**Final Exam 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 . 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.

**Final Exam OBJECT-ORIENTED PROG 01FA20 150 pts**

1. (20-Points) Define the terms abstract classes and interfaces. What are the similarities and differences between abstract classes and interfaces? Why interfaces are preferred over abstract classes? Explain and demonstrate with examples.

ANSWER

Explanation –

Interface: The interface is a plan that will be used to apply a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.

An interface cannot be instantiated. But classes that implement interfaces can be instantiated. Interfaces will never contain instance variables but, they can contain public static final variables.

Abstract: Abstract class is nothing special to look at when you want to differentiate with a regular class. But it has got some methods which are unimplemented, or only declared, not defined. In that case, all those methods and the class itself are renamed with the “abstract” keyword. An abstract class can have one or multiple number of unimplemented methods. As an abstract class is not proper or does not have ideally defined methods compared to a regular class, so abstract classes cannot be instantiated, that means we cannot create any direct objects of abstract classes.

Similarities: Abstract class and Interface are java basic object types.

Abstract class and Interface can contain variables and methods (Interface methods can only have statements, while class methods can have implementation code.)

Abstract class and Interface be inherited using Inheritance (implements keyword for interfaces and extends keyword for groups

Differences-

1)By default, variables declared in a Java interface are final. Non-final variables can be present in an abstract class.

2) The interface can be implemented using an abstract class. An abstract class cannot be implemented by an interface.

3) An abstract class can extend another Java class and implement several Java interfaces, while an interface can only extend another Java interface.

4) By design, members of a Java interface are public. Personal, safe, and other class members can be found in a Java abstract class.

Interface is preferred over abstract class because an interface helps you accomplish abstraction and polymorphism, which are two of the main four OOP principles.

It also makes it easier to keep your code loosely coupled instead of tightly coupled, which happens when high-level modules depend on low-level modules.

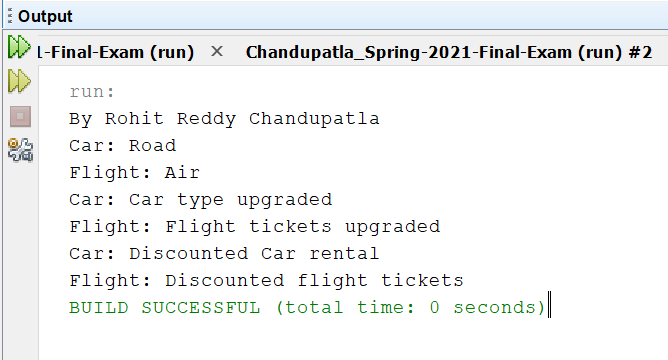
Example for Abstract:

Explanation:

An abstract class named reservations is created. And then two other classes named FlightReservation and CarReservation are created and extended by the abstract class reservation where all the methods are given and overridden. Finally, a driver class named Driver is created and all these methods are called and the final print statements are given.

|  |
| --- |
| /\*  \* 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 Question01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public abstract class Reservation {  final String category;  Reservation(String category) {  this.category = category;  }  // abstract methods  public abstract String upgrade();  public abstract String getDiscount();  }    /\*  \* 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 Question01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class CarReservation extends Reservation{  CarReservation() {  super("Road");  }  @Override  public String upgrade() {  return "Car type upgraded";  }  @Override  public String getDiscount() {  return "Discounted Car rental";  }  }  /\*  \* 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 Question01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class FlightReservation extends Reservation{  FlightReservation() {  super("Air");  }  @Override  public String upgrade() {  return "Flight tickets upgraded";  }  @Override  public String getDiscount() {  return "Discounted flight tickets";  }  }  /\*  \* 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 Question01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here    CarReservation carReservation = new CarReservation();  FlightReservation flightReservation = new FlightReservation();    System.out.println("By Rohit Reddy Chandupatla");  System.out.println("Car: " + carReservation.category);  System.out.println("Flight: " + flightReservation.category);  System.out.println("Car: " + carReservation.upgrade());  System.out.println("Flight: " + flightReservation.upgrade());  System.out.println("Car: " + carReservation.getDiscount());  System.out.println("Flight: " + flightReservation.getDiscount());  }  } |

OUTPUT-

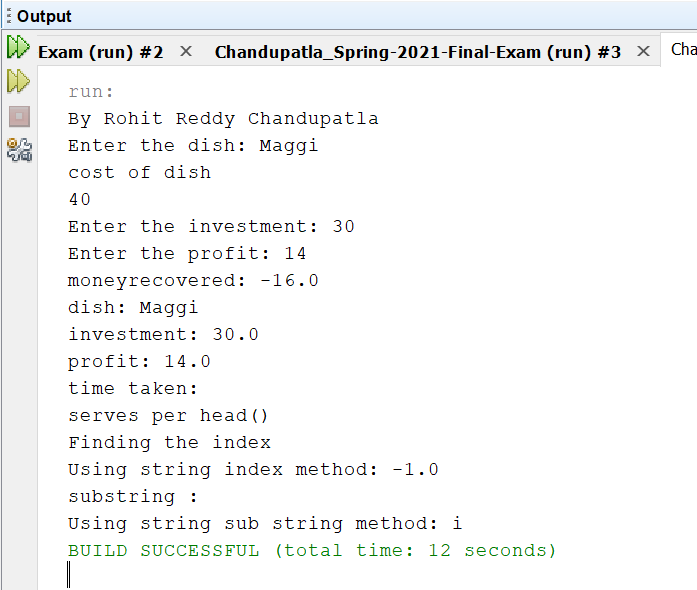


Example for Interface -

Explanation:

Interface classes are a type of interface. Where public methods are provided, a class maggi is created, and then duration and serves are created. All other methods are provided and overridden, except for Time and Distributed, which are implanted. Then a maggiDriver driver class is formed, in which all of these are called, and the print statements are given according to the performance.

|  |
| --- |
| /\*  \* 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 Question01.Interface01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public interface duration {  String timetaken();  default String myst(String name) {  String str = name.substring(3);  return str;  }  }  /\*  \* 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 Question01.Interface01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public interface serves {    default double String(String sh) {  int str = sh.indexOf("b");  return str;  }  /\*\*  \*  \* @return  \*/  String servesperhead();  }    /\*  \* 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 Question01.Interface01;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class maggi implements duration, serves {  private final String dish;  private final double investment;  private final double profit;  public maggi(String dish, double investment, double profit) {  this.dish = dish;  this.investment = investment;  this.profit = profit;  }  public String getDish() {  return dish;  }  public double getInvestment() {  return investment;  }  public double getProfit() {  return profit;  }  public double moneyRecovered() {  return profit - investment;  }  @Override  public String timetaken() {  String str = " ";  if (dish.equals("mutton")) {  str = "time taken is 30minutes";  }  if (dish.equals("chicken")) {  str = "time taken is 20minutes";  } else if (dish.equals("panneer")) {  str = "time taken is 10minutes";  }  return str;  }  /\*\*  \*  \* @return  \*/  @Override  public String servesperhead() {  String str = " ";  if (dish.equals("mutton")) {  str = ("servings are 1");  }  if (dish.equals("chicken")) {  str = ("servings are 2");  }  if (dish.equals("panneer")) {  str = ("servings are 3");  }  return str;  }  /\*\*  \*  \* @param sh  \* @return  \*/  @Override  public double String(String sh) {  System.out.print("Using string index method: ");  int str = (sh.indexOf("b"));  return str;  }  /\*\*  \*  \* @param name  \* @return  \*/  @Override  public String myst(String name) {  System.out.print("Using string sub string method: ");  String str = name.substring(3);  return str;  }  @Override  public String toString() {  return "dish: " + dish + "\ninvestment: " + investment + "\nprofit: " + profit  + "\ntime taken: " + timetaken() + "\nserves per head()" + servesperhead();  }  }    /\*  \* 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 Question01.Interface01;  import java.util.Scanner;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class maggiDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  // TODO code application logic here  Scanner scan = new Scanner(System.in);  System.out.println("By Rohit Reddy Chandupatla");  System.out.print("Enter the dish: ");  String dish = scan.next();  System.out.println("cost of dish");  double cost = scan.nextDouble();  System.out.print("Enter the investment: ");  double investment = scan.nextDouble();  System.out.print("Enter the profit: ");  double profit = scan.nextDouble();  maggi b = new maggi(dish, investment, profit);  System.out.println("moneyrecovered: " + b.moneyRecovered());  System.out.println(b.toString());  System.out.println("Finding the index");  System.out.println(b.String("Aku"));  System.out.println("substring : ");  System.out.println(b.myst("Bhai"));  }  } |



1. (10-Points) Design an interface named Colorable with a void method named howToColor(). Every class of a colorable object must implement the Colorable interface. Design a class named Square that extends GeometricObject and implements Colorable Implement howToColor to display the message Color all four sides.

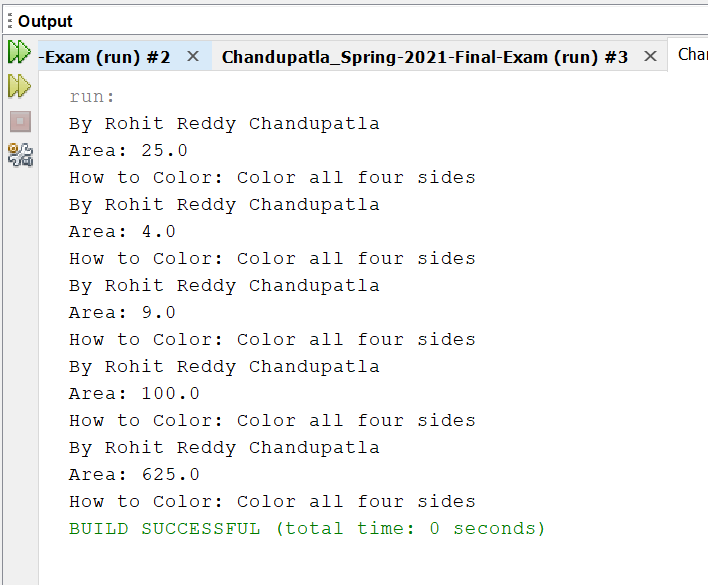
Draw a UML diagram that involves Colorable, Square, and GeometricObject. Write a test program that creates an array of five GeometricObjects. For each object in the array, display its area and invoke its howToColor method if it is colorable.

ANSWER

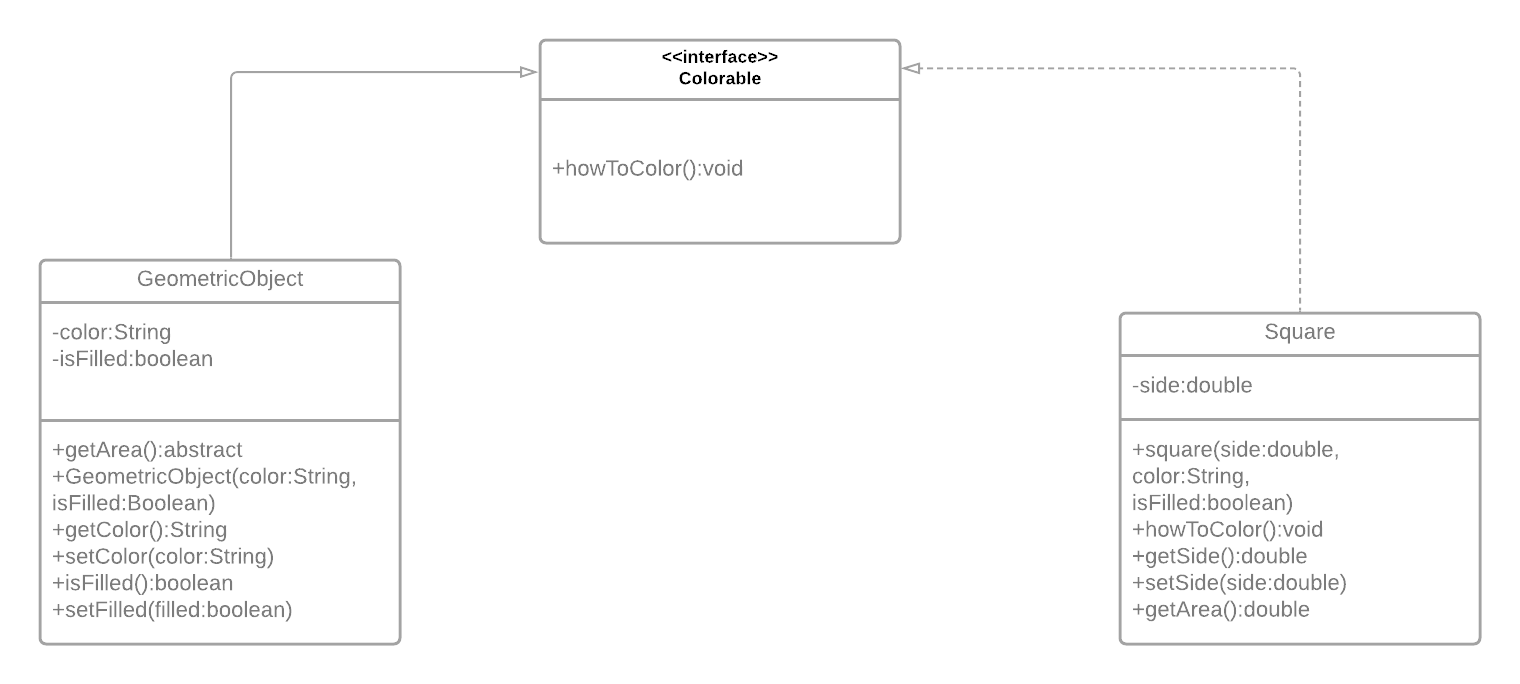
Explanation-

An interface named Colorable is created and void method called howToColor is used to construct Colorable (). The class GeomentricObject is then formed, along with an abstract double method getArea(). Then there's the class Square, which extends GeomentricObject and implements Colorable is created, with attributes for size1 and size2, and then a Driverclass named Test is created, with all of these methods and the print command.

|  |
| --- |
| /\*  \* 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 Question02;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public interface Colorable {  void howToColor();  }  /\*  \* 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 Question02;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public abstract class GeometricObject {  private String color;  private boolean isFilled;  public abstract double getArea();  public GeometricObject(String color, boolean isFilled) {  this.color = color;  this.isFilled = isFilled;  }  public String getColor() {  return color;  }  public void setColor(String color) {  this.color = color;  }  public boolean isFilled() {  return isFilled;  }  public void setFilled(boolean filled) {  isFilled = filled;  }  }  /\*  \* 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 Question02;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Square extends GeometricObject implements Colorable {  private double side;  public Square(double side, String color, boolean isFilled) {  super(color, isFilled);  this.side = side;  }  @Override  public void howToColor() {  System.out.println("Color all four sides");  }  public double getSide() {  return side;  }  public void setSide(double side) {  this.side = side;  }  public double getArea() {  return side\*side;  }  }  /\*  \* 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 Question02;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Test {  public static void main(String[] args) {  GeometricObject[] geometricObjects = new GeometricObject[5];  geometricObjects[0] = new Square(5, "red", true);  geometricObjects[1] = new Square(2, "blue", false);  geometricObjects[2] = new Square(3, "green", false);  geometricObjects[3] = new Square(10, "yellow", true);  geometricObjects[4] = new Square(25, "purple", false);  for (GeometricObject object: geometricObjects) {  System.out.println("By Rohit Reddy Chandupatla");  System.out.println("Area: " + object.getArea());  if (object instanceof Colorable) {  System.out.print("How to Color: ");  ((Colorable) object).howToColor();  }  }  }  } |



UML DIAGRAM -



1. (10-Points) What is casting? What are different types of casting? Explain and demonstrate with examples.

ANSWER

Explanation-

The determination of the importance of one primitive data form to another is known as typecasting. Upcasting and Downcasting are the two forms of casting in Java, as follows:

Downcasting -

Downcasting is the technique used when the subclass sort refers to the parent class's object. If it is done directly, the compiler will throw a ClassCastException at runtime, which will result in an error. It can only be done with the aid of the instance of operator. Only downcast can be done on an entity that has already been upcast.

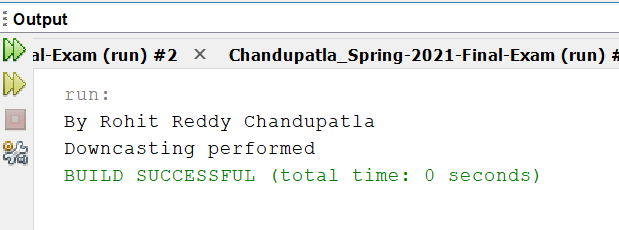
To conduct class style casting, we must adhere to the following two rules:

1.IS-A-Relationship classes are required.

2.A property of a class in which it will cast must be present in an entity.

Example for Downcasting – When Subclass named car and vehicles type refers to the object is created If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime

|  |
| --- |
| /\*  \* 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 Question03.Downcasting;  import java.io.\*;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  // Class 1  // Parent class  public class Vehicles {      }  /\*  \* 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 Question03.Downcasting;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  class Car extends Vehicles {  static void method(Vehicles v)  {    //  if (v instanceof Car) {    // Downcasting  Car c = (Car)v;    // Display message  System.out.println("Downcasting performed");  }  }    }  /\*  \* 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 Question03.Downcasting;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  // Creating an object of Vehicle class  // and referring it to Car class  System.out.println("By Rohit Reddy Chandupatla");  Vehicles v = new Car();  Car.method(v);  }  } |



Upcasting:

Casting a subtype to a super type in an upward direction in the inheritance tree is known as upcasting. When a sub-class object is referred by a superclass reference variable, it is an automated process for which no effort is expended. It is comparable to dynamic polymorphism.

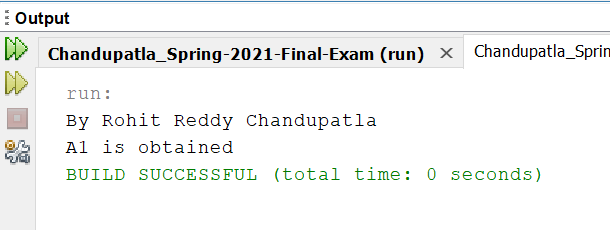
a. Implicit casting is the typecasting of classes by the compiler without the use of cast syntax.

b. Explicit casting is when a programmer uses cast syntax to typecast a class.

Example for Upcasting-

Firstly two classes named one and two are created where each of them are as a variable and class two is extended class one, then as at last all these are called in the driver class and upcasting is performed.

|  |
| --- |
| /\*  \* 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 Question03.Upcasting;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class One {  void A1()  {  System.out.println("A1 is obtained");  }  }  /\*  \* 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 Question03.Upcasting;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Two extends One {  void B2()  {  System.out.println("B2 is obtained");  }  }  /\*  \* 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 Question03.Upcasting;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here    {  System.out.println("By Rohit Reddy Chandupatla");  One o = (One)new Two();  o.A1();    }  }    } |



1. (15-Points) Suppose that Fruit, Apple, Orange, GoldenDelicious, and McIntosh are defined in the following inheritance hierarchy:

Fruit

Orange

Apple

GoldenDelicious

McIntosh

Assume that the following code is given:

Fruit fruit = new GoldenDelicious();

Orange orange = new Orange();

Answer the following questions and explain why these Statements are legal or illegal.

1. Is fruit instanceof Fruit?

Answer- Yes,Instance of subclass is also an instance of super class.

1. Is fruit instanceof Orange?

Answer- No, it super class instance can’t always be sub-class.

1. Is fruit instanceof Apple?

Answer- We know that GoldenDelicious is the sub class of Apple.Instance of sub class is an instance of super class

1. Is fruit instanceof GoldenDelicious?

Answer- Yes,fruit is an instanceof GoldenDelicious

1. Is fruit instanceof McIntosh?

Answer - No, as fruit is not an instance of McIntosh since it not a superclass of GoldenDelicious(Apple is superclass).fruit contains an instance if GoldenDelicious

1. Is orange instanceof Orange?

Answer- Yes, orange is instanceof Orange.

1. Is orange instanceof Fruit?

Answer- Yes, Orange object is an instance of Orange. Orange is a sub class of Fruit so orange instanceof Fruit

1. Is orange instanceof Apple?

Answer- No, here Apple is not an super class of Orange (object orange contains instance of class Orange)

1. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method?

Answer- Fruit can invoke method orange but orange cannot as orange is not an instance of Apple whereas fruit is an instance of Apple.

1. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method?

Answer- The object orange is an instance of the class Orange therefore, orange can invoke makeOrangeJuice method whereas fruit is not an instance of Orange so it cannot.

1. Is the statement Orange p = new Apple() legal?

Answer- Illegal Statement, instance of Apple cannot be assigned to object of orange class because Apple is not a sub class of Orange

1. Is the statement McIntosh p = new Apple() legal?

Answer- Illegal Statement,super class(Apple) instance cannot be assigned to sub-class reference(McIntosh)

m. Is the statement Apple p = new McIntosh() legal?

Answer- legal Statement, Apple is the super-class for class McIntosh and assignment of the instance of a sub-class to an object of the super class is possible.

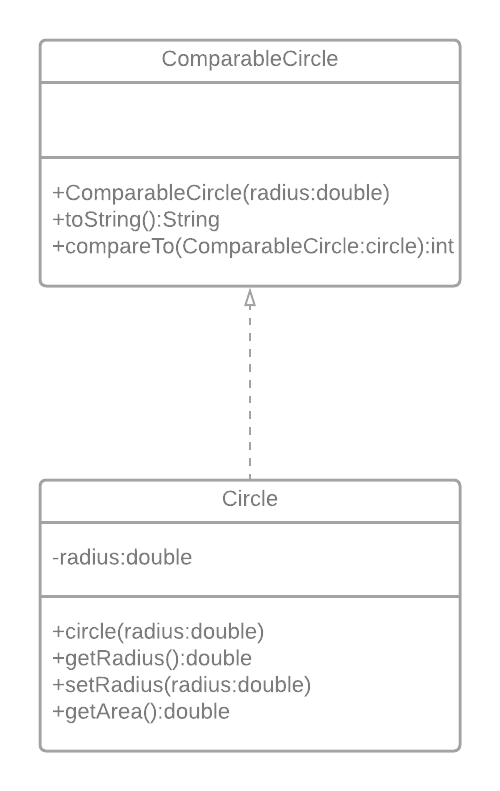
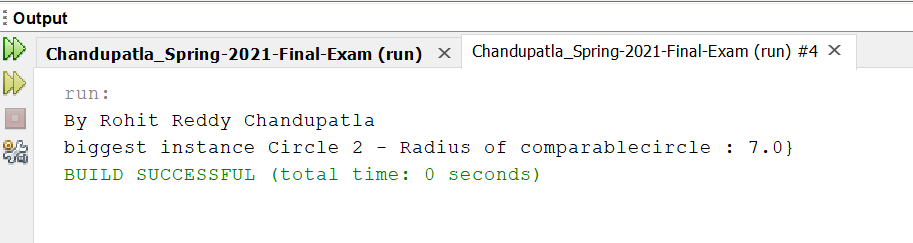
1. (10-Points) Define a class named ComparableCircle that extends Circle and implements Comparable. Draw the UML diagram and implement the compareTo method to compare the circles on the basis of area. Write a test class to find the larger of two instances of ComparableCircle objects.

ANSWER

Explanation:

First, a circle class is created with methods for getting the area and radius, and then a ComparableCircle class is created with methods for comparing the radius and area of the circles. Then, in a driver class, all of these methods are named, and finally, print statements are provided to give us the 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 Question05;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Circle {  private double radius;  public Circle(double radius) {  this.radius = radius;  }  public double getRadius() {  return radius;  }  public void setRadius(double radius) {  this.radius = radius;  }  public double getArea() {  return Math.PI \* radius \* radius;  }  }  /\*  \* 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 Question05;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class ComparableCircle extends Circle implements Comparable<ComparableCircle> {  public ComparableCircle(double radius) {  super(radius);  }  @Override  public String toString() {  return "Radius of comparablecircle : " + getRadius() + "}";  }  @Override  public int compareTo(ComparableCircle circle) {  return Double.compare(getArea(), circle.getArea());  }  }  /\*  \* 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 Question05;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class TestDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ComparableCircle circle1 = new ComparableCircle(3);  ComparableCircle circle2 = new ComparableCircle(7);  System.out.println("By Rohit Reddy Chandupatla");  System.out.println("biggest instance " + findLargest(circle1, circle2));  }  public static String findLargest(ComparableCircle c1, ComparableCircle c2) {  switch (c1.compareTo(c2)) {  case -1: return "Circle 2 - " + c2.toString();  default: return "Circle 1 - " + c1.toString();  }  }  } |



1. (15-Points) What is an exception? What are checked and unchecked exceptions? Explain and demonstrate with examples.

ANSWER

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions.

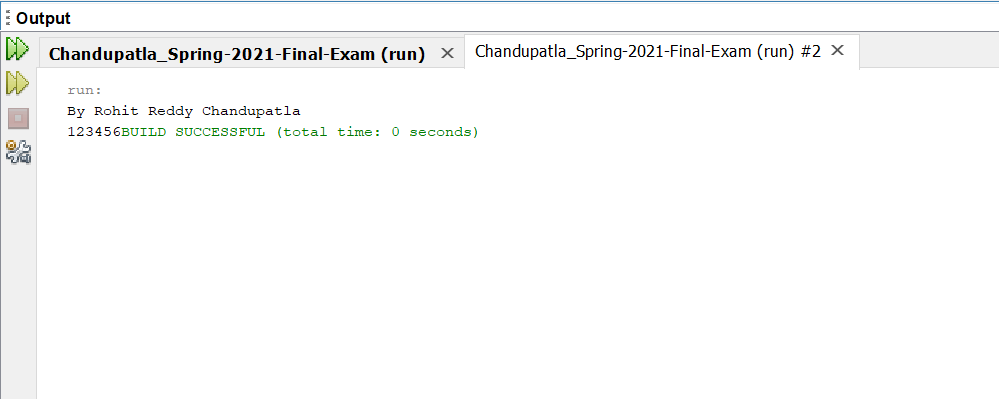
Checked exceptions-

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try catch-block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

Explanation: A Driver class named CheckedDriver is created where a code is given such that the input is taken from the input file kept in the netbeans in the project and in the package ,if the input file matches then the output is printed else exception is thrown.

|  |
| --- |
| /\*  \* 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 Question06;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class CheckedDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException, IOException {  // TODO code application logic here  FileInputStream fis = null;  /\*This constructor FileInputStream(File filename)  \* throws FileNotFoundException which is a checked  \* exception  \*/    System.out.println("By Rohit Reddy Chandupatla");  fis = new FileInputStream("question06.txt");  int k;  /\* Method read() of FileInputStream class also throws  \* a checked exception: IOException  \*/  while(( k = fis.read() ) != -1)  {  System.out.print((char)k);  }  /\*The method close() closes the file input stream  \* It throws IOException\*/  fis.close();  }  } |

Output:



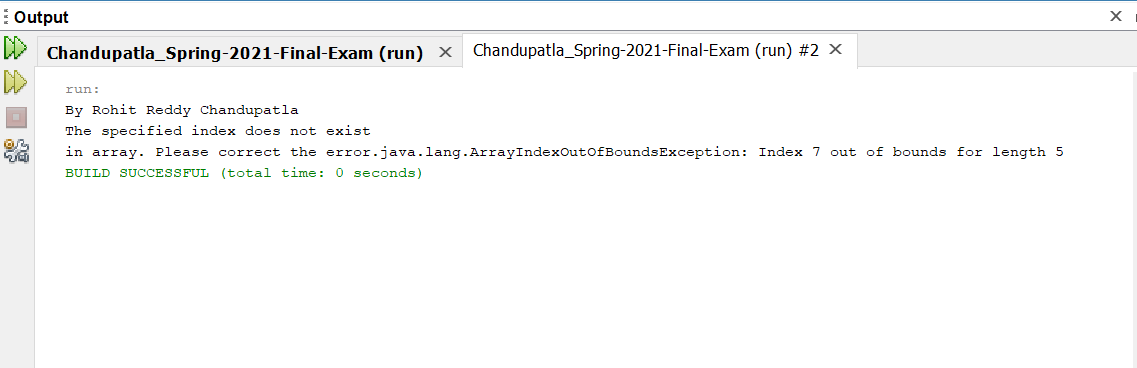
Unchecked exceptions-

Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **Runtime Exception** class.

Explanation: Initially a Unchecked driver is created When a number is selected from the array list, the output is printed by displaying the number; if the number is not in the array list, an ArrayIndexOutOfBoundsException ar is thrown, and an output Exception is thrown

|  |
| --- |
| /\*  \* 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 Question06;  /\*\*  \*  \* @author Rohit Reddy chandupatla  \*/  public class UncheckedDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  try{  int arr[] ={1,2,3,4,5};  System.out.println("By Rohit Reddy Chandupatla");  System.out.println(arr[7]);  }  catch(ArrayIndexOutOfBoundsException e){  System.out.println("The specified index does not exist ");  System.out.println("in array. Please correct the error."+e);  }  }  } |

Output:



1. (10-Points) Write a program that meets the following requirements:

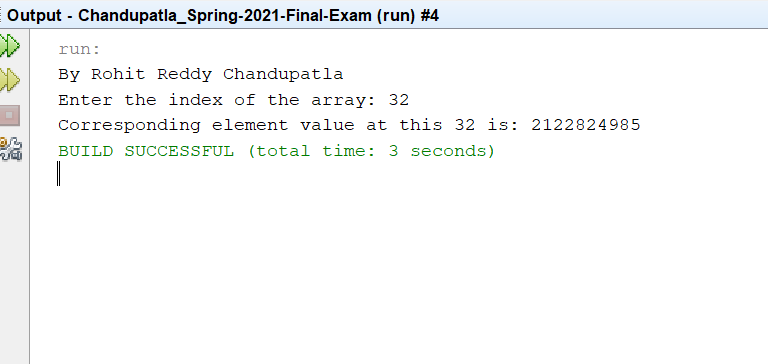
* Creates an array with 100 randomly chosen integers.
* Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds.

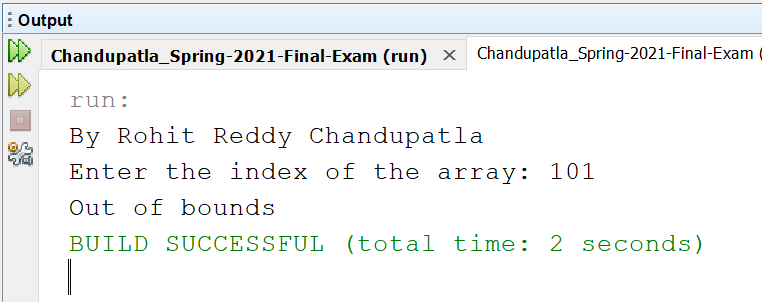
ANSWER

Explanation:

A driver class is created where an array of 100 index numbers is created and when a number is told to select , it checks if the number is between the index numbers of those 100 values and tells if the values is inside the bounds or outside the bounds.

|  |
| --- |
| /\*  \* 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 Question07;  import java.util.Arrays;  import java.util.Random;  import java.util.Scanner;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Random rand=new Random();  int[] array = new int[100];  for (int i = 0; i < array.length; i++) {  array[i] = rand.nextInt(1000)+1;  }  Scanner nacs=new Scanner(System.in);  System.out.print("By Rohit Reddy Chandupatla");  System.out.print("Enter the index of the array: ");  int index=nacs.nextInt();  try{  System.out.println("Corresponding element value at this "+index+" is: "+array[index]);  }catch(ArrayIndexOutOfBoundsException ex){    System.out.println("Out of bounds");    }    }  } |





1. (10-Points) What is the purpose of declaring exceptions? How do you declare an exception, and where? Can you declare multiple exceptions in a method header? Explain and demonstrate with examples.

ANSWER

Declaring exceptions informs the Java runtime system of what could go wrong. The throws keyword in the process declaration is used to declare an exception. Multiple exceptions may be declared, separated by commas.

If a procedure does not manage a verified exception, the throws keyword must be used to assert it. The keyword throws is found at the end of a method's signature. The throw keyword can be used to throw an exception, either a newly instantiated one or one that you have just captured.

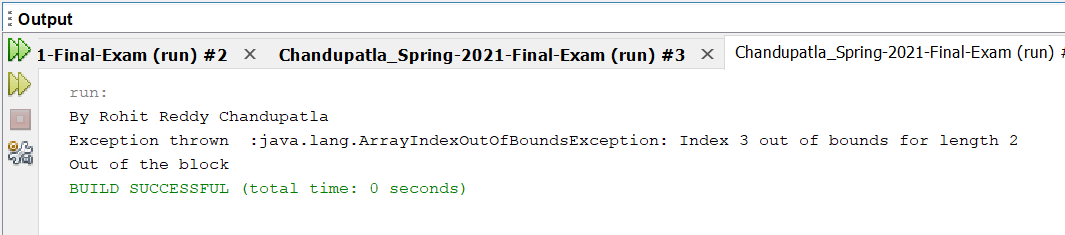
Yes, several exceptions may be declared in a method header. If the method declares several exceptions, follow throws with a list of the exceptions, separated by commas.

Explanation

In driver class, firstly given array size as 2, if at all the user gives the array size as 3 while execution, in side the program it catches a exception and throws Array Index Out Of Bounds Exception. In the below program, I have given the array size as 3 and that’s why the output is out of the block along with the thrown exception.

|  |
| --- |
| /\*  \* 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 Question08;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class ExcepTest {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  try {  int a[] = new int[2];  System.out.println("By Rohit Reddy Chandupatla");  System.out.println("Access element three :" + a[3]);  } catch (ArrayIndexOutOfBoundsException e) {  System.out.println("Exception thrown :" + e);  }  System.out.println("Out of the block");  }  } |

Output-



Example2:

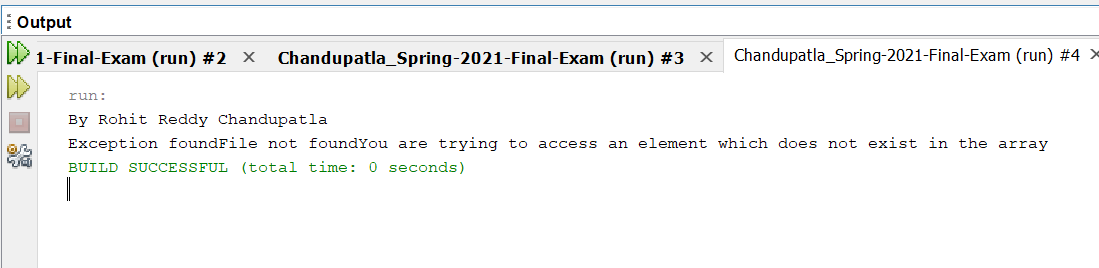
Explanation: In driver class, array size given as 10, and index are given from 1 to 5.

If the input file given is wrong then, it throws exception as file not found, using try and catch keywords.

Similarly, given the index as 7, then it throws array index out of bounds, as the index declaration at the beginning given was only till 5.

Therefore, the above are multiple exceptions using try keyword

|  |
| --- |
| /\*  \* 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 Question08;  import java.io.File;  import java.io.FileReader;  import java.io.IOException;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class MultipleException {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws Exception,IOException {  // TODO code application logic here  {  int x=10,res;  int arr[]={1,2,3,4,5};  try  {  res=x/0;  }  catch(Exception e){  System.out.println("By Rohit Reddy Chandupatla");  System.out.print("Exception found");  }  try  {  File file = new File("file.txt");  FileReader fr = new FileReader(file);  }  catch(Exception FileNotFoundException)  {  System.out.print("File not found");  }  try  {  System.out.print(arr[7]);  }  catch(Exception ArrayIndexOutOfBound){  System.out.println("You are trying to access an element which does not exist in the array");  }  }  }    } |



1. (10-Points) What is the keyword throw used for? What is the keyword throws used for? Can you throw multiple exceptions in one throw statement? Explain with examples.

ANSWER

The throws keyword is used to specify which exceptions a method may throw, while the throw keyword is used to throw an exception directly within a method or block of code. In a method signature, the throws keyword specifies which exceptions may be thrown by the method.

Yes, we can use multiple exceptions in a single throw statement.

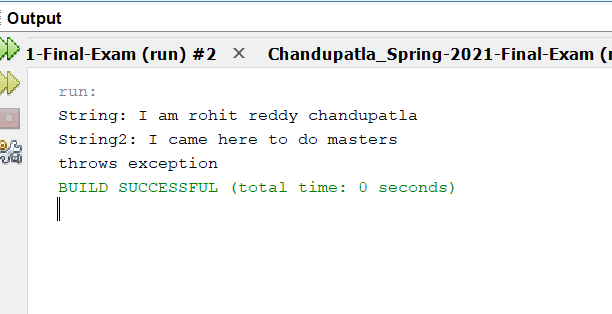
When our code throws several exceptions, you can either use a different try block for each statement that could throw an exception or use a single try block for all statements that could throw an exception. For multiple statements that can throw multiple exceptions, use a single try block.

Example for throw exception:

Explanation: A class named UsingThrow is created where two Strings are given and the all the getter and setter methods for these strings are given along with IndexOutOfBoundsException exception is thrown, then a driver class is created and the methods created in the regular class are called here and the two strings are checked, and both seen if the cross the bound of the size of an array and later that enters the try catch statement and prints the exception thrown 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 Question09;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class UsingThrow {  private String String1;  private String String2;  public UsingThrow(String String1, String String2) {  this.String1 = String1;  this.String2 = String2;  }  public String getString1() {  return String1;  }  public void setString1(String String1) {  this.String1 = String1;  }  public String getString2() {  return String2;  }  public void setString2(String String2) {  this.String2 = String2;  }    public String throwException() {  String string = " ";  if (String1.length()>String1.indexOf(String1) ){  throw new StringIndexOutOfBoundsException("Exception thrown");  } else {  string = "Out of bounds";  }  return string;  }  @Override  public String toString() {  return "String: " + String1 + "\nString2: " + String2;  }  }  /\*  \* 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 Question09;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class ThrowDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here    String String1="I am rohit reddy chandupatla";  String String2="I came here to do masters";  UsingThrow ut =new UsingThrow( String1, String2);  try {  System.out.println(ut.toString());  System.out.println(ut.throwException());  } catch (Exception e) {  System.out.println("throws exception");  }  }  } |

Output

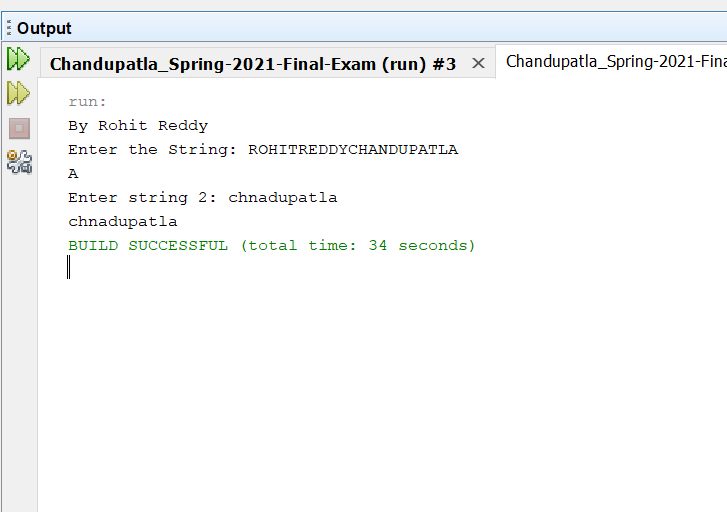


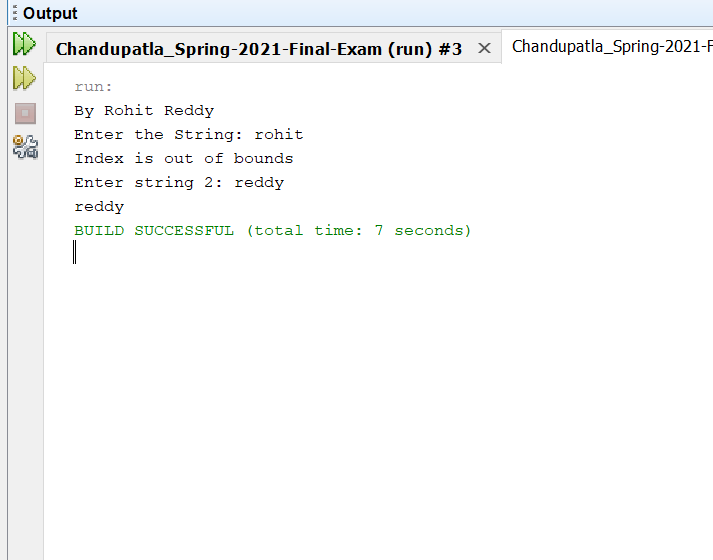
Example2:

Explanation: In main class named ThrowsDriver is created where two strings supposed to be taken from the keyboard using the scanner object, then Stringindexoutofbound exception is given where the string at index 12 if present is show else out of bound is displayed.

|  |
| --- |
| /\*  \* 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 Question09\_Throws;  import java.util.Scanner;  /\*\*  \*  \* @author S542423  \*/  public class ThrowsDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws StringIndexOutOfBoundsException{  // TODO code application logic here    Scanner sc=new Scanner(System.in);  System.out.println("By Rohit Reddy");  System.out.print("Enter the String: ");  String st1=sc.next();  try{    System.out.println(st1.charAt(12));  }catch(Exception e){  System.out.println("Index is out of bounds");  }    String st=null;  System.out.print("Enter string 2: ");    String st2=sc.next();    try{  if("rohit".equals(st)){  System.out.println("same");  }  else  System.out.println(st2);  }catch(Exception e){  System.out.println("null pointer exception");        }  }    } |

Output:





1. (15-Points) What is a recursive method? What is an infinite recursion? Explain and demonstrate with examples. Implement the search (element) in a list using recursion.

ANSWER

The technique of making a function call itself is known as recursion. This technique allows you to break down complex problems into smaller, easier-to-solve problems.

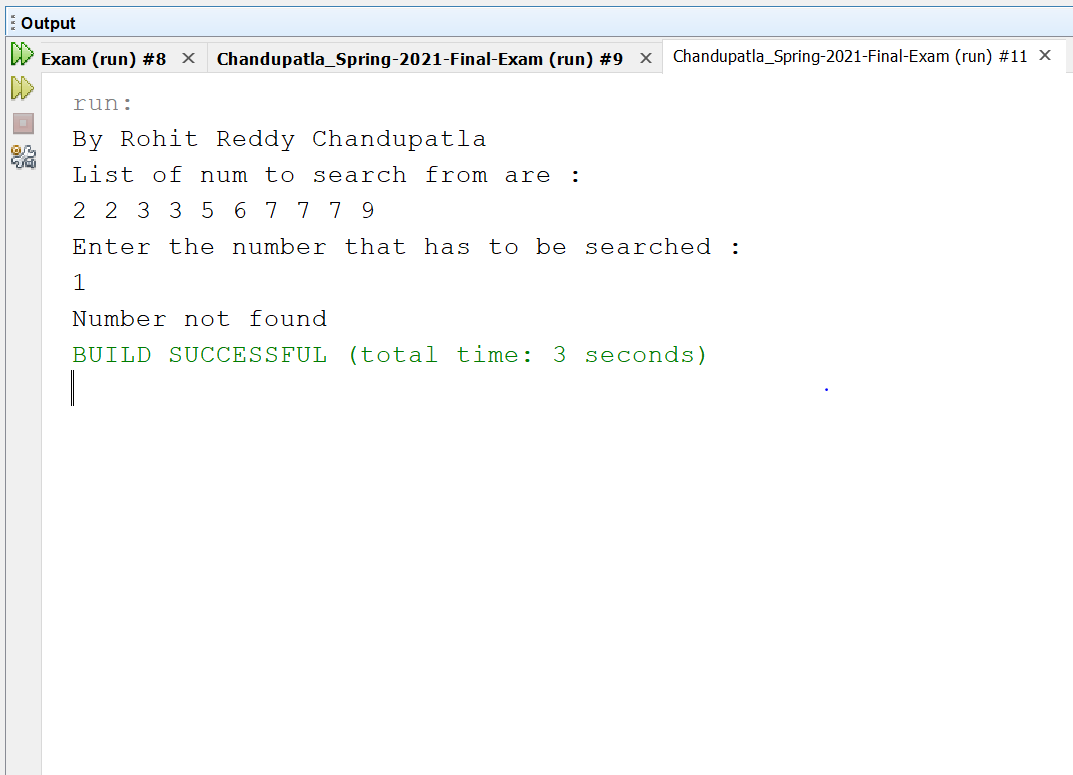
Infinite Recursion: Iteration and recursion will go on forever: with iteration, an infinite loop occurs if the loop-continuation test never becomes false; with recursion, infinite recursion occurs if the recursion stage does not converge on the base case.

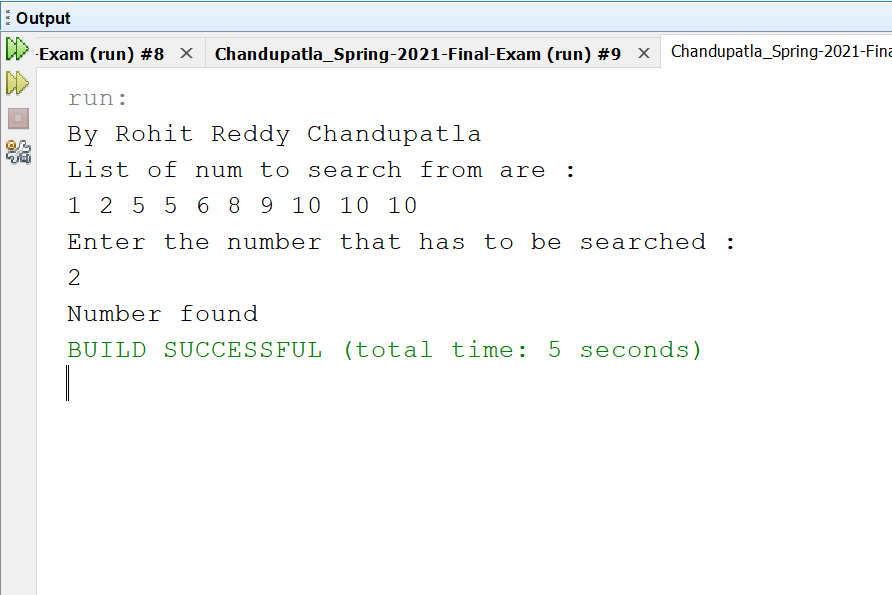
BinaryRecursion Example:

Explanation:

A class named BinaryRecursion is created where a raray list is given of size 10, then random elements are sent into the array and each of then checked if it is present in the array or no. If the element is present then the number is shown ,tif the element is not present then the element is not shown.

|  |
| --- |
| /\*\*  \*  \*/  package Question10;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Scanner;  /\*\*  \* @author Rohit Reddy Chandupatla  \*  \*/  public class BinaryRecursion {  /\*\*  \*  \* @param args  \*/  public static void main(String args[]) {  ArrayList<Integer> list = new ArrayList<>();  while (list.size() < 10) {  list.add((int) ((Math.random() \*10) +1));  }  // binary Search works only on sorted elements  Collections.sort(list);  System.out.println("By Rohit Reddy Chandupatla");  System.out.println("List of num to search from are :");  for (int j = 0; j < list.size(); j++) {  System.out.print(list.get(j) + " ");  }  Scanner sc = new Scanner(System.in);  System.out.println("\nEnter the number that has to be searched:");  int x = sc.nextInt();  sc.close();  boolean result = binarySearch(list, 0, list.size() - 1, x);  if (result)  System.out.println("Number found");  else  System.out.println("Number not found");  }  // method for binarySearch given the arraylist using recursion, left, right and  // search number  /\*\*  \*  \* @param list  \* @param l  \* @param r  \* @param x  \* @return  \*/  public static boolean binarySearch(ArrayList<Integer> list, int l, int r, int x) {  if (r >= l) {  int center = l + (r - l) / 2;  // If the number is present at the center  if (list.get(center) == x) {  return true;  }  // If number is smaller than center  if (x < list.get(center)) {  return binarySearch(list, l, center - 1, x);  }  // If number is greater than center  else {  return binarySearch(list, center + 1, r, x);  }  }  // We reach here when the number is not present  return false;  }  } |





InfiniteRecursion:

Explanation:

A main Driver class is created named infiniteRecursion where loop keeps running as shown in the output. To demonstrate infinite recursion, we used a factorial example.

To begin, infiniteRecursion is a function.

The main() function invokes Factorial(num).

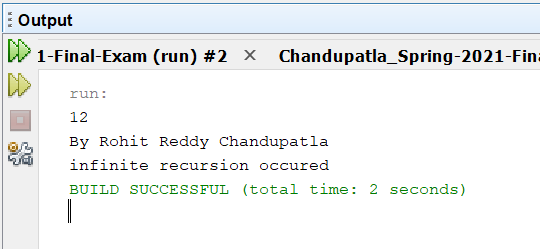
We set the num to 12, and it begins to follow the method, but since there is no condition to stop it, it continues to run. We can see from the above illustration that the process is never-ending. This occurs because the process (infiniteRecursionFactorial(int num)) lacks a base condition.

The condition is not fulfilled at any point in this situation.

Since there is no endpoint, recursive calls can be made indefinitely.

|  |
| --- |
| /\*  \* 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 Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class infiniteRecursion {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner sc = new Scanner(System.in);  int num = sc.nextInt();  System.out.println("By Rohit Reddy Chandupatla");  try{  long factorial = infiniteRecursion(num);    System.out.println(num + "! = " + factorial);  sc.close();  }catch(StackOverflowError st){  System.out.println("infinite recursion occured");  }  // example for infinite recursion  }  /\*\*  \*  \* @param num  \* @return  \*/  public static long infiniteRecursion(int num) {  return num \* infiniteRecursion(num - 1);  }  } |

Output-



1. (10-Points) Write a java program that illustrates how equals() and hashCode() methods work? Explain your code in comments.

ANSWER

equals() method

In java equals() method is used to compare equality of two Objects. The equality can be compared in two ways:

Shallow comparison: The default implementation of equals method is defined in Java.lang.Object class which simply checks if two Object references (say x and y) refer to the same Object. i.e. It checks if x == y. Since Object class has no data members that define its state, it is also known as shallow comparison.

Deep Comparison: Suppose a class provides its own implementation of equals() method in order to compare the Objects of that class w.r.t state of the Objects. That means data members (i.e. fields) of Objects are to be compared with one another. Such Comparison based on data members is known as deep comparison

hashCode() method

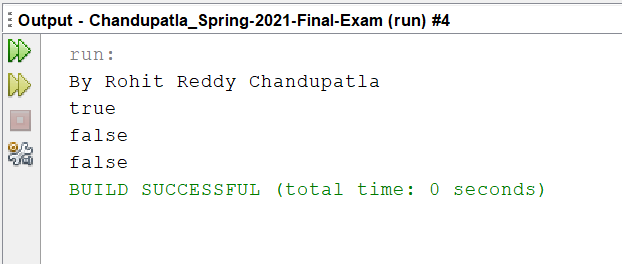
It returns the hashcode value as an Integer. Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable….etc. This method must be overridden in every class which overrides equals() method

Explanation:

A class named Hashcode is created where attributes name, age weight are given and then using our equals() and hashcode() methods the values given in the driver class are checked with the values in our regular class and told if they are equal or not.

|  |
| --- |
| /\*  \* 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 Question11;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Equal\_Hashcode {  private String name;  private int age;  private int weight;    /\*\*  \*  \* @param name will take the name  \* @param age will take the age  \* @param weight will take the weight  \*/  public Equal\_Hashcode(String name, int age, int weight) {  this.name = name;  this.age = age;  this.weight = weight;  }  /\*\*  \*  \* @param will check if the values are equal  \* @return will return if they are true or false after comparing the true or false  \*/  @Override  public boolean equals(Object o) {  if (this == o) {  return true;  }  if (o == null || getClass() != o.getClass()) {  return false;  }  Equal\_Hashcode Rohit = (Equal\_Hashcode) o;  return age == Rohit.age &&  weight == Rohit.weight &&  name.equals(Rohit.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 Question11;  /\*\*  \*  \* @author Rohit Reddy chandupatla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("By Rohit Reddy Chandupatla");  System.out.println(new Equal\_Hashcode("simmi", 17, 50)  .equals(new Equal\_Hashcode("simmi",17,50)));    System.out.println(new Equal\_Hashcode("Abhishek", 21, 90)  .equals(new Equal\_Hashcode("El Barto", 10, 45)));    System.out.println(new Equal\_Hashcode("anni", 60,45)  .equals(new Object()));  }    } |

Output-



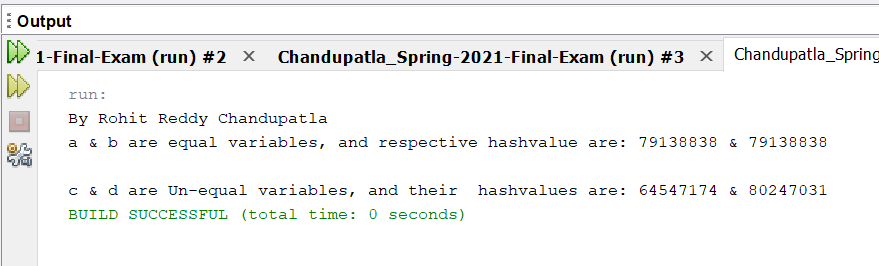
Example 2-

Explanation:

Initially a driver class named Test\_Hash is created where I gave strings named a and b in the beginning and both of the values are equal, and then they are checked by using hashcode() and equals() and will be told if equal or no. The same with the other two string values and are those two values are actually given

|  |
| --- |
| /\*  \* 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 Question11;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Test\_Hash {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here    /\*\*  \* two strings are initialised and declared  \*/  String a = "Rohit";  String b = "Rohit";    System.out.println("By Rohit Reddy Chandupatla");  /\*\*  \* two strings are compared and checked with equal() and the hashcode for each if them will be given  \*/  if(a.equals(b)){ //checking the equality of objects using equals() methods  System.out.println("a & b are equal variables, and respective hashvalue are:" + " "+ a.hashCode() + " & " + b.hashCode());    }    String c = "Bunny";  String d= "Sunny";  /\*\*  \* two strings are compared and checked with equal() and the hashcode for each if them will be given  \*/  if(!c.equals(d)){ //checking the equality of objects using equals() method  System.out.println("\nc & d are Un-equal variables, and their hashvalues are:" + " "+ c.hashCode() + " & " + d.hashCode());    }    }    } |

Output-



1. (15-Points) Design Employee class and Employee driver class as follows:
2. **Employee Class implements Comparable<Employee**>

* Data fields named empId, empName and empSalary
* A constructor with parameters, listed in the same order as above.
* Create getter methods for all the parameters.
* A toString method that prints the empId, empName and empSalary. There should be one space between each value output.
* Because Employee implements the Comparable interface, you must also implement the compareTo method as defined by the Comparable interface. Define this method in such a way that the natural ordering of employees will be by id number, in ascending order.

1. **EmployeeDriver Class**

* Begin by filling an ArrayList with at least 5 employees. Add employees in random order – not by id number, not by name, and not by salary. The original list should not be in order by any of these attributes.
* Use an enhanced for loop to print the original list.
* Call the one-parameter sort method of the Collections class to sort the list by its natural order (empId number) and then print the list again.
* Call the two-parameter sort method of the Collections class, supplying a new Comparator<Employee> that sorts by salary. Print the list again.
* Call the two-parameter sort method of the Collections class, supplying a new Comparator<Employee> that sorts by name. Print the list again.

Explanation:

In the Employee class is responsible for implementing Where the variables empId, employeeName, and empSalary are used. The constructors for all the variables were then written, as well as getter and setter methods.

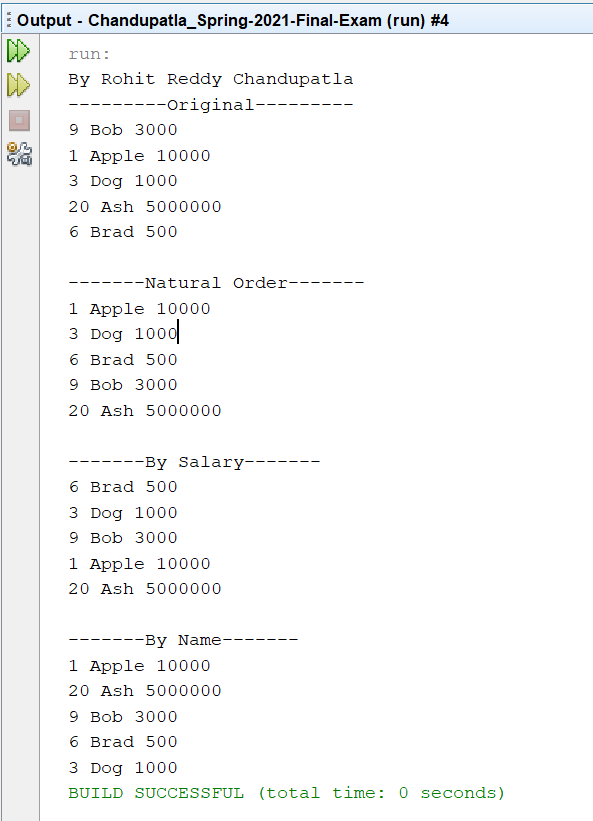
The toString methods for those variables have been published. The compareTo approach was then overridden for natural empId ordering.

The five employee descriptions were added to an array list called employee List in the driver class, which was then printed in order using the print statement. If employee equals the letter ‘s,' the letter ‘s' is printed.

Then, using the sorting method, if emp1 salary is less than emp2, it returns -1, if they are equal, it returns 0, and otherwise, it returns +1.

The sala is then printed based on the sorting.

|  |
| --- |
| /\*  \* 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 Question12;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class Employee implements Comparable<Employee> {  private int empId;  private String empName;  private int empSalary;  public Employee(int empId, String empName, int empSalary) {  this.empId = empId;  this.empName = empName;  this.empSalary = empSalary;  }  public int getEmpId() {  return empId;  }  public void setEmpId(int empId) {  this.empId = empId;  }  public String getEmpName() {  return empName;  }  public void setEmpName(String empName) {  this.empName = empName;  }  public int getEmpSalary() {  return empSalary;  }  public void setEmpSalary(int empSalary) {  this.empSalary = empSalary;  }  @Override  public String toString() {  return empId + " " + empName + " " + empSalary;  }  @Override  public int compareTo(Employee o) {  return Integer.compare(getEmpId(), o.getEmpId());  }    }  /\*  \* 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 Question12;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  /\*\*  \*  \* @author Rohit Reddy Chandupatla  \*/  public class EmployeeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ArrayList<Employee> arrayList = new ArrayList();    arrayList.add(new Employee(9, "Bob", 3000));  arrayList.add(new Employee(1, "Apple", 10000));  arrayList.add(new Employee(3, "Dog", 1000));  arrayList.add(new Employee(20, "Ash", 5000000));  arrayList.add(new Employee(6, "Brad", 500));    System.out.println("By Rohit Reddy Chandupatla");  System.out.println("---------Original---------");  for (Employee employee: arrayList) {  System.out.println(employee);  }  Collections.sort(arrayList);  System.out.println("\n-------Natural Order-------");  for (Employee employee: arrayList) {  System.out.println(employee);  }  Comparator<Employee> bySalary = Comparator.comparing(Employee::getEmpSalary);  Collections.sort(arrayList, bySalary);  System.out.println("\n-------By Salary-------");  for (Employee employee: arrayList) {  System.out.println(employee);  }  Comparator<Employee> byName = Comparator.comparing(Employee::getEmpName);  Collections.sort(arrayList, byName);  System.out.println("\n-------By Name-------");  for (Employee employee: arrayList) {  System.out.println(employee);  }      }  } |



Link to Git Hub Repo-