**`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**

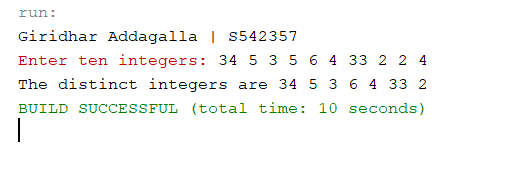
1. (5-Points) (1D-Array - )Write a method that removes the duplicate elements from an array list of integers using the following header:

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:

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

|  |
| --- |
| public class oneDimensionArray {  public static void main(String[] args) {  System.err.print("Enter ten integers: ");  Scanner sc = new Scanner(System.in);  ArrayList<Integer> intArray = new ArrayList<>();  while (intArray.size() < 10) {  intArray.add(sc.nextInt());  }  removeDuplicate(intArray);  }  public static void removeDuplicate(ArrayList<Integer> list) {  for (int i = 0; i < list.size() - 1; i++) {  for (int j = i + 1; j < list.size(); j++) {  if (Objects.equals(list.get(i), list.get(j))) {  list.remove(j);  }  }  }  System.out.print("The distinct integers are ");  for (int l : list) {  System.out.print(l + " ");  }  System.out.println("");  }  } |



**Explanation:** Theone-dimension array is looped internally and validated with one each other and the method returns the distant values. The method works while sending the array of integers and return void but it prints over there saying the distinct integers are.

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:

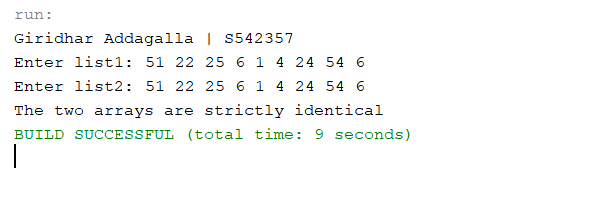
public static boolean equals(int[][] m1, int[][] m2)

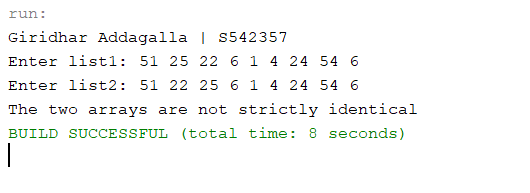
Write a test program that prompts the user to enter two 3 \* 3 arrays of integers and displays whether the two are strictly identical. Provide screenshot of executable code with input and output. Here are the sample runs.

|  |
| --- |
| 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 |

|  |
| --- |
| package Question2;  import java.util.Arrays;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class twodArrays {  public static void main(String[] args) {  int[][] arrayOne = new int[3][3];  int[][] arrayTwo = new int[3][3];  System.out.print("Enter list1: ");  Scanner sc = new Scanner(System.in);  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  arrayOne[i][j] = sc.nextInt();  }  }  System.out.print("Enter list2: ");  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  arrayTwo[i][j] = sc.nextInt();  }  }  if (equals(arrayOne, arrayTwo)) {  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) {  if (m1.length == m2.length) {  for (int i = 0; i < m1.length; i++) {  if (!Arrays.equals(m1[i], m2[i])) {  return false;  }  }  } else {  return false;  }  return true;  }  } |





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

**Loan.java**

|  |
| --- |
| package Question3;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Loan {  private String lName;  private double lAmount;  private int repayPeriod;  public Loan(String lName, double lAmount, int repayPeriod) {  this.lName = lName;  this.lAmount = lAmount;  this.repayPeriod = repayPeriod;  }  @Override  public String toString() {  return "Loan Details: \n" + "Name:" + lName + ", Loan Amount=" + lAmount + ", Repay Period=" + repayPeriod;  }  } |

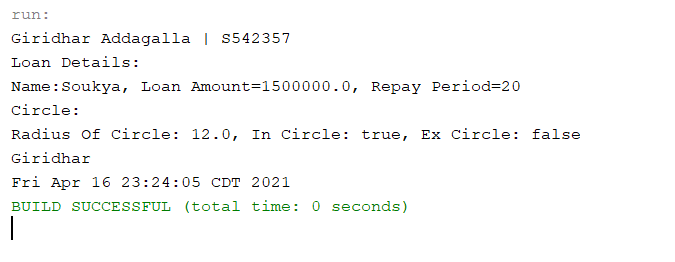
**Circle:**

|  |
| --- |
| package Question3;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Circle {  private double radius;  private boolean in\_Circle;  private boolean ex\_Circle;  public Circle(double radius, boolean in\_Circle, boolean ex\_Circle) {  this.radius = radius;  this.in\_Circle = in\_Circle;  this.ex\_Circle = ex\_Circle;  }  @Override  public String toString() {  return "Circle: \n" + "Radius Of Circle: " + radius + ", In Circle: " + in\_Circle + ", Ex Circle: " + ex\_Circle;  }  } |

**Driver:**

|  |
| --- |
| package Question3;  import java.util.ArrayList;  import java.util.Date;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class LoanCircleDriver {  public static void main(String[] args) {  ArrayList<Object> objArrayList = new ArrayList<>();  objArrayList.add(new Loan("Soukya", 1500000, 20));  objArrayList.add(new Circle(12, true, false));  objArrayList.add("Giridhar");  objArrayList.add(new Date());  for (Object obj : objArrayList) {  System.out.println(obj.toString());  }  }  } |

**Output:**



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

**Inheritance:**

It is the process of deriving a new class with the derived class properties by using the extend keyword. It is a one-way process, the base class properties are used in derived, but the derived properties will not be used in base class. It is mostly used for code reusability.

**Polymorphism:**

It means creating many forms from one. They are two types compile time and run time polymorphism. It can be performed by method overloading and method overriding.

**Late binding Polymorphism:**

Late binding or dynamic binding, the compiler doesn’t decide the method to be called. Overriding is an example for dynamic binding.

The compiler must map the name of the method to final implementation. It occurs after polymorphism substitution.

**Explanation:**

Here I created the plant class which is a parent class and it has the sub classes fruits and flowers. I inherit the parent classes to the sub classes that helps to gain the properties and override few methods, it is an example of the inheritance. In the Driver class called Plant Driver, in this I am creating an child class object and holding it with parent object, We called this is an late binding polymorphism. And it checks weather the class has that calling method or not then it will pick the method if it is not available it will get from the parent this is because of the late binding polymorphism. Creating the child class object with parent class called as the polymorphic substitution.

**Example: Plant Class (Parent class)**

|  |
| --- |
| package Question4;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Plants {  private String season;  private String plant\_color;  public Plants(String season, String color) {  this.season = season;  this.plant\_color = color;  }  @Override  public String toString() {  System.out.println("Inside parent plant class toString() method.");  return season + " Season plant " + ", Plant color is " + plant\_color;  }  } |

**Fruit Plant Class:**

|  |
| --- |
| package Question4;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Fruit\_Plants extends Plants {  private String fruitName;  public Fruit\_Plants(String fruitName, String season, String plant\_color) {  super(season, plant\_color);  this.fruitName = fruitName;  }  /\*\*  \* overriding the method toString from the parent class  \*  \* @return  \*/  @Override  public String toString() {  System.out.println("Inside Fruit Plants Sub class");  return super.toString() + ", Fruit Name is " + fruitName;  }  } |

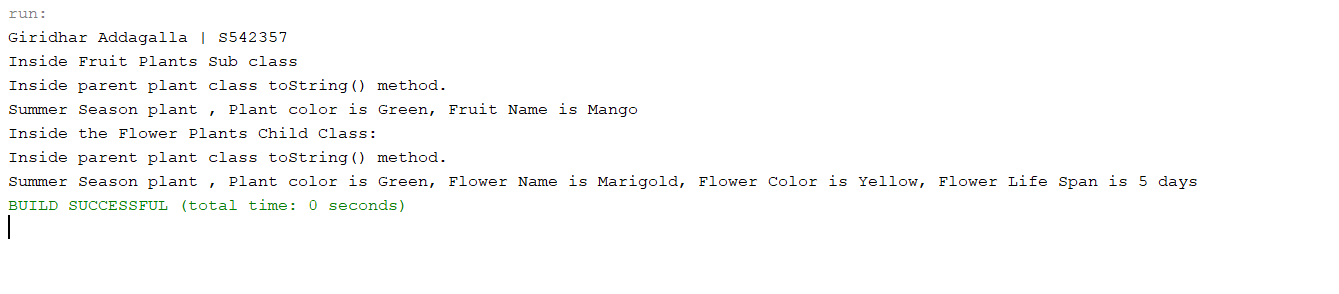
**Flowering Plant:**

|  |
| --- |
| package Question4;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Flower\_Plants extends Plants {  private String flowerColor;  private String flowerLifeSpan;  private String flowerName;  public Flower\_Plants(String flowerName, String flowerColor, String flowerLifeSpan, String season, String plant\_color) {  super(season, plant\_color);  this.flowerColor = flowerColor;  this.flowerName = flowerName;  this.flowerLifeSpan = flowerLifeSpan;  }  @Override  public String toString() {  System.out.println("Inside the Flower Plants Child Class:");  return super.toString() + ", Flower Name is " + flowerName + ", Flower Color is " + flowerColor + ", Flower Life Span is " + flowerLifeSpan;  }  } |

**Plant Driver Class:**

|  |
| --- |
| package Question4;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class PlantDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  //polymorphic substitution  Plants fruit\_plants;  fruit\_plants = new Fruit\_Plants("Mango", "Summer", "Green");  //late binding polymorphism  System.out.println(fruit\_plants.toString());  Plants flower\_plants = new Flower\_Plants("Marigold", "Yellow", "5 days", "Summer", "Green");  // Late binding polymorphism  System.out.println(flower\_plants.toString());  }  } |

**Output:**



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.

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

**UML Diagram:**

Diagram

Description automatically generated

**Person.java**

|  |
| --- |
| package Question5;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Person {  private String name;  private String address;  private long phoneNumber;  private String emailAddress;  public Person(String name, String address,  long phoneNumber, String emailAddress) {  this.name = name;  this.address = address;  this.phoneNumber = phoneNumber;  this.emailAddress = emailAddress;  }  public String getName() {  return name;  }  public String getAddress() {  return address;  }  public long getPhoneNumber() {  return phoneNumber;  }  public String getEmailAddress() {  return emailAddress;  }  @Override  public String toString() {  return "Person from " + this.getClass().getName() + " class is " + name;  }  } |

**Student.java**

|  |
| --- |
| package Question5;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Student extends Person {  private double grade;  public static final String CLASS\_STATUS = "Graduate";  public Student(double grade, String name, String address,  long phoneNumber, String emailAddress) {  super(name, address, phoneNumber, emailAddress);  this.grade = grade;  }  @Override  public String toString() {  return "Person name from " + this.getClass().getName() + " is "  + super.getName();  }  } |

**Employee.java**

|  |
| --- |
| package Question5;  import java.util.Date;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  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, long phoneNumber, String emailAddress) {  super(name, address, phoneNumber, emailAddress);  this.office = office;  this.salary = salary;  this.dateHired = dateHired;  }  @Override  public String toString() {  return "Person name from " + this.getClass().getName()  + " is " + super.getName();  }  } |

**Staff.java**

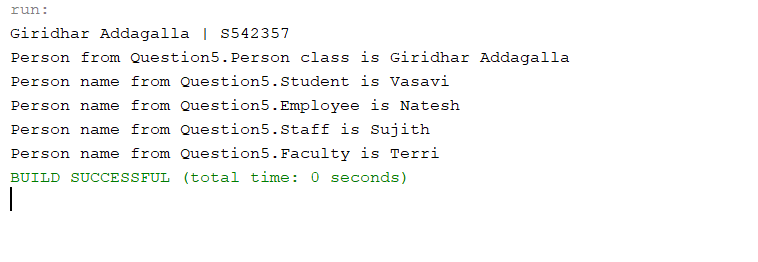
|  |
| --- |
| package Question5;  import java.util.Date;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Staff extends Employee {  private String title;  public Staff(String title, String office, double salary,  Date dateHired, String name, String address,  long phoneNumber, String emailAddress) {  super(office, salary, dateHired, name, address,  phoneNumber, emailAddress);  this.title = title;  }  @Override  public String toString() {  return "Person name from " + this.getClass().getName()  + " is " + super.getName();  }  } |

**Faculity.java**

|  |
| --- |
| package Question5;  import java.util.Date;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Faculty extends Employee {  private int officeHours;  private int numberTeachingSubjects;  public Faculty(int officeHours, int numberTeachingSubjects,  String office, double salary, Date dateHired, String name,  String address, long phoneNumber, String emailAddress) {  super(office, salary, dateHired, name, address, phoneNumber,  emailAddress);  this.officeHours = officeHours;  this.numberTeachingSubjects = numberTeachingSubjects;  }  @Override  public String toString() {  return "Person name from " + this.getClass().getName() + " is "  + super.getName();  }  } |

**PersonDriver.java**

|  |
| --- |
| package Question5;  import java.util.Date;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class PersonDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Date d = new Date();  Person personObj = new Person("Giridhar Addagalla", "Maryville",  91960321, "g196@gmail.com");  Student stuObj = new Student(10.0, "Vasavi", "Maryville",  98334462, "v@gmail.com");  Employee empObj = new Employee("Northwest", 13000, d, "Natesh",  "Maryville", 123456, "nghan@gmail.com");  Staff staffObj = new Staff("Service Worker", "Dinning", 15000, d,  "Sujith", "MAryville", 3256789, "sujith@gmail.com");  Faculty facObj = new Faculty(12, 3, "Colden Hall", 700000, d,  "Terri", "Northwest", 787878787, "terri@gmail.com");  System.out.println(personObj);  System.out.println(stuObj);  System.out.println(empObj);  System.out.println(staffObj);  System.out.println(facObj);  }  } |



**Explanation:**

The person class is extended by the student and employee class. I am overriding the to string method in all the classes to print the class name and the person name. Then finally I extended the feature form the parent class and I use the constructor of that class to set the values.

I also used Employee as parent class for Staff and faculty classes. In the driver classes I create objects for all the classes and invoke the toString() method and print their results.

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.

**UML Diagram:**

Diagram

Description automatically generated

**GeomentricObject.java**

|  |
| --- |
| package Question6;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class GeometricObject {  abstract double perimeter();  abstract double area();  } |

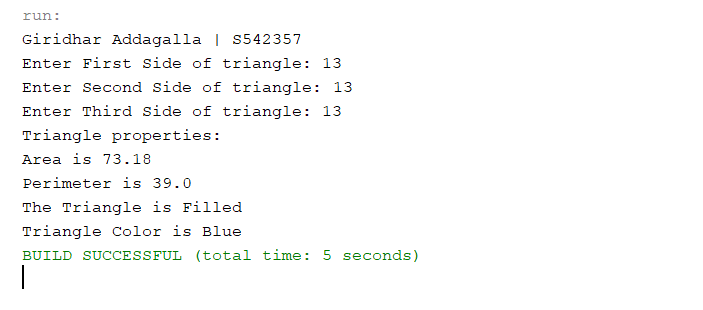
**Triangle.java**

|  |
| --- |
| package Question6;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Triangle extends GeometricObject {  private double firstSide;  private double secondSide;  private double thirdSide;  private boolean isFilled;  private String triangleColor;  public Triangle(double firstSide, double secondSide, double thirdSide,  boolean isFilled, String triangleColor) {  this.firstSide = firstSide;  this.secondSide = secondSide;  this.thirdSide = thirdSide;  this.isFilled = isFilled;  this.triangleColor = triangleColor;  }  @Override  double perimeter() {  return this.firstSide + this.secondSide + this.thirdSide;  }  @Override  double area() {  double pHAF = this.perimeter() / 2;  return Math.sqrt(Math.abs(pHAF \* (pHAF - firstSide)  \* (pHAF - secondSide) \* (pHAF - thirdSide)));  }  @Override  public String toString() {  return "Triangle properties: \nArea is "  + Math.round(this.area() \* 100) / 100.0  + "\nPerimeter is " + this.perimeter()  + "\nThe Triangle is "  + (isFilled ? "Filled" : "Not Filled ")  + "\nTriangle Color is " + triangleColor;  }  } |

**TriangleDriver.java**

|  |
| --- |
| package Question6;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class TriangleDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  Scanner scan = new Scanner(System.in);  boolean flag = false;  double firstSide = 0, secondSide = 0, thirdSide = 0;  do {  flag = false;  System.out.print("Enter First Side of triangle: ");  firstSide = scan.nextDouble();  System.out.print("Enter Second Side of triangle: ");  secondSide = scan.nextDouble();  System.out.print("Enter Third Side of triangle: ");  thirdSide = scan.nextDouble();  if (firstSide + secondSide <= thirdSide || secondSide + thirdSide <= firstSide || thirdSide + firstSide <= secondSide) {  System.out.println("The sum of 2 sides should be greater than the third side");  flag = true;  }  } while (flag == true);  Triangle triangle = new Triangle(firstSide, secondSide, thirdSide, true, "Blue");  System.out.println(triangle);  }  } |

**Output:**



**Explanation:**

There is an abstract class GeometricObject.java and it has 2 abstract methods which are extended and implemented in the child class called triangle. The abstract methods are area and parameter.

In the class triangle it has the constructor which can take the parameters and pass the values like sides, color and Boolean and tells weather the triangle is filled or not.

It also has the driver class which creates object of the triangle class and we invoke toString() method which prints area parameter and color it is filled or not.

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

**Enum:**

Enum is the typed constraint, it represents the group of the named constraints. The values defined in the Enum class are the runtime. Every type inside the classes are classed as the Enum types. It is always public, and the constructor will invoke the values. All the constraint defined in enum are always public static final.

**Explanation:**

I have take an example of the bag zips, The total no of zips for a bag, They are different types of bags but they all do have zips, So I have created a Enum class bag Zips and add the SCHOOLBAG, COLLEGEBAG, TOURBAG, HANDBAG. This Enum class helps me to find number of zips for a bag and this can be access using the public get method name getBagZips().

In the driver class I used the appropriated method to get the information. The constant can be get by calling BagZips.SCHOOLBAG etc, It will get the name of the Enum and if we call our public getNofZips() method the we will get the count of the zips.

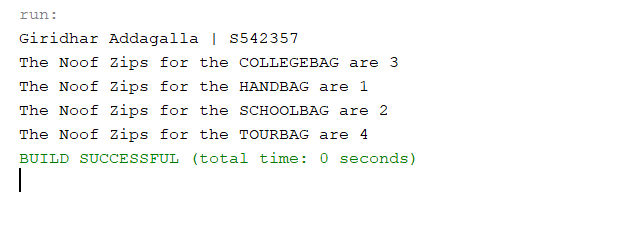
**Enum Class BagZips**

|  |
| --- |
| package Question7;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public enum BagZips {  SCHOOLBAG(2), COLLEGEBAG(3), TOURBAG(4), HANDBAG(1);  private int bagZips;  private BagZips(int bagZips) {  this.bagZips = bagZips;  }  public int getNofZips() {  return bagZips;  }  } |

**Driver class BagZipDriver**

|  |
| --- |
| package Question7;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class BagZipDriver {  public static void main(String[] args) {  System.out.println("The Noof Zips for the " + BagZips.COLLEGEBAG + " are " + BagZips.COLLEGEBAG.getNofZips());  System.out.println("The Noof Zips for the " + BagZips.HANDBAG + " are " + BagZips.HANDBAG.getNofZips());  System.out.println("The Noof Zips for the " + BagZips.SCHOOLBAG + " are " + BagZips.SCHOOLBAG.getNofZips());  System.out.println("The Noof Zips for the " + BagZips.TOURBAG + " are " + BagZips.TOURBAG.getNofZips());  }  } |

**Output:**



**Example 2:**

Explanation: In this example I have choose the fan wings, All fans will not have the 3 wings some of them will have 4 or more based on the work it is created. I have created an Enum class called FanWings and it has HEATERFAN, NORMALFAN, TURBOFAN. In the driver class I have used called the Enum class in the different way using the advanced for loop, It will helps to access the data with less coding and gives more grip to the Enum class.

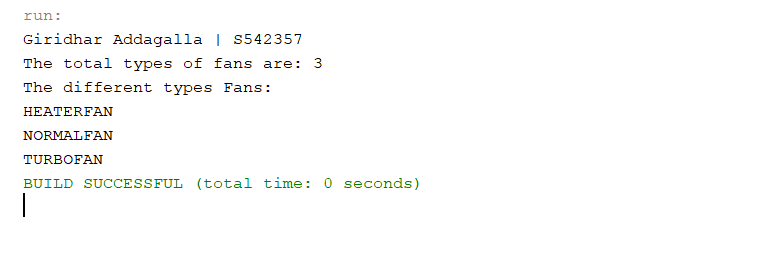
**FanWings.java:**

|  |
| --- |
| package Question7\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public enum FanWings {  HEATERFAN(4), NORMALFAN(3), TURBOFAN(8);  private int noOFWings;  FanWings(int noOFWings) {  this.noOFWings = noOFWings;  }  public int getNoOFWings() {  return noOFWings;  }  } |

**WingsDriver.java:**

|  |
| --- |
| package Question7\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class WingsDriver {  public static void main(String[] args) {  System.out.println("The total types of fans are: " + FanWings.values().length);  System.out.println("The different types Fans: ");  for (FanWings value : FanWings.values()) {  System.out.println(value);  }  }  } |

**Output:**



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

**Abstract:**

Abstract is the key word which is used to create the abstract class. The abstract class has the unimplemented methods which needs to be developed in the extended class. Once the abstract class is extended automatically the abstract methods needs to be implemented.

The abstract class has the abstract methods names which are not implemented because they are the necessary method need to be implemented individually.

**Example:**

Let us take a abstract class called device properties, it has the abstract method called chargingRunTime and screen size.

But the charging run time method is not implemented in the abstract class because it is an abstract method. Now I will take an class iPhone which is an electronic device and I extends the abstract class called device properties, Now the iPhone class should implement the chargingRunTime and screen size method and provide the information.

**Device Properties abstract class:**

|  |
| --- |
| package Question8;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class DeviceProperties {  public abstract double chargingRunTime();  public abstract double displaySize();  } |

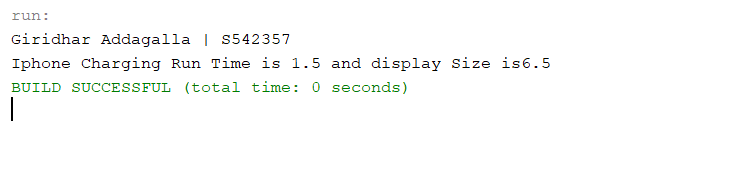
**Iphone.java**

|  |
| --- |
| package Question8;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Iphone extends DeviceProperties {  @Override  public double chargingRunTime() {  return 1.5;  }  @Override  public double displaySize() {  return 6.5;  }  } |

**ElectronicDriver.java**

|  |
| --- |
| package Question8;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ElectronicDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  Iphone iphone = new Iphone();  System.out.println("Iphone Charging Run Time is " + iphone.chargingRunTime() + " and display Size is" + iphone.displaySize());  }  } |

**Output:**



**Example 2:**

Here I have choose the abstract class called the bead properties, This class has the abstract method called bed size and the return type is string, This will say what kind of the bed we are choose, the method is implemented in the Beds class where it will fetch the values like length and breadth and returns the size of the bed either small, queen or king size bed.

**Bed Properties abstract class:**

|  |
| --- |
| package Question8\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class BedProperties {  abstract String beadSize();  } |

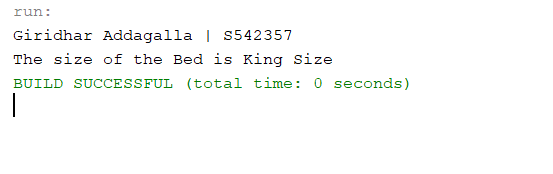
**Bed.java**

|  |
| --- |
| /\*  \* 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 Question8\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Beds extends BedProperties {  private double length, breath;  Beds(double length, double breath) {  this.breath = breath;  this.length = length;  }  @Override  String beadSize() {  if (length > 12 && breath > 20) {  return "Small Size";  } else if (length > 20 && breath > 30) {  return "Queen size";  } else {  return "King Size";  }  }  } |

**BedDriver.java**

|  |
| --- |
| package Question8\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class BedDriver {  public static void main(String[] args) {  Beds bd = new Beds(10, 20);  System.out.println("The size of the Bed is" + bd.beadSize());  }  } |

**Output:**



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

**Interface:**

It is the collection of the abstract methods, It may also contain the constants and methods without the definition. It is the inter connection between two entities. The method should be implemented by using the implements key word at the start of the class.

**Example:**

I created a toll gate as the interface class and added two abstract method, The two abstract method are implemented using the child class which implements the parent class called the toll gate class. Now I have added few properties to the vehicle class name as no of wheels, Based on the wheels count I have written the code to say weather it is a two wheel or car or truck, Now using the same thing the tax percentage is also calculated. All the two-vehicle type and tax are the abstract methods.

**Tollgate Interface:**

|  |
| --- |
| package Question9;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public interface TollGate {  public abstract String vehicleType();  public abstract double tax();  } |

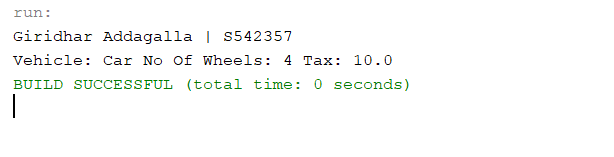
**Vehicle Class**

|  |
| --- |
| /\*  \* 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 Question9;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Vehicle implements TollGate {  private int noOfWheels;  public Vehicle(int noOfWheels) {  this.noOfWheels = noOfWheels;  }  @Override  public String vehicleType() {  if (noOfWheels == 4) {  return "Car";  } else if (noOfWheels == 2) {  return "Two Wheeler";  } else {  return "Truck";  }  }  @Override  public double tax() {  if (null == vehicleType()) {  return 20;  } else {  switch (vehicleType()) {  case "Car":  return 10;  case "Two Wheeler":  return 5;  default:  return 20;  }  }  }  @Override  public String toString() {  return "Vehicle: " + vehicleType() + " No Of Wheels: " + noOfWheels + " Tax: " + tax();  }  } |

**Toll Gate Driver**

|  |
| --- |
| package Question9;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class TollGateDriver {  public static void main(String[] args) {  Vehicle v = new Vehicle(4);  System.out.println(v);  }  } |

**Output:**



**Example 2:**

Here I created the interface called the call charges and added the default method and a few methods which are not implemented but they are not abstract. Still we can able to override the methods an create our own functionality. The default method returns the default values. And the other methods are implemented.

**Call charges Interface**

|  |
| --- |
| package Question9\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public interface CallCharges {  default double local() {  return 0;  }  double sTDCalls();  double iSDCalls();  } |

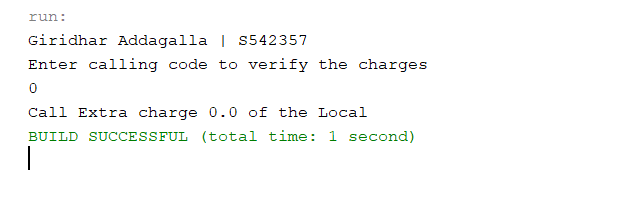
**Calling.java**

|  |
| --- |
| /\*  \* 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 Question9\_Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Calling implements CallCharges {  @Override  public double local() {  return CallCharges.super.local(); //To change body of generated methods, choose Tools | Templates.  }  @Override  public double sTDCalls() {  return 0.5;  }  @Override  public double iSDCalls() {  return 10;  }  } |

**CallingDriver.java**

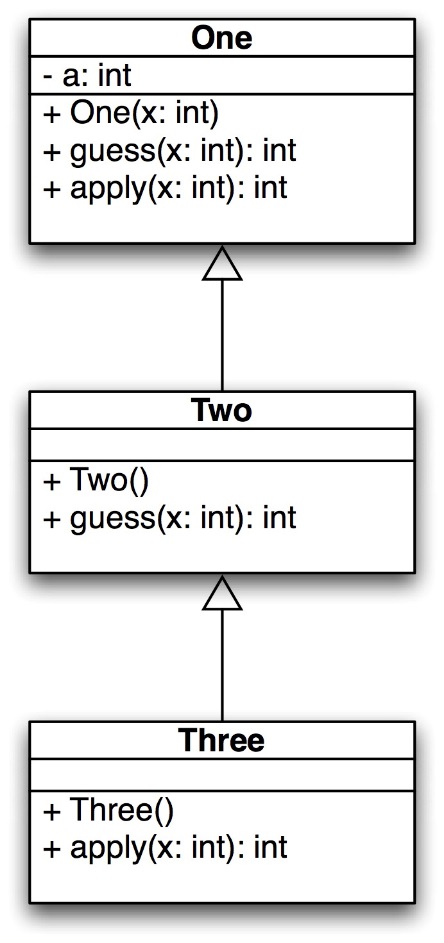
|  |
| --- |
| package Question9\_Example2;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class CallingDriver {  public static void main(String[] args) {  Calling c = new Calling();  Scanner sc = new Scanner(System.in);  System.out.println("Enter calling code to verify the charges");  int code = sc.nextInt();  if (code == 0) {  System.out.println("Call Extra charge " + c.local() + " of the Local");  } else if (code == 1) {  System.out.println("Call Extra charge " + c.iSDCalls() + " of the ISD");  } else {  System.out.println("Call Extra charge " + c.sTDCalls() + " of the STD");  }  }  } |

**Output:**



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

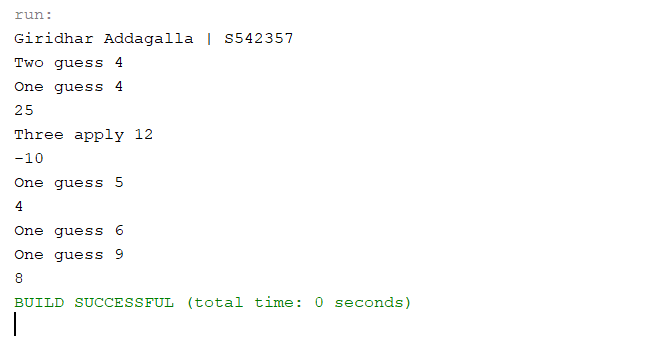
|  |
| --- |
| public class One {  private int a;  public One(int in){  a = in;  }  public int guess (int x){  System.out.println("One guess " + x);  return a + x;  }    public int apply (int x){  System.out.println("One guess " + x);  return guess(x + 3);  }  } // end class One  public class Two extends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



**Explanation and Tracing:**

The driver class is execute and as the first statement the One hippo = new Three(); get executed and it is an polymorphic substitution. Since it is holding the instance of the child object.

When the method is called using the hippo object the method inside the sub class three are called. It is late binding polymorphism.

Hippo.guess(4) is called the print statement now the java will check for the method and Finds in the super class, The it is called and print the result. All this is happening because of the inheratence.

The two may has the guess method but it is from the super **one.**

The constructor invoked and the value of 11 will be from the two class, Now it will give the value a+x that is a is 11 and x is 4 which add the 10 in the return. The TWO class returns 25.

The output from the first statement is:

Two guess 4

One guess 4

25

The hippo.apply(12) , the three class will be presented and return the value -10.

The print output is

Three apply 12

-10

The Lion instance is created from the one class with the parameter -1.

One class has the guess and apply this will invoke

The output is and return 4

One guess 5

4

The remaining super class will be executed and it goes to the one class and get printed

One guess 6

One guess 9 //and returns

8

Git hub Link: [giridhar196/Java-Exam-2 (github.com)](https://github.com/giridhar196/Java-Exam-2)