**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.

**ABSTRACT CLASS:**

A Java abstract class is a class which cannot be instantiated, meaning you cannot create new instances of an abstract class. The purpose of an abstract class is to function as a base for subclasses. The abstract class can have the abstract methods.

**INTERFACE:**

A Javainterface is a bit like a [Java class](http://tutorials.jenkov.com/java/classes.html), except a Java interface can only contain method signatures and fields. A Java interface is not intended to contain implementations of the methods, only the signature (name, parameters and exceptions) of the method.

**SIMILARITIES:**

1. Object cannot be created in Abstract class and Interfaces.
2. Abstract class and Interfaces can contain abstract method.
3. Abstract methods written in the abstract classes as well as interfaces must be implemented by extending class.
4. Constants can be declared in both the abstract classes and interfaces.

**DIFFERENCES:**

1. Abstract class can be extended using abstract keyword. Whereas Interface can be implemented using implements keyword.
2. Abstract class can have both abstract and non-abstract methods. But Interface can have only abstract methods.
3. Abstract class can have constructors and Interface does not constructors.
4. Abstract can have static, non-static, final and non-final variables. Interface can have only static final variables.

Example for Abstract Class

EXPLANATION: The abstract class automotive is created in which abstract method automobile is written and a void method vehicle is written to print the statement. Then a boat class is created which is the sub class of automotive and implements the abstract methods and over ride it and prints a statement.Then car class is created which is the sub class of the automotive and implements the abstract methods and over rides those methods and in the driver class an object is created since car is a sub class of automotive the statement is legal and the methods were called and again other object for boat is created and the methods were called and printed.

**Automotive.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 question1a;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  Public abstract class automotive {  public abstract void automobile();  public void vehicle() {  System.out.println("Types of vehicles ");  }  } |

**Boat.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 question1\_abstract;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class boat extends automotive {  @Override  public void automobile() {  System.out.println("THIS runs on water ");  }  } |

**Car.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 question1a;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class car extends automotive {  public void vehicle() {  System.out.println("IT IS A FOUR WHEEL VEHICLE");  }  @Override  public void automobile() {  System.out.println("This is automotive abstract method");  }  } |

**Driver 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 question1\_abstract;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Answer by MNAOJ NUVVALA question 1");  automotive car1 = new car();  car1.automobile();  car1.vehicle();  automotive boat1 = new boat();  boat1.automobile();  }  } |

OUTPUT:



Example for interface :

Here the two interfaces circle and parabola were created conic sections is the class that implements both methods of the interfaces and the methods were over ridden and in the driver class methods are invoked and printed by creating the object and the input is given by scanner class

Circle.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 question1\_multipleinterface;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public interface Circle {  double perimeterOfCircle();  default int getAngle(int angle) {  int angleInSemi = angle / 2;  return angleInSemi;  }  } |

parabola.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 question1\_multipleinterface;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public interface Parabola {  double area();  default double directrix(int a) {  double directrixOfParabola = -a;  return directrixOfParabola;  }  } |

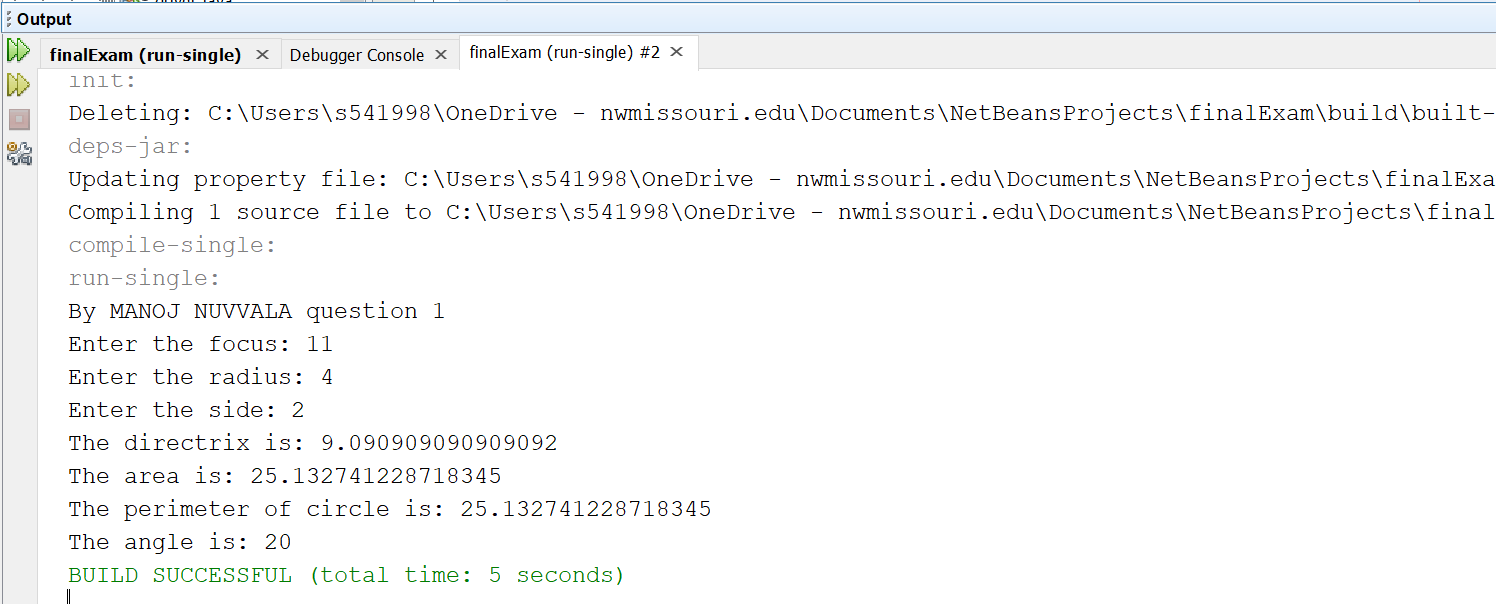
conicsections.java

|  |
| --- |
| /\*  \* 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 question1\_multipleinterface;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public class ConicSections implements Parabola, Circle {  private int focus;  private double radius;  private double side;  public ConicSections(int focus, double radius, double side) {  this.focus = focus;  this.radius = radius;  this.side = side;  }  public int getFocus() {  return focus;  }  public double getRadius() {  return radius;  }  public double getSide() {  return side;  }  @Override  public double area() {  //throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.  return Math.PI \* radius \* side;  }  @Override  public double directrix(int a) {  return Math.pow(a, 2) / focus; //To change body of generated methods, choose Tools | Templates.  }  @Override  public double perimeterOfCircle() {  //throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.  return 2 \* Math.PI \* radius;  }  @Override  public int getAngle(int angle) {  return angle / 3; //To change body of generated methods, choose Tools | Templates.  }  } |

Conicdriver.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 question1\_multipleinterface;  import java.util.Scanner;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public class ConicDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner nacs = new Scanner(System.in);  System.out.print("Enter the focus: ");  int focus = nacs.nextInt();  System.out.print("Enter the radius: ");  double radius = nacs.nextDouble();  System.out.print("Enter the side: ");  double side = nacs.nextDouble();  ConicSections conic = new ConicSections(focus, radius, side);  System.out.print("The directrix is: ");  System.out.println(conic.directrix(10));  System.out.print("The area is: ");  System.out.println(conic.area());  System.out.print("The perimeter of circle is: ");  System.out.println(conic.perimeterOfCircle());  System.out.print("The angle is: ");  System.out.println(conic.getAngle(60));  }  } |

**Output :**



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.

Colorable.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 question2;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public interface Colorable {  public abstract void howToColor();  } |

GeometricObjects.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 question2;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public abstract class GeometricObjects implements Colorable {  public abstract double getArea();  } |

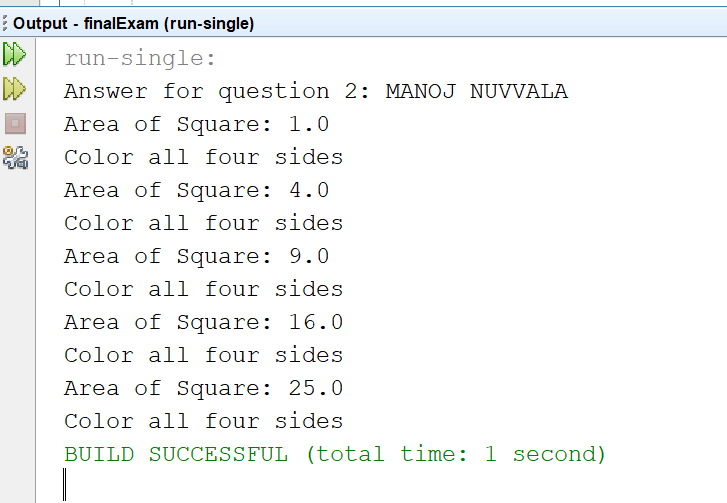
Square.java

|  |
| --- |
| /\*  \* 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 question2;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class square extends GeometricObjects implements Colorable {  private double side;  public square(double side) {  this.side = side;  }  @Override  public void howToColor() {  System.out.println("Color all four sides");  }  @Override  public double getArea() {  return side \* side;  }  } |

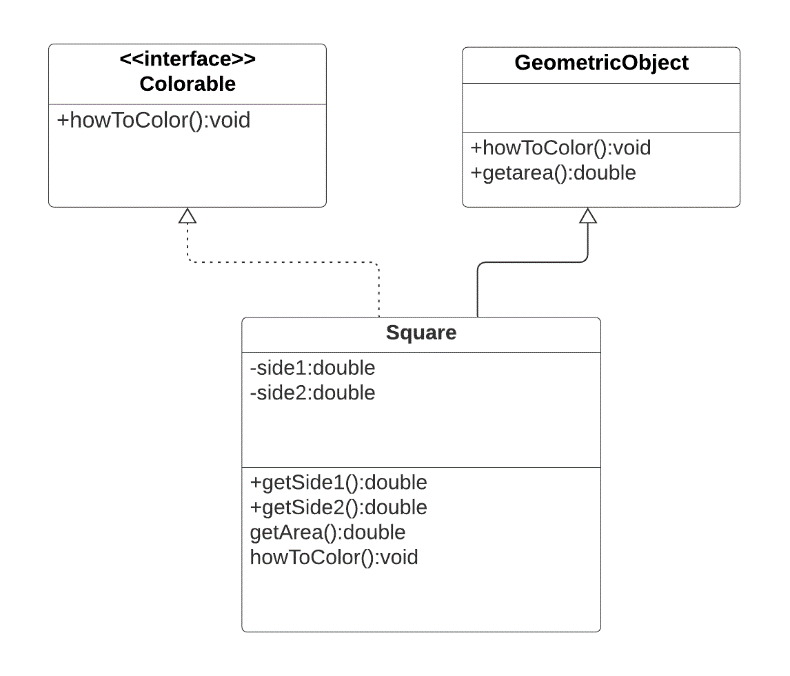
GeometricObjectsdriver.java

|  |
| --- |
| /\*  \* 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 question2;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class GeometricObjectsdriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  GeometricObjects go[] = new GeometricObjects[5];  GeometricObjects s1 = new square(1);  GeometricObjects s2 = new square(2);  GeometricObjects s3 = new square(3);  GeometricObjects s4 = new square(4);  GeometricObjects s5 = new square(5);  go[0] = s1;  go[1] = s2;  go[2] = s3;  go[3] = s4;  go[4] = s5;  System.out.println("Answer for question 2: MANOJ NUVVALA");  for (GeometricObjects g : go) {  System.out.println("Area of Square: " + g.getArea());  g.howToColor();  }  }  } |

**OUTPUT:**



UML DIAGRAM:



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

Casting is a process of changing one type value to another type. In Java, we can cast one type of value to another type. It is known as type casting.

In Java, type casting is classified into two types,

* Widening Casting (Implicit): This is type is used to change data type of one to another (lowest to the highest).

Int 🡪Long 🡪 Float 🡪 Double.

Widening Type casting take place when,

* + The two types are compatible
  + The target type is larger than the source type
* Narrowing Casting (Explicitly done): This is the vice versa of widening casting where it changes the data type one to another in a decreasing way.

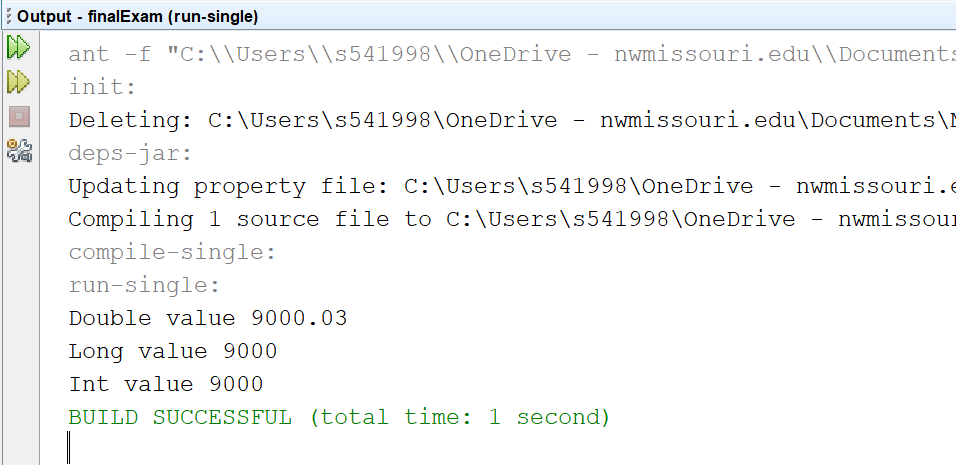
When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting. If we don't perform casting then compiler reports compile time error.

Double 🡪Float 🡪 Long 🡪 Int.

Example for narrowing casting: In this program we will narrow the primitive data types, we will be changing double data type to long and then to the int data type.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question3;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class narrowing {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  double d = 9000.03;  long l = (long) d;  int i = (int) l;  System.out.println("Double value " + d);  System.out.println("Long value " + l);  System.out.println("Int value " + i);  }  } |

Output :

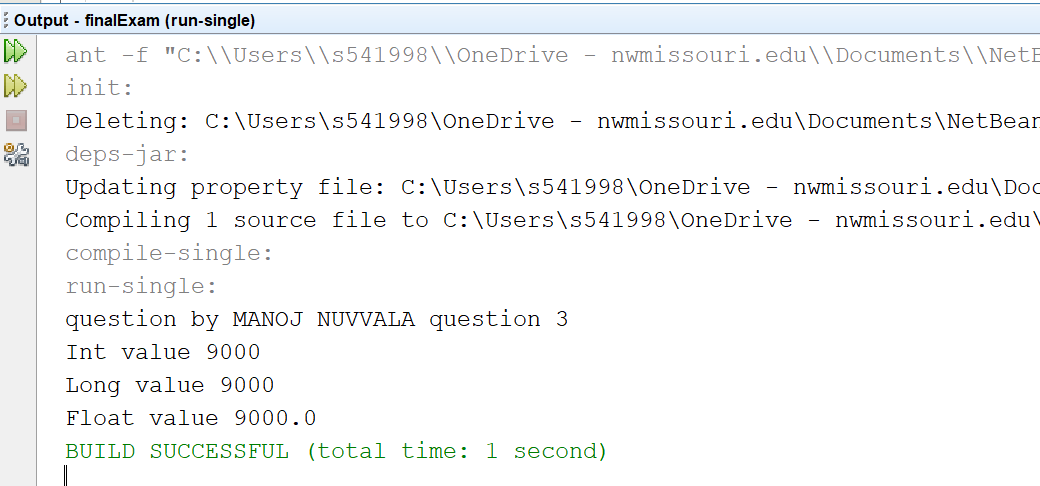


Example for narrowing :

In this example we will enhance data type from int to long then to double.

|  |
| --- |
| /\*  \* 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 question3;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class widening {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  int i = 9000;  long l = i;  float f = l;  System.out.println("Int value " + i);  System.out.println("Long value " + l);  System.out.println("Float value " + f);  }  } |

OUTPUT:



Example with Non -Primitive data types :

This example demonstrates object reference casting which casts a parent class to child class and vice-versa. We created a super class called Oil.java and sub class called EngineOil.java which extends Oil. In the driver method we try cast parent object to child object.

Oil o=eo;

This statement is an implicit conversion since a parent can hold the child’s object.

Engineoil engoil =(engineoil) eo;

This is an explcconversion

Engineoil engoil = (engineoil) oil;

This statement is an explicit conversion because we are trying to hold oil object by type casting it into EngineOil. Since we are converting it from child to parent it throws ClassCastException which we handle using try and catch.

Oil.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 question3.wib;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public class Oil {  private String oilType;  public Oil(String oilType) {  this.oilType = oilType;  }  public String getOilType() {  return oilType;  }  @Override  public String toString() {  return "THIS " + this.getOilType() + " used";  }  } |

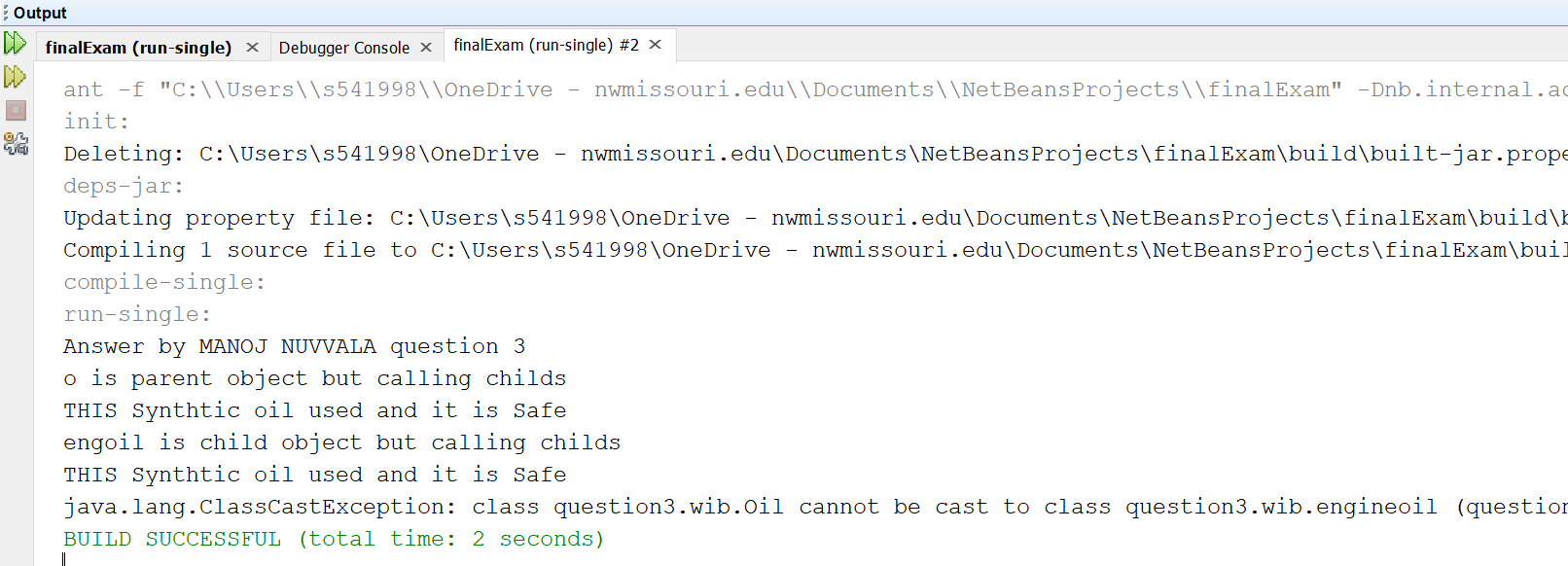
Engineoil.java

|  |
| --- |
| /\*  \* 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 question3.wib;  /\*\*  \*  \* @author MANOJ NUVVALA  \*/  public class engineoil extends Oil {  private boolean s4toil;  public engineoil(boolean s4toil, String oilType) {  super(oilType);  this.s4toil = s4toil;  }  public boolean isgood() {  return s4toil;  }  @Override  public String toString() {  String safe = "";  if (s4toil) {  safe = "Safe";  } else {  safe = "Not Safe";  }  return super.toString()  + " and it is "  + safe;  }  } |

Oildriver.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 question3.wib;  /\*\*  \*  \* @author Jeevan Bodigam  \*/  public class OilDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws ClassCastException {  // TODO code application logic here  System.out.println("Answer by MANOJ NUVVALA question 3");  Oil eo = new engineoil(true, "Synthtic oil");  Oil o = eo; //implicit conversion parent type will automatically hold child object  try {  System.out.println("o is parent object but calling childs");  System.out.println(o.toString());  System.out.println("engoil is child object but calling childs");  Oil oil = new Oil("4toil");  engineoil engoil = (engineoil) eo; //Explict COnversion of parent to child type  System.out.println(engoil.toString());  engineoil engoil1 = (engineoil) oil;  //we get an exception here because we are converting from parent to child  System.out.println(engoil1.toString());  } catch (ClassCastException e) {  System.out.println(e);  }  }  } |

OUTPUT:



1. (15-Points) Suppose that Fruit, Apple, Orange, Golden Delicious, 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.

a. Is fruit instanceof Fruit?

Yes, fruit is an instanceOf Fruit. Instance of subclass is also an instance of super class

b. Is fruit instanceof Orange?

The given Statement is illegal. fruit object is of type GoldenDelicious which is a subclass of Apple. Subclass objects will always be instance of direct parent classes.

c. Is fruit instanceof Apple?

The above statement is legal. fruit is an object of GoldenDelicious which is a subclass of Apple which is Parent class.

d. Is fruit instanceof GoldenDelicious?

The above statement is legal. fruit is an object of GoldenDelicious class which is object of itself.

e. Is fruit instanceof McIntosh?

The above statement is illegal. fruit is an object of GoldenDelicious which is direct subclass for Fruit. McIntosh is subclass of apple but there is no relation between McIntosh and GoldenDelicious.

f. Is orange instanceof Orange?

The above statement is legal. orange is an object of Orange class which direct object creation of the class. It is always an instance.

g. Is orange instanceof Fruit?

The above statement is legal. orange object is of Orange class which is subclass of parent class Fruit. Since a subclass is always instance of base class or super class.

h. Is orange instanceof Apple?

The above statement is illegal. Orange and Apple classes or subclasses of Fruit but orange object is of type Orange so orange cannot be instance of Apple which another subclass of parent. There is no relation between Apple and Orange.

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

fruit is an object of GoldenDelicious which is subclass of parent Apple. But the object created is of Fruit type which is a parent class. The given statement is polymorphic substitution, in which methods will be called from left hand side class in this case it is Fruit. So makeAppleCider method which is in Apple class cannot be invoked using fruit which is of type Fruit. Even orange object cannot invoke the method from the Apple because Orange and Apple classes are subclasses of Fruit, but they are no way related.

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

makeOrangeJuice () method is written in Orange class and orange is direct object creation of Orange hence object orange can invoke the given method whereas fruit which is instance of Apple and there is no relation between Orange and Apple. fruit cannot invoke makeOrangeJuice () method.

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

The above statement is illegal because Apple and Orange are two separate subclasses of the Fruit class. They inherit Fruit but Orange is not super class for Apple. For the above statement to be legal Orange has to be parent class.

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

The above statement is illegal because Apple is parent class of McIntosh which is subclass. Always a parent class can hold the instance of sub class but vice versa is not possible. This is against polymorphic substitution. This can be legal by casting.

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

The above statement is legal because McIntosh is child class and Apple is parent class. This way of parent class holding child’s object is known as polymorphic substitution.

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.

Cicle.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 question5;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class Circle {  private double radius;  public Circle(double radius) {  this.radius = radius;  }  public double getradius() {  return radius;  }  public double getpi() {  return Math.PI;  }  public double area() {  return getpi() \* radius \* radius;  }  @Override  public String toString() {  return "Circle with" + radius + "has area : " + area();  }  } |

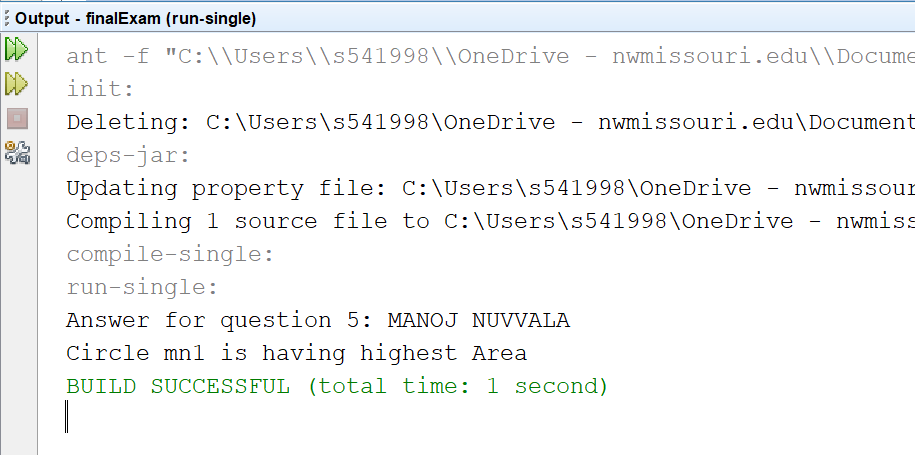
ComparableCircle.java

|  |
| --- |
| /\*  \* 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 question5;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class ComparableCircle extends Circle implements Comparable<Circle> {  public ComparableCircle(int radius) {  super(radius);  }  @Override  public int compareTo(Circle arg0) {  Circle mn1 = this;  Circle mn2 = arg0;  Integer Area1 = (int) mn1.area();  Integer Area2 = (int) mn2.area();  Integer Result = -(Area1.compareTo(Area2));  return Result;  }  } |

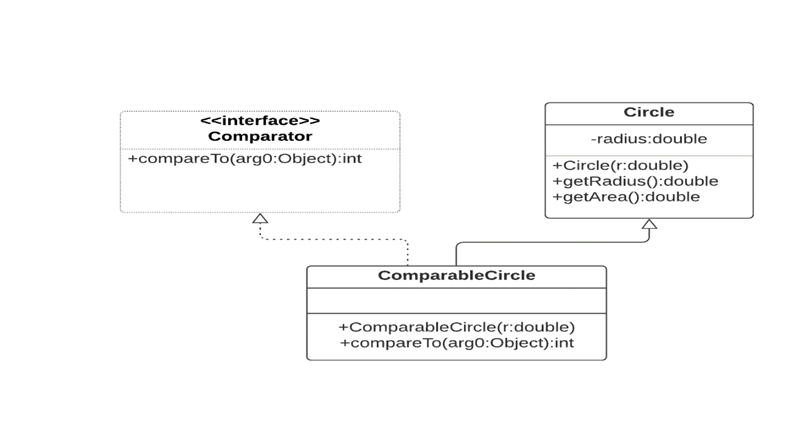
driver5.java

|  |
| --- |
| /\*  \* 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 question5;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class driver5 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ComparableCircle mn1 = new ComparableCircle(6);  ComparableCircle mn2 = new ComparableCircle(4);  System.out.println("Answer for question 5: MANOJ NUVVALA");  int result = mn1.compareTo(mn2);  if (result == -1) {  System.out.println("Circle mn1 is having highest Area");  } else if (result == 0) {  System.out.println("Area are same");  } else {  System.out.println("Circle mn2 is having highest Area");  }  }  } |

OUTPUT:



UML DIAGRAM:



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

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. In java By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message.

There are two types of exceptions in Java:  
1) Checked exceptions  
2) Unchecked exceptions

**Checked exceptions**

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error.

For example, SQLException, IOException, ClassNotFoundException etc.

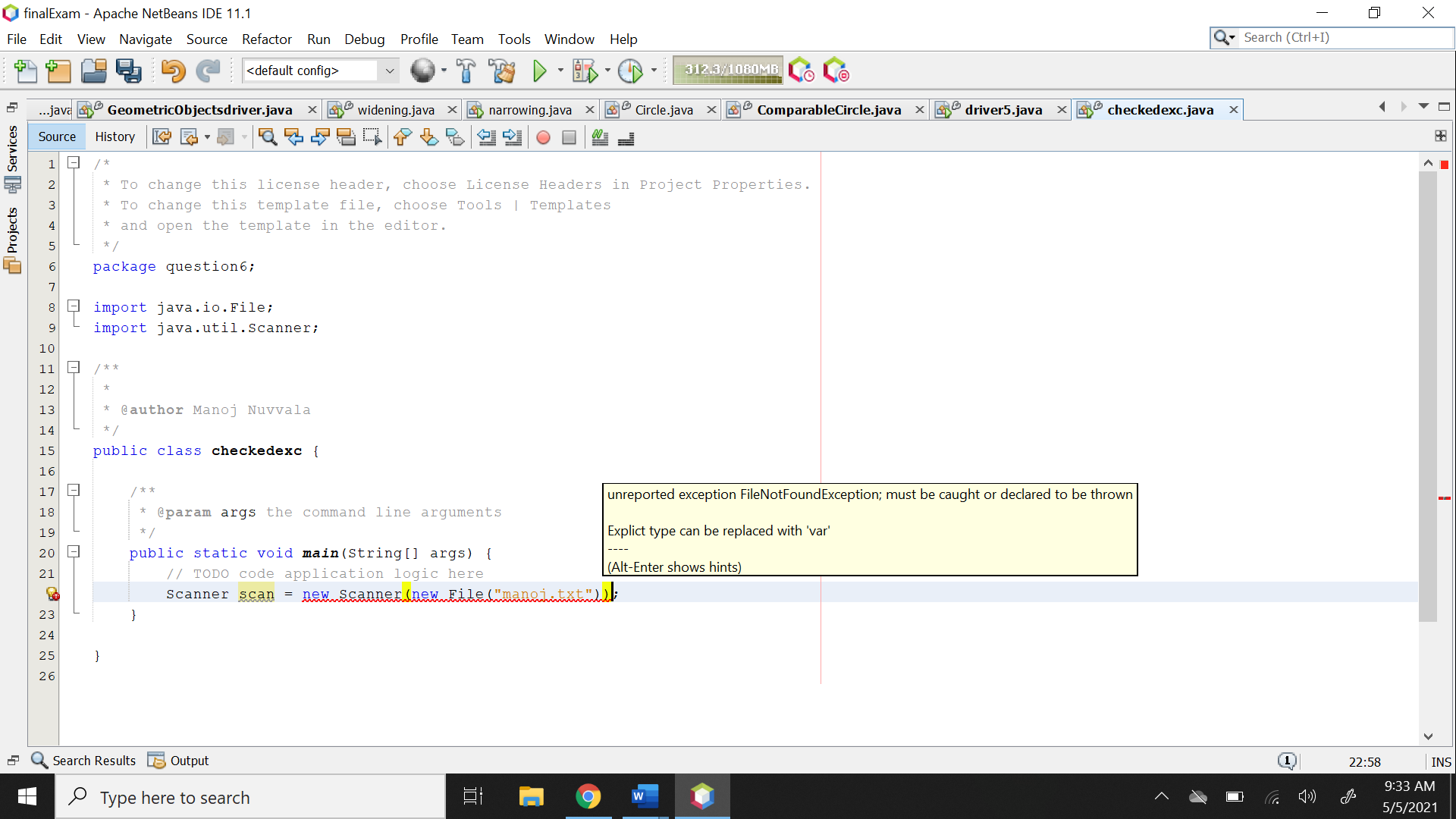
**Unchecked Exceptions**

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it’s the responsibility of the programmer to handle these exceptions and provide a safe exit.

For example,ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

**Example for Checked exceptions**

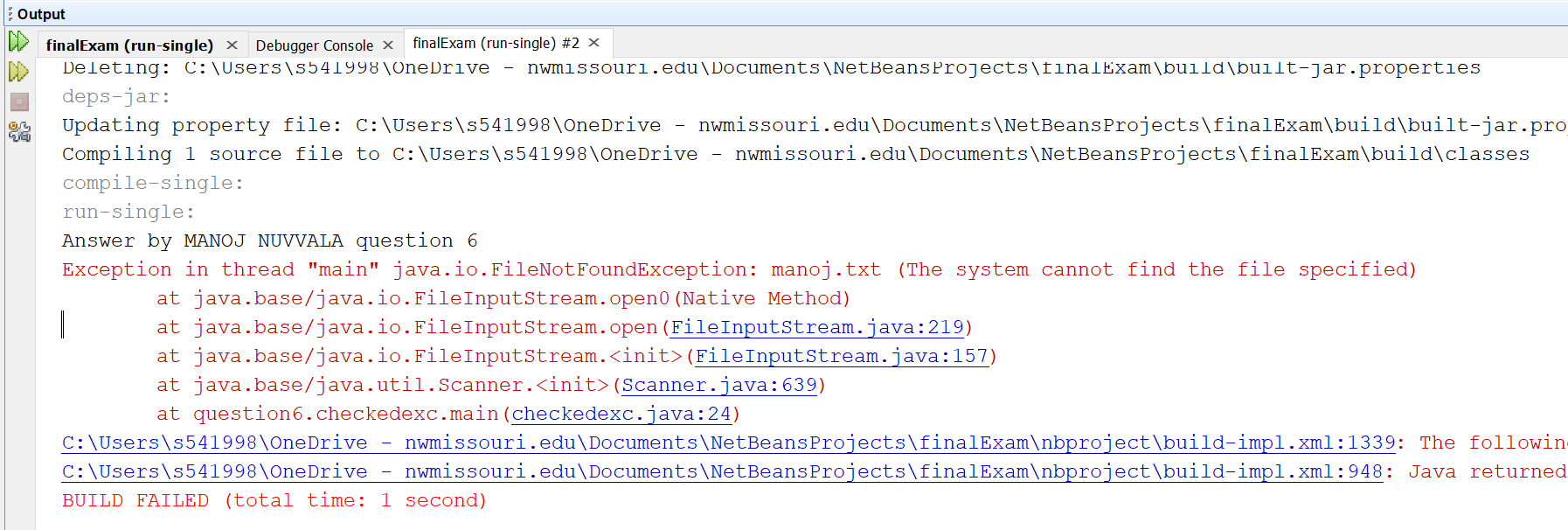
Let us illustrate the Checked exception FileNotFoundException with following examples. I have written the code to read the input from the text file “manoj.txt” which is not present in the project. The following error will occur if we do not handle the checked exception.



To handle this exception, we should through the file not found 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 question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class checkedexc {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException {  // TODO code application logic here  System.out.println("Answer by MANOJ NUVVALA question 6");  Scanner scan = new Scanner(new File("manoj.txt"));  }  } |

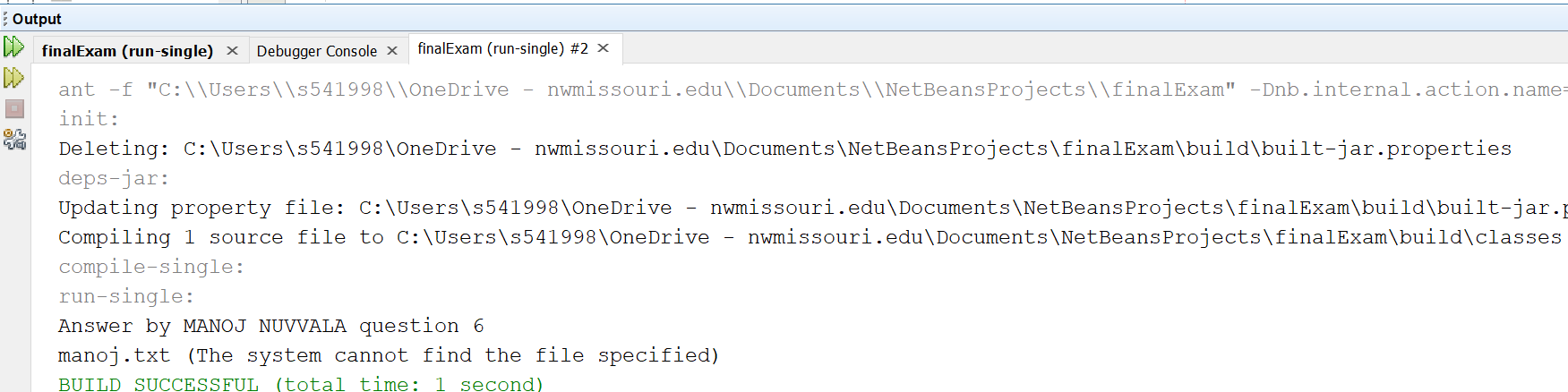
OUTPUT:



The other way to handle this exception is to catch the exception using try catch which is illustrated below.

|  |
| --- |
| /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class trycatch {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Answer by MANOJ NUVVALA question 6");  try {  Scanner scan = new Scanner(new File("manoj.txt"));  } catch (FileNotFoundException f) {  System.out.println(f.getMessage());  }  }  } |

OUTPUT:



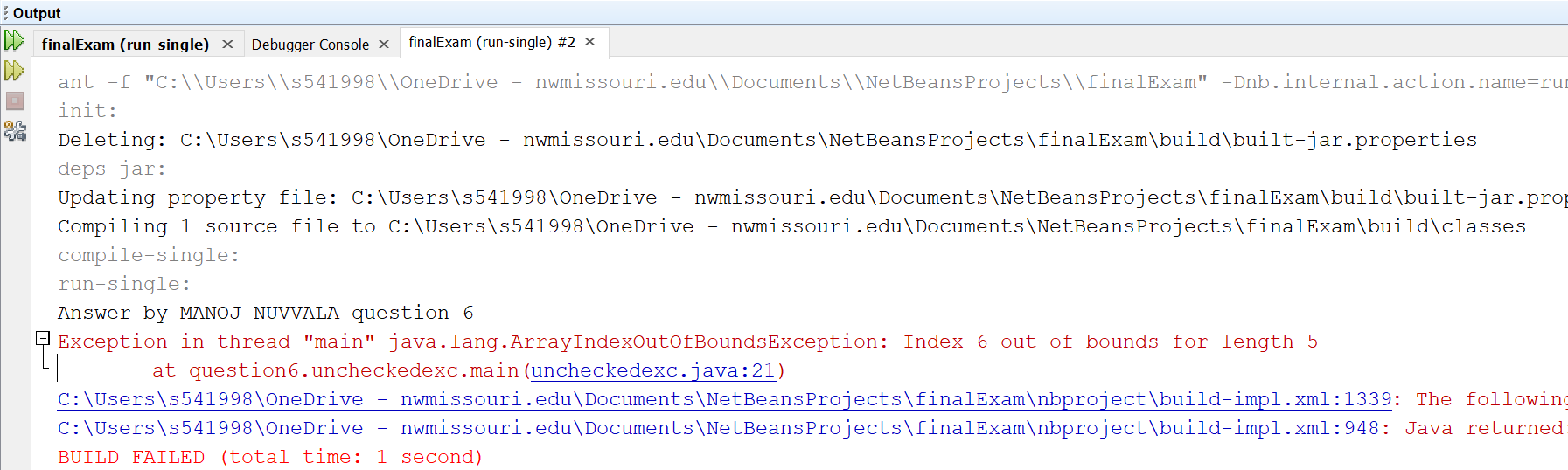
**Example for Unchecked exceptions**

Below are the examples which illustrates the unchecked exceptions, here in these examples we will illustrate IndexOutofBoundException.

Here, Example we are trying to access the index of an array which out of bound in that even we will get IndexOutofBoundException.

|  |
| --- |
| /\*  \* 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 question6;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class uncheckedexc {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Answer by MANOJ NUVVALA question 6");  int numbers[] = {1, 2, 3, 4, 5};  System.out.println(numbers[6]);  }  } |

OUTPUT:

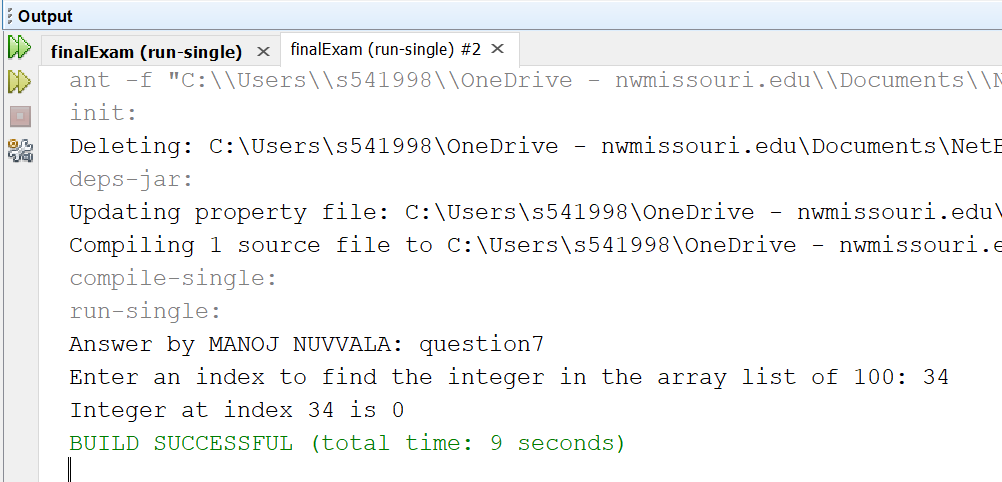


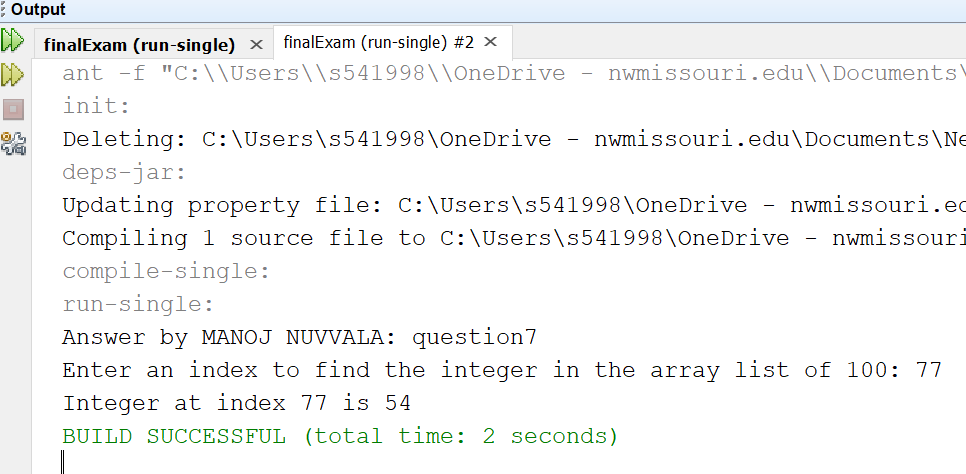
1. (10-Points) Write a program that meets the following requirements:

* 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.

|  |
| --- |
| /\*  \* 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 question7;  import java.util.Scanner;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class random100arrays {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  int numbers[] = new int[100];  for (int i = 0; i < numbers.length; i++) {  numbers[i] = (int) (Math.random() \* 100);  }  System.out.println("Answer by MANOJ NUVVALA: question7");  System.out.print("Enter an index to find the integer in the array list of 100: ");  Scanner scan = new Scanner(System.in);  int index = scan.nextInt();  try {  System.out.println("Integer at index " + index + " is "  + numbers[index]);  } catch (Exception m) {  System.out.println(m.getMessage());  }  }  } |

OUTPUT:





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.

The purpose of declaring exceptions is to tell the Java runtime system what can go wrong. You declare an exception using the throws keyword in the method declaration. You can declare multiple exceptions, separated by commas.

In the bellow scenarios we can use exceptions

When creating a custom exception. You can have a checked exception by extending the Exception class. If you want to have an unchecked exception we can extend RuntimeException class.

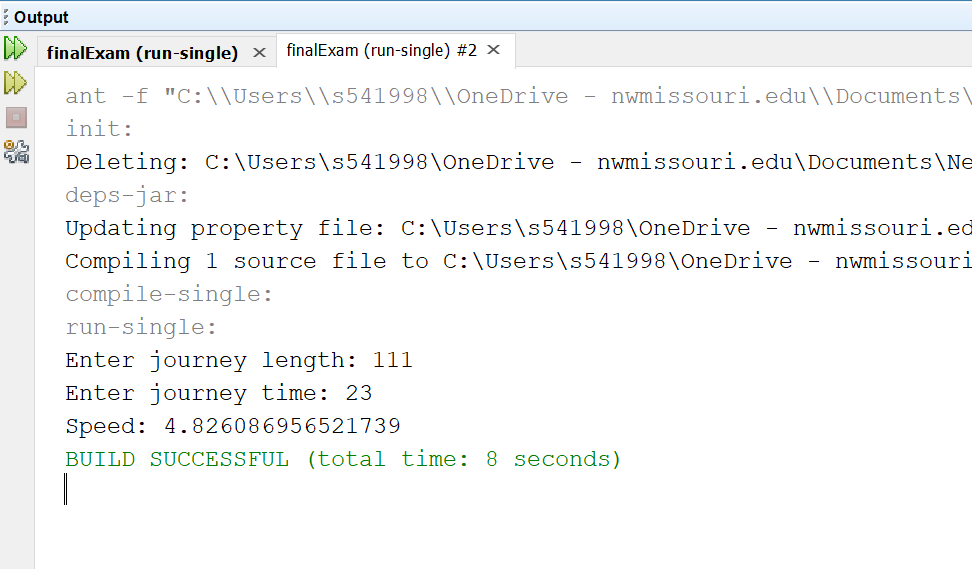
When catching the exception after try block in try catch. While throwing the exception you always throw a single exception in a single java statement but when catching the exception you can catch multiple exceptions or also use the parent/ super class which has multiple exception subclasses

In the method header if the method throws an exception. Within the method for custom exceptions or user thrown exceptions you’d use the throw keyword. Also Here you can declare multiple exceptions in the method header using the throws keyword.

In this example we are throwing ArithematicException and InputMismatchException where InputMismatchException occurs if we enter wrong input and ArithematicException occurs if we enter time as 0.

|  |
| --- |
| /\*  \* 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.example1;  import java.util.InputMismatchException;  import java.util.Scanner;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class FindSpeed {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws ArithmeticException,  InputMismatchException {  // TODO code application logic here  double length = 0;  double time = 0;  Scanner scan = new Scanner(System.in);  System.out.print("Enter journey length: ");  length = scan.nextDouble();  System.out.print("Enter journey time: ");  time = scan.nextDouble();  System.out.println("Speed: " + calculateSpeed(length, time));  }  public static double calculateSpeed(double length, double time) throws  ArithmeticException {  double speed = 0;  try {  speed = length / time;  } catch (ArithmeticException e) {  System.out.println(e);  }  return speed;  }  } |

OUTPUT:



In this example, I have taken an example of ballot where there are two attributes voterName,

Age. In this class setAge() method is written such a way that if age is less than 18, then exception is thrown saying not eligible for voting. In the Driver class we are throwing multiple exception which are ArithematicException and NullPointerException. NullPointerException occurs when we try to invoke Election class methods with null object of type Election. Arithematic exception occurs when we enter age less than 18.

Ballot.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 Manoj Nuvvala  \*/  public class BallotDriver {  public static void main(String[] args) throws ArithmeticException,  NullPointerException {  System.out.println("Answer for question 8: Raghavendra Reddy");  Ballot ele = new Ballot("rakesh");  ele.setAge(35);  System.out.println(ele);  Ballot ele1 = new Ballot("manoj");  ele1.setAge(20);  Ballot ele2 = null;  ele2.setAge(20);  }  } |

Ballotdriver.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 Manoj Nuvvala  \*/  public class BallotDriver {  public static void main(String[] args) throws ArithmeticException,  NullPointerException {  System.out.println("Answer for question 8: Raghavendra Reddy");  Ballot ele = new Ballot("rakesh");  ele.setAge(35);  System.out.println(ele);  Ballot ele1 = new Ballot("manoj");  ele1.setAge(20);  Ballot ele2 = null;  ele2.setAge(20);  }  } |

OUTPUT:



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.

**Keyword throw**: keyword throw is used to throw an exception explicitly inside a block of code. The keyword throw can be used only with unchecked exceptions.

**Keyword throws**: keyword throws a clause used in the method signature which is used to declare an exception which will be thrown by the block of the code while executing the code.

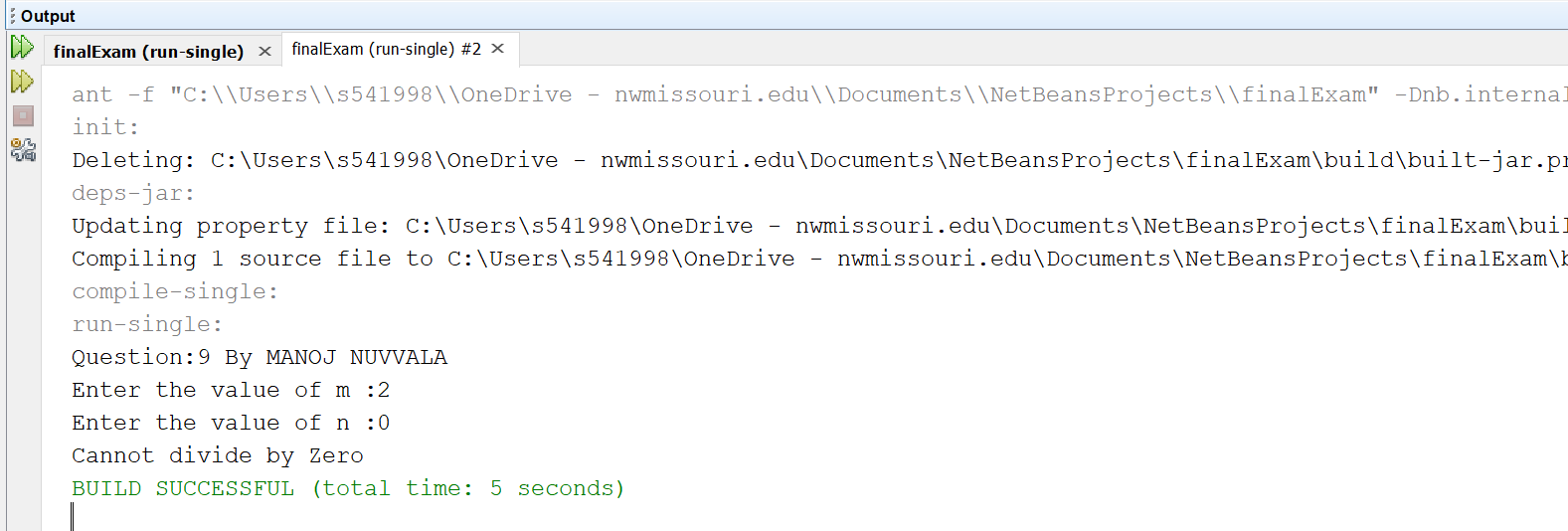
No, we cannot throw multiple exceptions at a time using a single throw statement.

The below program illustrates an example for throw keyword. In this example we are calculating the division of two numbers. When the denominator is equal to zero then the Arithmetic exception is thrown of message given by us.

throwexample.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;  import java.util.Scanner;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class throwexample {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Question:9 By MANOJ NUVVALA");  Scanner scan = new Scanner(System.in);  System.out.print("Enter the value of m :");  int m = scan.nextInt();  System.out.print("Enter the value of n :");  int n = scan.nextInt();  try {  if (n == 0) {  throw new ArithmeticException("Cannot divide by Zero");  }  int output = m / n;  System.out.println("Output : " + output);  } catch (ArithmeticException e) {  System.out.println(e.getMessage());  }  }  } |

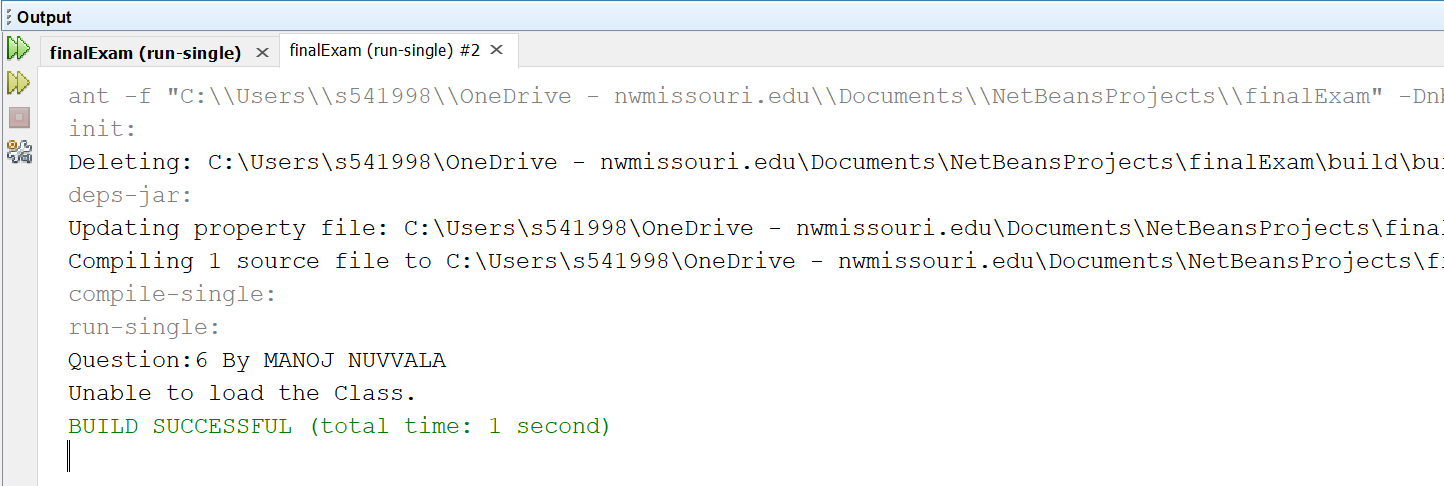
Output:



Let us take ClassNotFoundException to illustrate the throws clause in the program. I have written the code with class of different which is not present in the project. We use keyword throws in the method header to advertise that this method throws ClassNotFoundException if the class is not present in project.

|  |
| --- |
| /\*  \* 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 Manoj Nuvvala  \*/  public class throwsexample {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws ClassNotFoundException {  // TODO code application logic here  System.out.println("Question:6 By MANOJ NUVVALA");  try {  Class c = Class.forName("incorrectPackage.IncorrectClass");  } catch (Exception e) {  System.out.println("Unable to load the Class.");  }  }  } |

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.

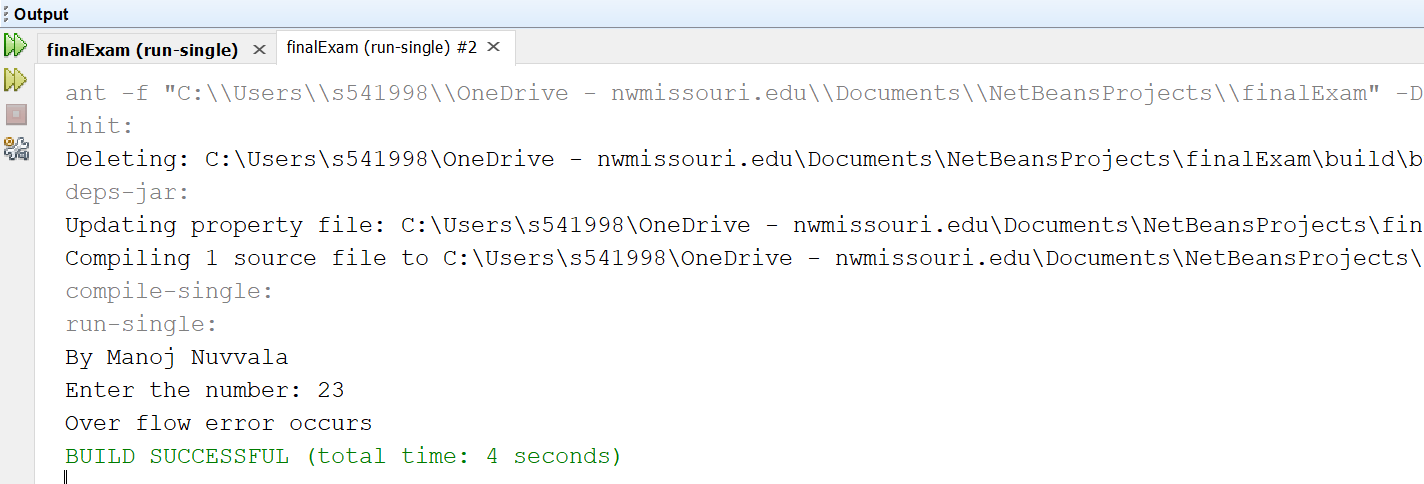
Recursive Method : A method invokes itself straightforwardly or by implication is called recursion and the specific method is called Recursive method.

Infinite Recursion : Recursive method invokes consistently and it is highly unlikely to stop the recursive calls is called infinite recursion.

In this program we first take a number from the user through scanner class, and the infinite recursion method is called which is written at the end of driver class such that the method invokes consistently and no condition is provided such that the method executes many number of times. If the input we have given is zero then it prints zero. If the input given is any number other than zero then infinite recursion occurs and we have thrown a stackoverflow error and printing the over flow error occurs.

|  |
| --- |
| /\*  \* 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 MANOJ NUVVALA  \*/  public class InfiniteRecursion {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("By Manoj Nuvvala");  Scanner scan = new Scanner(System.in);  try {  System.out.print("Enter the number: ");  int number = scan.nextInt();  long rec = infiniteRecursion(number);  System.out.println(number + "! = " + rec);  } catch (StackOverflowError st) {  System.out.println("Over flow error occurs");  }  }  // example for infinite recursion  public static long infiniteRecursion(int num) {  return num \* infiniteRecursion(num);  }  } |

Output:



In the Driver class a array is defined with size 5 and the numbers were given and by utilizing the scanner class an information is entered and a recursive method is written at the end of the driver class to search for an element in the array. In that method b<a returns -1 implies the component is absent in the array since the component has not found on both left and right most limit i.e., a and b ,now check for x. Next index!=-1 indicates that if index is found then the element exists or else the element is not present in the array.

|  |
| --- |
| /\*  \* 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 MANOJ NUVVALA  \*/  public class SearchingElement {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scan = new Scanner(System.in);  int numbersarray[] = new int[]{12, 145, 6, 60, 10};  System.out.println("By Manoj Nuvvala question 10");  System.out.print("Enter the element : ");  int element = scan.nextInt();  int indexInList = recursiveSearch(numbersarray, 0, numbersarray.length - 1, element);  if (indexInList != -1) {  System.out.println("Element " + element + " is present at index "  + indexInList);  } else {  System.out.println("Element " + element + " is not present in the list");  }  }  private static int recursiveSearch(int[] ary, int a, int b, int c) {  if (b < a) {  return -1;  }  if (ary[a] == c) {  return a;  }  if (ary[b] == c) {  return b;  }  return recursiveSearch(ary, a + 1, b - 1, c);  }  } |

Output;



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

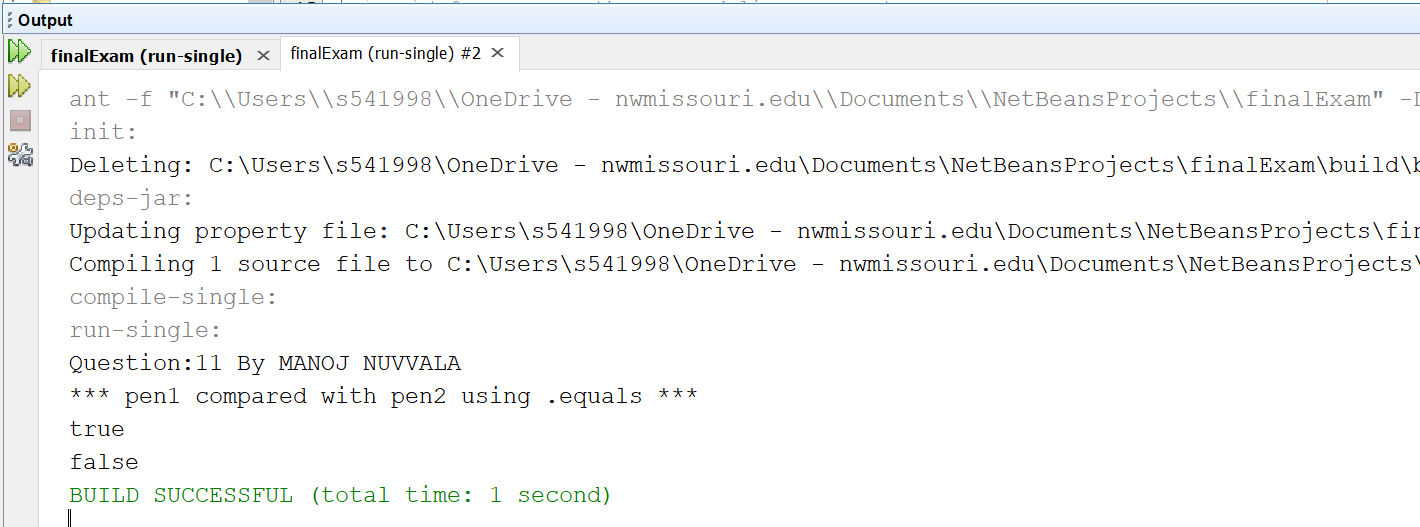
Pen.java

|  |
| --- |
| /\*  \* 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 question11;  import java.util.Objects;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class pen {  private String brand;  private String ink;  public pen(String brand, String ink) {  this.brand = brand;  this.ink = ink;  }  public String getBrand() {  return brand;  }  public String getInk() {  return ink;  }  @Override  public String toString() {  return "Pen{" + "brand=" + brand + ", ink=" + ink + '}';  }  @Override  public int hashCode() {  int hash = 7;  hash = 41 \* hash + Objects.hashCode(this.brand);  hash = 41 \* hash + Objects.hashCode(this.ink);  return hash;  }  @Override  public boolean equals(Object obj) {  /\*  Step 1: this check. if yes , then return true.  In this step the references of pen1 and pen2 are compared.If equal return true.  \*/  if (this == obj) {  return true;  }  /\*  Step 2: null check. if yes , then return false.  In this step,we will check if object is null.If null return false.  \*/  if (obj == null) {  return false;  }  /\*  Step 3: getClass check.if not equal then return false.  We need to check if both the objects being compared are of same class type or not.  In this example we will compare pen1 and pen2 are of same class type,if not will return false.  \*/  if (getClass() != obj.getClass()) {  return false;  }  /\*  Step 4: Typecast the object.  This step must be done after getClass check  \*/  final pen other = (pen) obj;  /\*  Step 5: Comparing Objects  Here we need to make a decision to consider the class member variables,  when comparing two objects of a class.  For example:  As per the below code,two pen objects are equal, only if ink and brand  member variables are equal.And will return false, if any one of them is not equal.  \*/  if (this.ink != other.ink) {  return false;  }  if (!Objects.equals(this.brand, other.brand)) {  return false;  }  return true;  }  } |

penDriver.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 question11;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class penDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Question:11 By MANOJ NUVVALA");  pen pen1 = new pen("Reynolds", "blue");  pen pen2 = new pen("Reynolds", "blue");  pen pen3 = new pen("Bic", "black");  System.out.println("\*\*\* pen1 compared with pen2 using .equals \*\*\*");  System.out.println(pen1.equals(pen2));  System.out.println(pen1.equals(pen3));  }  } |

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.

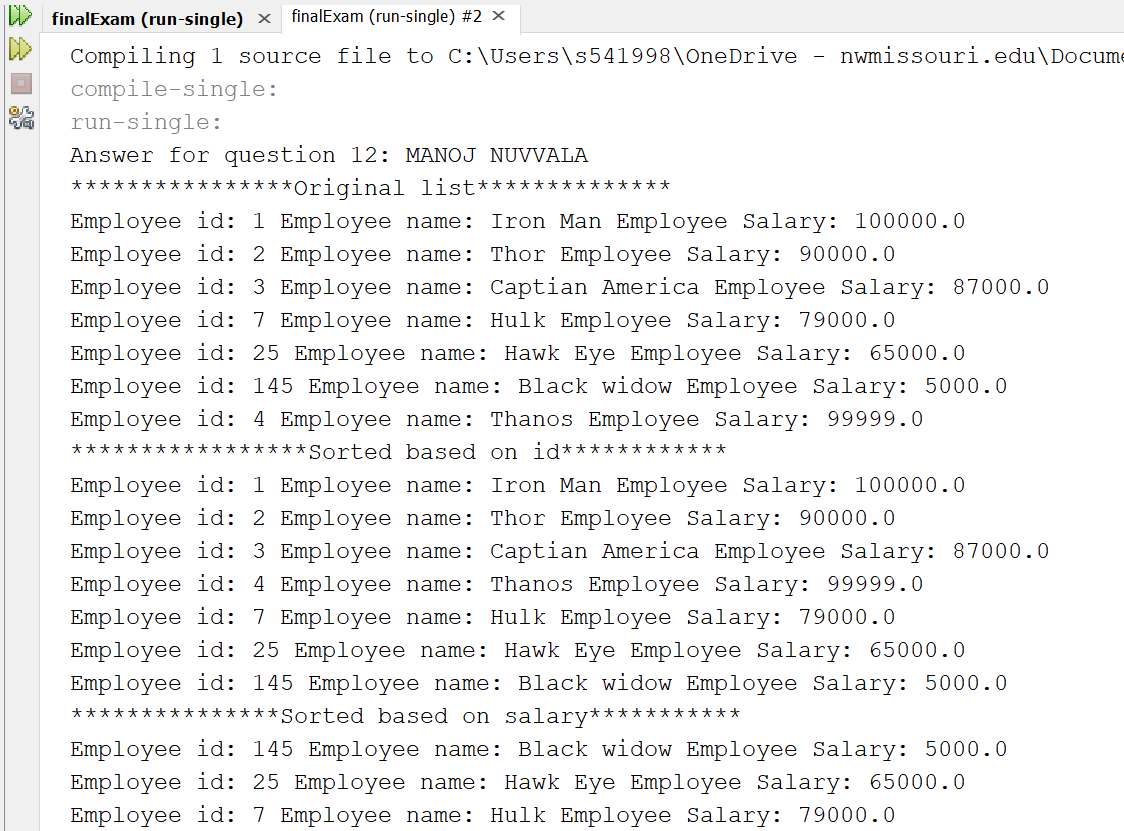
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question12;  /\*\*  \*  \* @author Manoj Nuvvala  \*/  public class Employee implements Comparable<Employee> {  private int empId;  private String empName;  private double empSalary;  public Employee(int empId, String empName, double empSalary) {  this.empId = empId;  this.empName = empName;  this.empSalary = empSalary;  }  public int getEmpId() {  return empId;  }  public String getEmpName() {  return empName;  }  public double getEmpSalary() {  return empSalary;  }  @Override  public String toString() {  return "Employee id: " + empId + " Employee name: " + empName  + " Employee Salary: " + empSalary;  }  @Override  public int compareTo(Employee obj) {  return Integer.compare(this.empId, obj.getEmpId());  }  } |

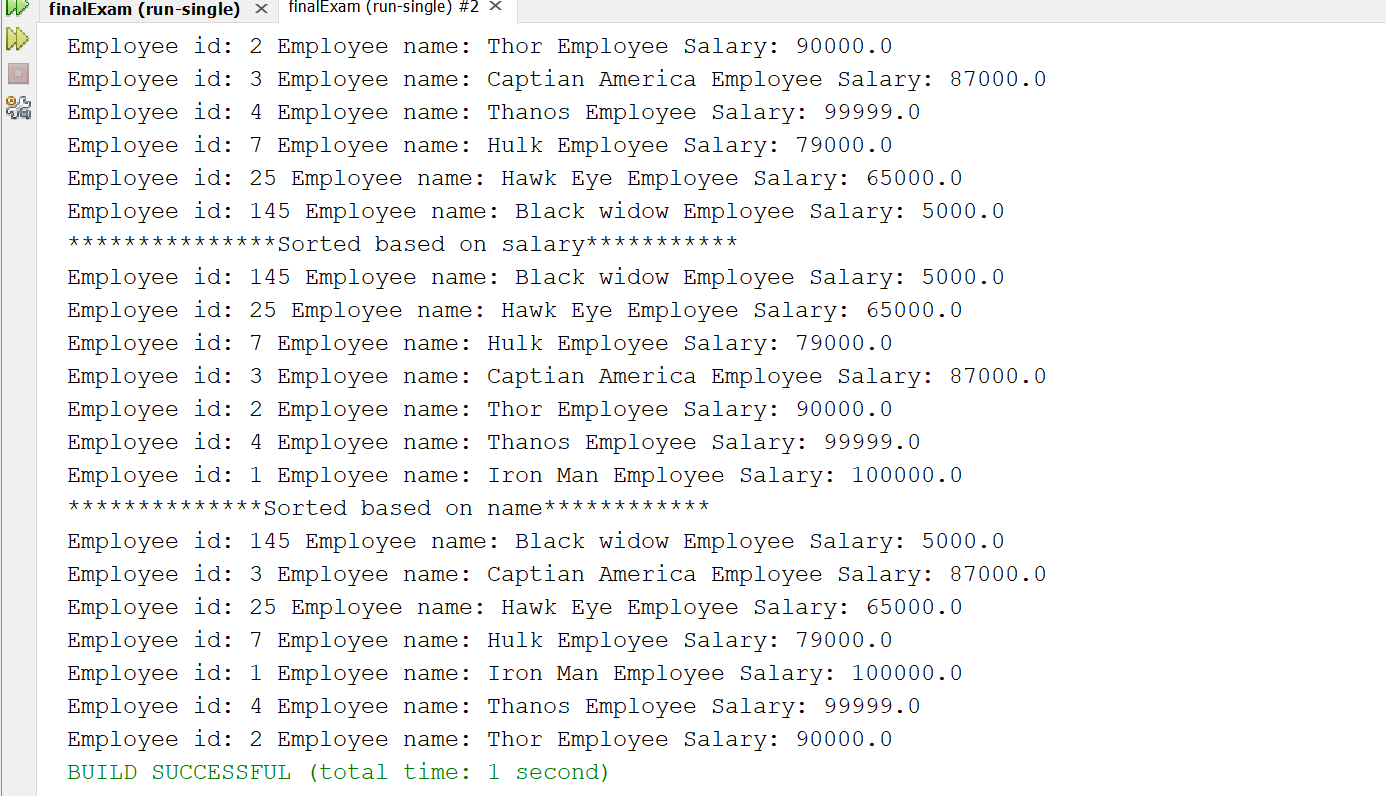
1. **EmployeeDriver Class**

* 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.

|  |
| --- |
| /\*  \* 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 Manoj Nuvvala  \*/  public class EmployeeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ArrayList<Employee> enmployees = new ArrayList<>();  System.out.println("Answer for question 12: MANOJ NUVVALA");  enmployees.add(new Employee(01, "Iron Man", 100000));  enmployees.add(new Employee(02, "Thor", 90000));  enmployees.add(new Employee(03, "Captian America", 87000));  enmployees.add(new Employee(07, "Hulk", 79000));  enmployees.add(new Employee(25, "Hawk Eye", 65000));  enmployees.add(new Employee(145, "Black widow", 5000));  enmployees.add(new Employee(04, "Thanos", 99999));  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Original list\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  for (Employee employee : enmployees) {  System.out.println(employee);  }  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sorted based on id\*\*\*\*\*\*\*\*\*\*\*\*");  Collections.sort(enmployees);  for (Employee employee : enmployees) {  System.out.println(employee);  }  Collections.sort(enmployees, new Comparator<Employee>() {  @Override  public int compare(Employee arg0, Employee arg1) {  return Double.compare(arg0.getEmpSalary(), arg1.getEmpSalary());  }  });  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sorted based on salary\*\*\*\*\*\*\*\*\*\*\*");  for (Employee employee : enmployees) {  System.out.println(employee);  }  Collections.sort(enmployees, new Comparator<Employee>() {  @Override  public int compare(Employee arg0, Employee arg1) {  return arg0.getEmpName().compareTo(arg1.getEmpName());  }  });  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sorted based on name\*\*\*\*\*\*\*\*\*\*\*\*");  for (Employee employee : enmployees) {  System.out.println(employee);  }  }  } |

**OUTPUT :**





**GITHUB LINK:** https://github.com/manojnuvvala/NUVVALA\_Spring-2021-Final-Exam