {  public static void main(String[] args) {  System.out.println(ThreadStates.START);  System.out.println(ThreadStates.RUNNING);  System.out.println(ThreadStates.WAITING);  System.out.println(ThreadStates.DEAD);  }  } |



1. (10-points) Define the term abstract class in java? Explain and demonstrate with some examples. Provide executable code screenshots for examples.

**Abstract Class:**

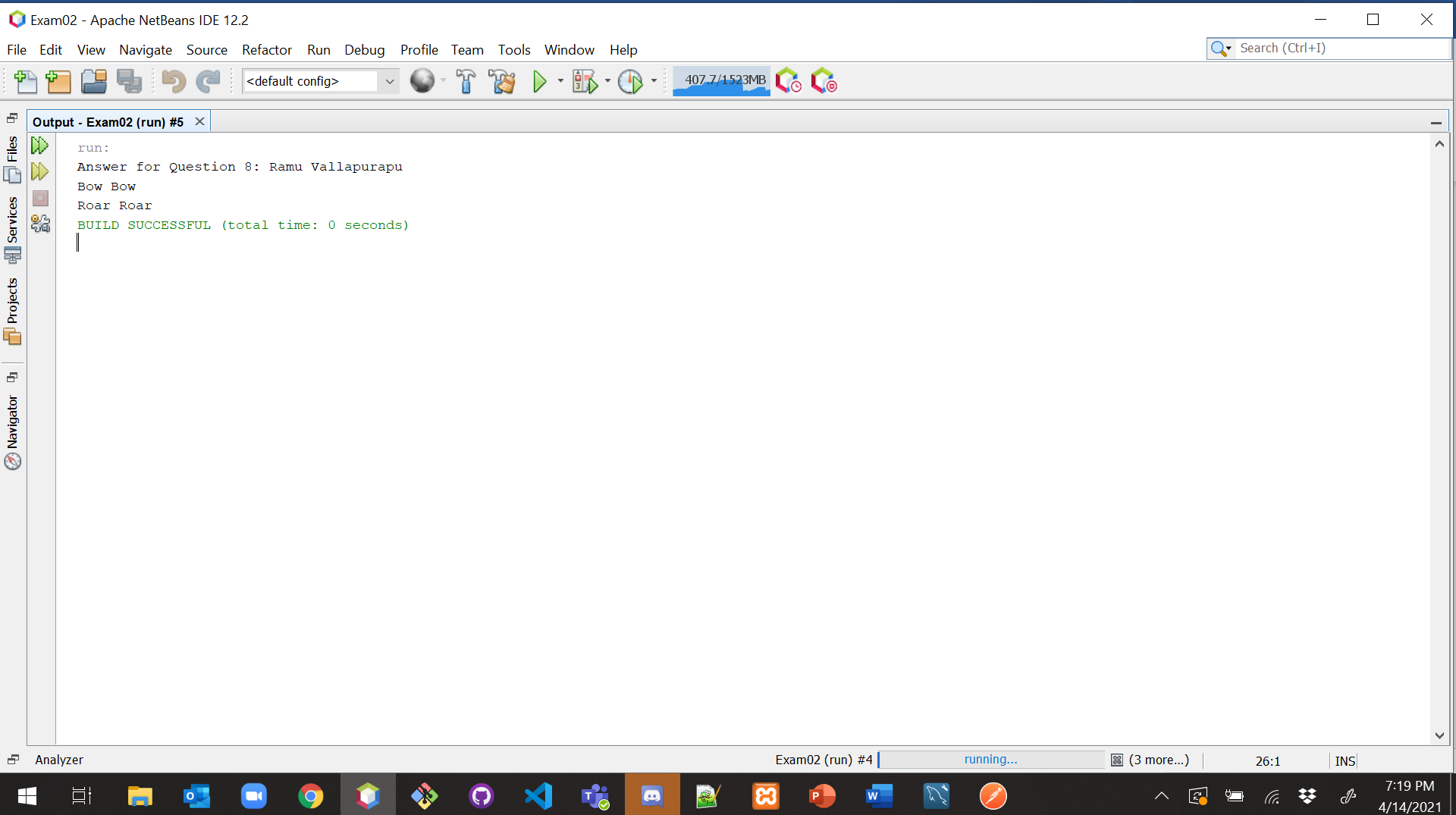
In general, if we want to implement/declare some method which is used commonly used among the other classes we will declare it in abstract class. Both abstract classes and methods are marked with the abstract keyword. It may or may not contain abstract methods (methods without body) as well as concrete methods (regular methods with the body). A normal class (non-abstract class) cannot have abstract methods. since we don’t have a constructor it is not allowed to create an object of abstract class. A class derived from the abstract class must implement all those methods that are declared as abstract in the parent class. If a child does not implement all the abstract methods of the abstract parent class, then the child class must need to be declared abstract as well.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  abstract class Animal {  //abstract method  public abstract void sound();    } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Dog extends Animal {  @Override  public void sound() {  System.out.println("Bow Bow");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Lion extends Animal {  @Override  public void sound() {  System.out.println("Roar Roar");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class AnimalDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for Question 8: Ramu Vallapurapu");  Dog d = new Dog();  d.sound();  Lion l = new Lion();  l.sound();  }  } |



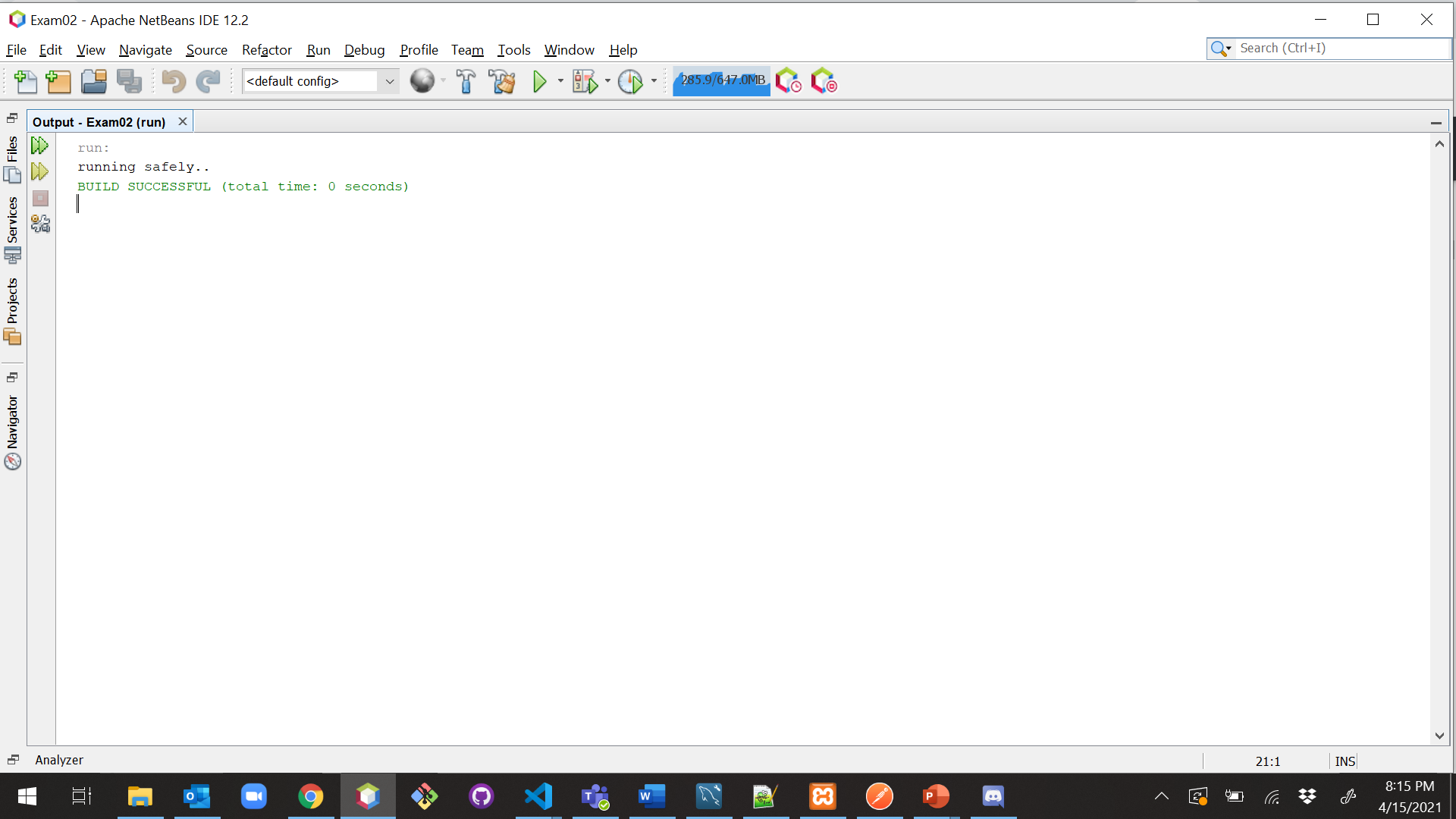
**Explanation:**

In the above example, we have created an abstract class with a method called sound which has the only declaration with no definition/body, now the Dog and Lion class extend the sound class and we can have our implementation if you see it in the main class the output is based on the respective class implementation on the methods extended.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  abstract class Bike {  abstract void run();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Honda extends Bike {  @Override  void run() {  System.out.println("running safely..");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package abstarctexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class VehicleDriver {  public static void main(String[] args) {  Honda h = new Honda();  h.run();  }  } |



**Explanation:**

In the above example, I have declared an abstract class with an abstract method (only method declaration) now it was extended by the Honda class since it is not an abstract class now class Honda has to implement the method. With the help of the abstract class, we will achieve abstraction (data hiding)

1. (10-points) Define the term interface in java? Explain and demonstrate with some examples. Provide executable code screenshots for examples.

**Interface:**

An interface is one of the ways to achieve abstraction which is hiding the implementation details and showing only functionality. Like a class, an Interface can have fields, method declarations and methods, but cannot be instantiated. The methods declared in an interface are by default abstract. To declare an interface, use the keyword interface. All the methods in an interface are public and all fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use keyword called **implements**.

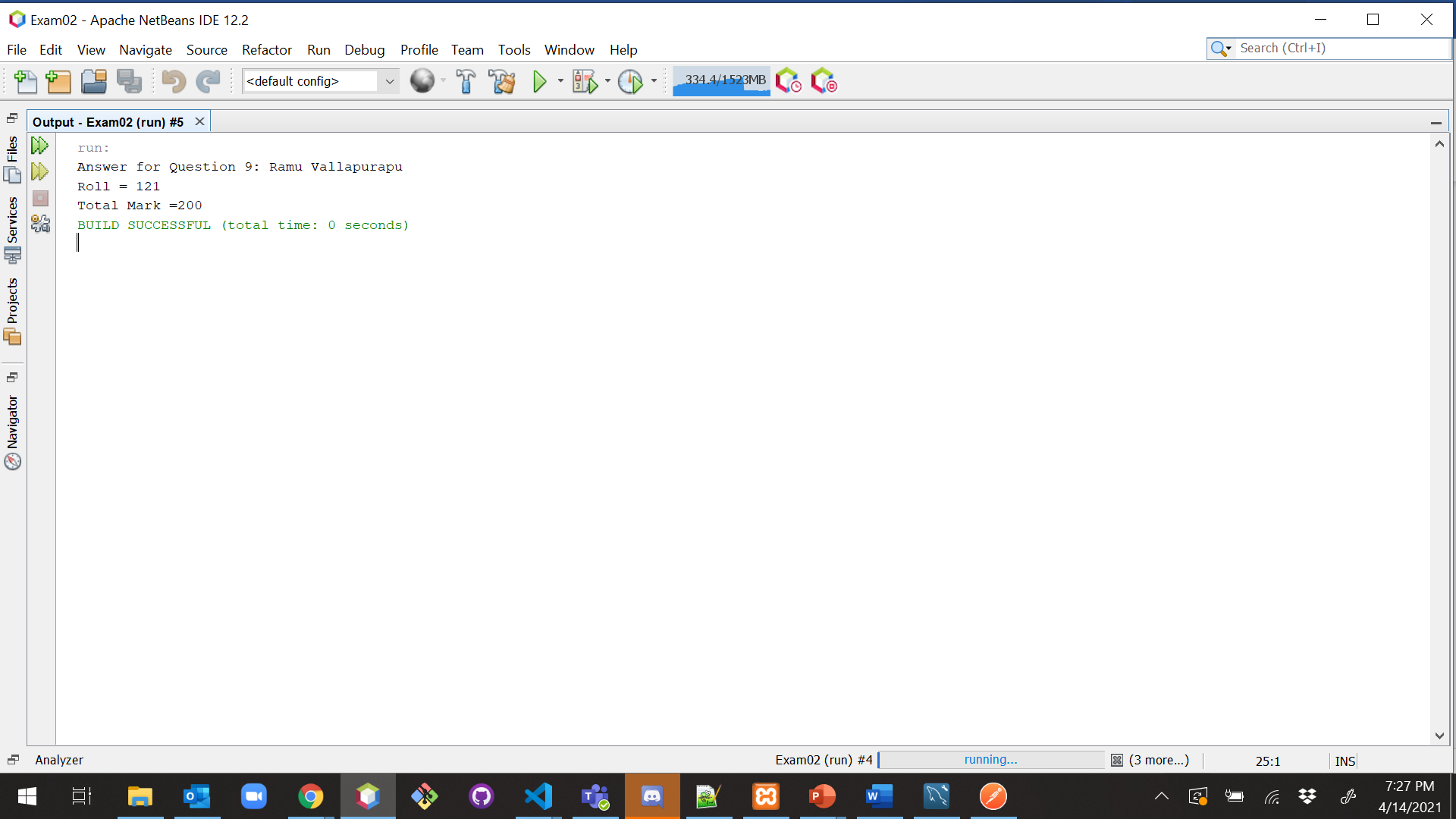
By interface, we can support the functionality of multiple inheritance.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface Father {  public static final int mark = 100;  int roll = 121;  void disp();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface Mother {  int mark = 100;  void add();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Son implements Mother, Father {  int sum = Father.mark + Mother.mark;  @Override  public void add() {  System.out.println("Roll = " + Father.roll);  }  @Override  public void disp() {  System.out.println("Total Mark =" + sum);  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class FamilyDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for Question 9: Ramu Vallapurapu");  Son s = new Son();  s.add();  s.disp();  }  } |



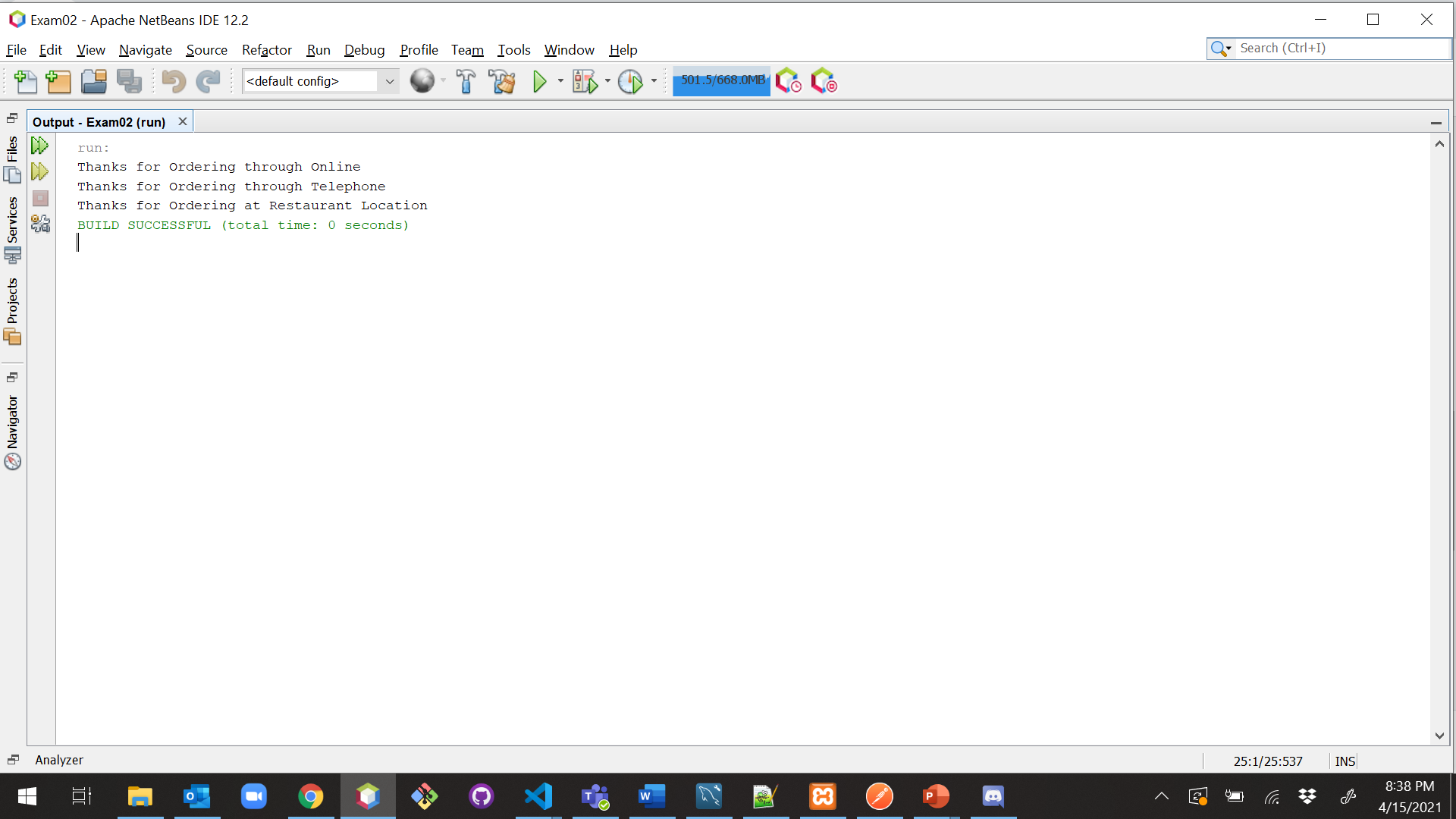
**Explanation:**

In the above example, we have created an interface called father and mother and now it was implemented by class Son. I have declared a constant, a variable, and a method as part of the interface, now the class which implements that interface must implement all the methods declared in that interface. Now we have implementation of methods declared in some other class and other classes can also implements that interface and can have their own implementation.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface RestaurantInterface {  public void acceptOnlineOrder();  public void takeTelephoneOrder();  public void walkInCustomerOrder();  } |

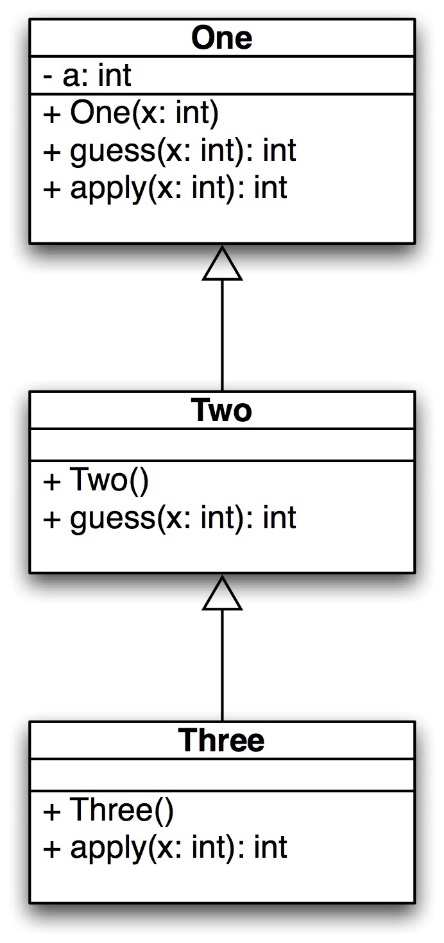
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class PlaceOrder implements RestaurantInterface {  @Override  public void acceptOnlineOrder() {  System.out.println("Thanks for Ordering through Online");  }  @Override  public void takeTelephoneOrder() {  System.out.println("Thanks for Ordering through Telephone");  }  @Override  public void walkInCustomerOrder() {  System.out.println("Thanks for Ordering at Restaurant Location");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package interfaceexample;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class RestaurantDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  PlaceOrder p = new PlaceOrder();  p.acceptOnlineOrder();  p.takeTelephoneOrder();  p.walkInCustomerOrder();  }  } |



1. (15-Points) Consider the following code for three classes One, Two, and Three. (A UML diagram is included for your convenience.)

|  |
| --- |
| public class One {  private int a;  public One(int in){  a = in;  }  public int guess (int x){  System.out.println("One guess " + x);  return a + x;  }    public int apply (int x){  System.out.println("One guess " + x);  return guess(x + 3);  }  } // end class One  public class Two extends One {  public Two(){  super(11);  }  public int guess(int x){  System.out.println("Two guess " + x);  return super.guess(x)+10;  }    } // end class Two  public class Three extends Two {  public int apply(int x){  System.out.println("Three apply " + x);  return -10;  }  } // end class Three |



What is the output of the following code? Explain it.

|  |
| --- |
| public static void main(String[] args) {  One hippo = new Three();  System.out.println(hippo.guess(4));  System.out.println(hippo.apply(12));  One lion = new One(-1);  System.out.println(lion.guess(5));  System.out.println(lion.apply(6));  } // end |

**Output:**

|  |
| --- |
| Two guess 4  One guess 4  25  Three apply 12  -10  One guess 5  4  One guess 6  One guess 9  8 |

**Explanation:**

In the first step, an object will create for class **Three** since it doesn't have any constructor it will call the default constructor of that class. Now compiler will comes to the next line Since it was polymorphic substitution it will check in the child class for method **Guess** first, it was not there it will check for parent class which is TWO and we have the method and it is override from class **One** it will start executing

The output for the first statement of method **Guess** is **Two guess 4** now it will go to 2nd line in the method where it has a return statement, but it is calling the parent class method

Now we are in Class **One** in this method, we have a SOUT statement so the output will be

**One guess 4**

now, it will go to return of the first method here we have (a+x) 🡪a=11 and x==4 so it will return 15

now the result will be in class **Two** with **15** as superclass value. now it will return the result by adding 10 to it so now in main class we got the result as **25**

**2nd Statement in Main Class**:

In this statement, we are calling method **hippo.apply(12)** since the method was overided now compiler will check for the method apply in class **Three** then it will start executing and it has SOUT as the first statement so the o/p will be

**Three apply 12**

since the method has a return statement it will return -10

**3rd statement in Main Class:**

Here object for class **One** is created it will call the constructor in class one and the value for variable **a** will set to -1

now we are calling the **Guess** method, but we are using the reference variable for class One so now the method in class **One** will get invoke and starts executing

it has SOUT as the first statement so the o/p will be

**One guess 5**

now it will execute return statement so it will return a+x == -1+5==> 4

now the apply method will get invoke and starts executing

so, the o/p will be

**One guess 6**

now it will execute the return statement but it has a guess method called with parameter x added to 3 in an arithmetic format so now it will get into the guess method of the same class because the ref variable created is for the same class with 9 as parameter

so, it will return the following statement

**One guess 9**

the return statement of guess is (a+x) 🡪 now x== 9, a == -1 so the output will -1+9 🡪8

the value 8 will return to apply method which will send back to driver class so the output will be 8.