**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:**

It is a restructured class and cannot be used to create object directly. It can be used in abstract class only and can access when it is extended classed as inheritance. Abstract class abstract methods have only method names the body of the method is not implemented there. The body of abstract method is developed in the subclass.

**Interface:**

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

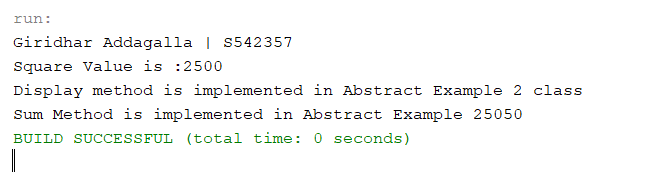
**Similarities and differences between abstract classes and interfaces**

The both are not instantiated, all abstract methods overridden by the subclass. They contain static and final variables. Dynamic polymorphism is possible using abstract class and interfaces. Because it does not have a particular structure. Using interface anything can be passed to the method with a minimum of exception. It represents minimal contract between the implementing class and caller. Interface are more flexible than base class. We can extend only one abstract class and when interface we can implements multiple interfaces. One more strong reason of choosing interface over abstraction is we can achieve full abstraction in the class methods.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| It is defined by key word abstract. | It is defined by the keyword interface. |
| It has abstract and concrete methods in it. | It has only abstract methods. |
| It has both final and non-final, static and non-static methods in it. | It has only static final variables in it. |
| It is extended using the keyword extend | It is implemented using implements keyword. |
| Abstract class can provide the implementation of the interface. | Interface cannot provide the implementation of an abstract class. |
| Abstract classes can have normal non-abstract methods. | Interfaces cannot have non-abstract methods. |

**Example:**

|  |
| --- |
| package Question01.Abstract;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class AbstractExample1 {  private int number;  public AbstractExample1(int number) {  this.number = number;  }  public int getVal() {  return number;  }  public String valSquare() {  return "Square Value is :" + (getVal() \* getVal());  }  public abstract void display();  public abstract void sum();  }  **Abstract Example 2:**  package Question01.Abstract;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class AbstractExample2 extends AbstractExample1 {  public AbstractExample2(int number) {  super(number);  }  @Override  public void display() {  System.out.println("Display method is implemented in Abstract Example 2 class");  }  @Override  public void sum() {  System.out.println("Sum Method is implemented in Abstract Example 2" + super.getVal() + super.getVal());  }  }  **Abstract Driver:**  package Question01.Abstract;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class AbstractDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  AbstractExample2 abstractObj = new AbstractExample2(50);  System.out.println(abstractObj.valSquare());  abstractObj.display();  abstractObj.sum();  }  } |



**Define a class abstract without abstract method:**

**Driver.java**

|  |
| --- |
| package Question01.WithAbstMethd;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  AbstractEx2 ex = new AbstractEx2();  ex.getValue(14);  AbstractEx2 ex1 = new AbstarctEx2b();  ex1.print();  ex1.getValue(1);  }  } |

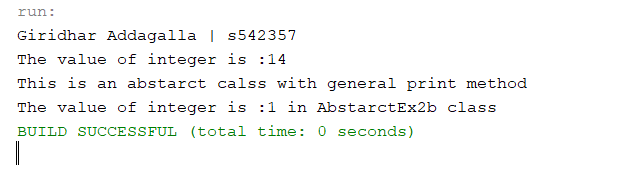
AbstractEx2b.java

|  |
| --- |
| /\*  \* 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 Question01.WithAbstMethd;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class AbstarctEx2b extends AbstractEx2 {  @Override  public void getValue(int integer) {  System.out.println("The value of integer is :" + integer + " in AbstarctEx2b class");  }  } |

AbstractEx2.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 Question01.WithAbstMethd;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class AbstractEx2 {  public void print() {  System.out.println("This is an abstarct calss with general print method");  }  public void getValue(int integer) {  System.out.println("The value of integer is :" + integer);  }  } |

Output:



**Explanation:**

In the above example the extends keyword is used to extend the class but it is not the abstract class or abstract method. It will override the getvalue method that helps to rewrite the method definition in the different way.

All Abstract Methods:

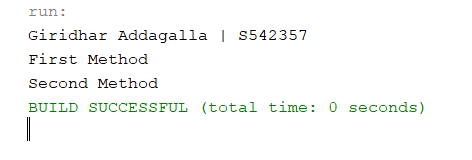
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question01.Abstract.AllAbstMethods;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class AbstractClas {  public abstract void firstMethod();  public abstract void secondMethod();  } |

ExtendsAllAbstarctMethods.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 Question01.Abstract.AllAbstMethods;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ExtendsAllAbst extends AbstractClas {  @Override  public void firstMethod() {  System.out.println("First Method");  }  @Override  public void secondMethod() {  System.out.println("Second Method");  }  } |

Driver.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 Question01.Abstract.AllAbstMethods;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  ExtendsAllAbst abs = new ExtendsAllAbst();  abs.firstMethod();  abs.secondMethod();  }  } |



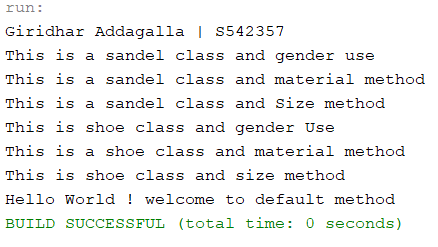
In the above example the abstract class is created and then all are abstract methods are created but not defined. The abstract methods are developed in the next class with extend class.

**Interface:**

**Footware Class:**

|  |
| --- |
| package Question01;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public interface Footware {  default void print() {  System.out.println("Hello World ! welcome to default method");  }  public void size();  public void genderUse();  public void material();  }  **Shoe Class:**  package Question01;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Shoe implements Footware {  @Override  public void size() {  System.out.println("This is shoe class and size method");  }  @Override  public void genderUse() {  System.out.println("This is shoe class and gender Use");  }  @Override  public void material() {  System.out.println("This is a shoe class and material method");  }  }  **Sandels Class:**  package Question01;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Sandels implements Footware {  @Override  public void size() {  System.out.println("This is a sandel class and Size method");  }  @Override  public void genderUse() {  System.out.println("This is a sandel class and gender use");  }  @Override  public void material() {  System.out.println("This is a sandel class and material method");  }  }  **Interface Driver:**  package Question01;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class InterfaceDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  Sandels s = new Sandels();  s.genderUse();  s.material();  s.size();  Shoe so = new Shoe();  so.genderUse();  so.material();  so.size();  so.print(); }  } |

Output:



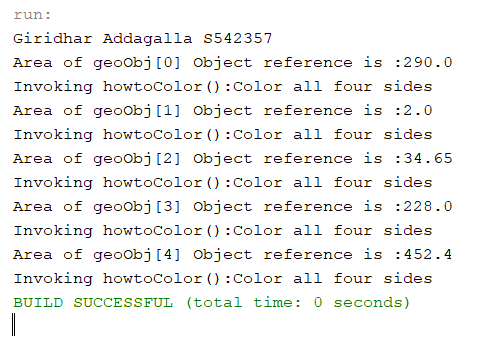
1. (10-Points) Design an interface named Colorable with a void method named howToColor(). Every class of a colorable object must implement the Colorable interface. Design a class named Square that extends GeometricObject and implements Colorable Implement howToColor to display the message Color all four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

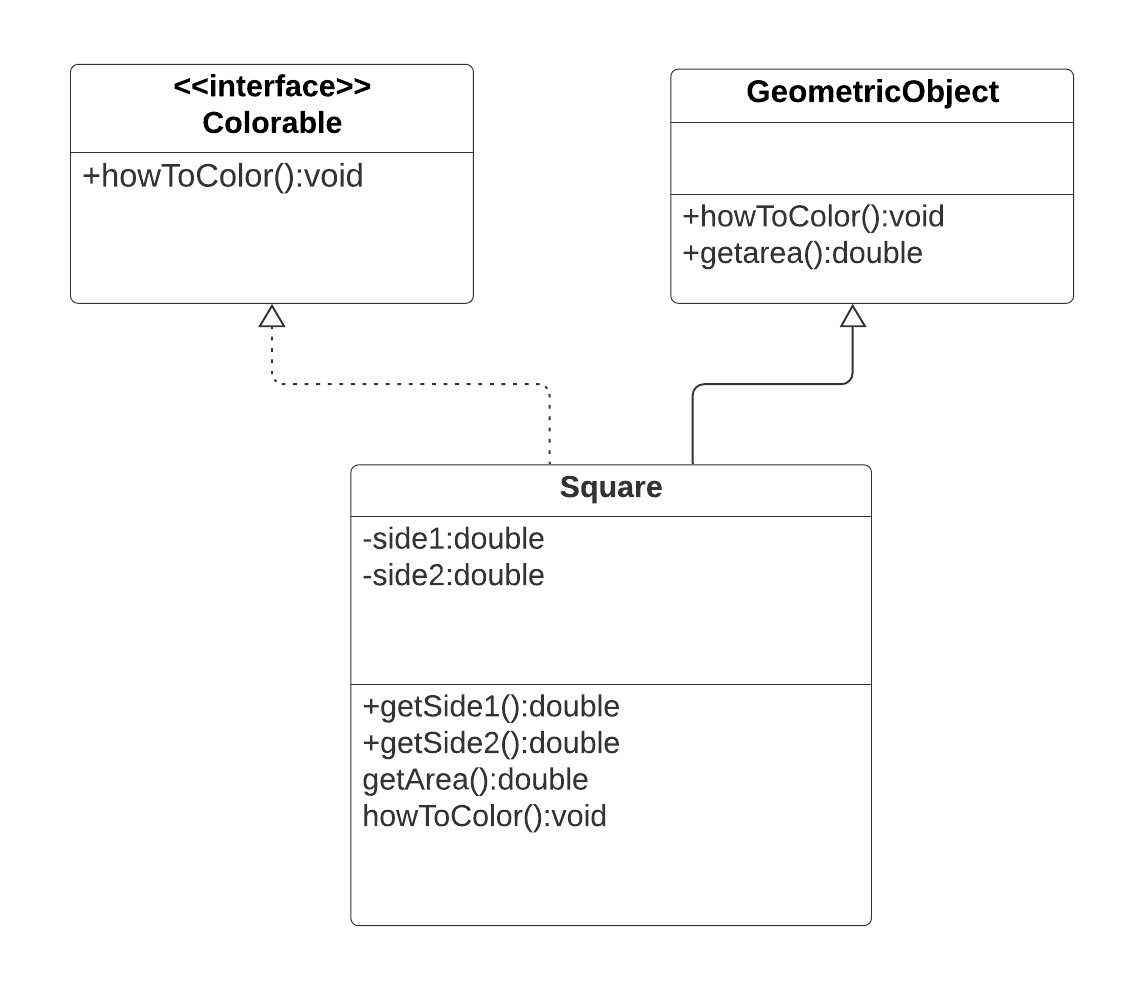
Driver Class

|  |
| --- |
| package Question02;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println(“Giridhar Addagalla S542357”);  GeometricObject[] geoObj = new GeometricObject[5];  geoObj[0] = new Square(10, 29);  geoObj[1] = new Square(1, 2);  geoObj[2] = new Square(3.3, 10.5);  geoObj[3] = new Square(12, 19);  geoObj[4] = new Square(13, 34.8);  for (int i = 0; i < geoObj.length; i++) {  System.out.println("Area of geoObj[" + i + "]" + " Object reference is :" + geoObj[i].getArea());  System.out.print("Invoking howtoColor():");  geoObj[i].howToColor();  }  }  }  GeomentricObject.java  package Question02;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public abstract class GeometricObject {  public abstract double getArea();  public void howToColor() {  System.out.println("howToColor");  }  }  Square.java  package Question02;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Square extends GeometricObject implements Colorable {  private double side1, side2;  public Square(double side1, double side2) {  this.side1 = side1;  this.side2 = side2;  }  public double getSide1() {  return side1;  }  public double getSide2() {  return side2;  }  @Override  public double getArea() {  return getSide1() \* getSide2();  }  @Override  public void howToColor() {  System.out.println("Color all four sides");  }  }  Colorable.java  package Question02;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public interface Colorable {  public void howToColor();  } |

**Output**:



UML



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

**Casting:**

Assigning a value of one primitive data type to another data type is called type casting.

Converting small type to large type size. Will be its simple definition and example.

They are two types of casting.

1. Widening casting (Up Casting)
2. Narrowing casting (Down Casting)

Up Casting:

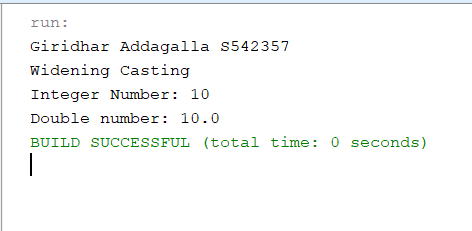
Upcasting is the type of casting converting the lower data type to higher data type. It is also called as the down casting down or implicit conversion it will done automatically.

**Example 01 (Widening Casting)**

|  |
| --- |
| package Question03.Example2;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Casting\_Exp02 {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla S542357");  System.out.println("Widening Casting");  int integer\_number = 10;  System.out.println("Integer Number: " + integer\_number);  System.out.println("Double number: " + (double) integer\_number);  }  } |

**Explination:**

In the above exam I ma converting the integer value to the double and this can be done by parsing the integer value in brackets with double before (double) integer value.



**Narrow Casting:**

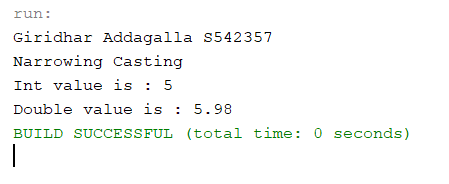
Narrow casting is the process of converting higher value to the lower one. It is also known as the casting up or explicit conversion. It performs the task manually. The compiler may give the the error while executing the code.

**Narrow Casting:**

|  |
| --- |
| package Question03;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Casting\_Ex01 {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla S542357");  System.out.println("Narrowing Casting");  double doubleValue = 5.98;  int conerted\_Int = (int) doubleValue;  System.out.println("Int value is : " + conerted\_Int);  System.out.println("Double value is : " + doubleValue);  }  } |

**Explanation:**

In the above example the double value is converted into the integer value, It means the decimal point are removed and displayed as the integer. This is one of the cases. The casing can be done in many ways but here I choose to convert double to integer.



**Example 3:**

The type casting is applicable to the objects too, I would like to consider the following example when a super class object is converted in to the sub class object the object t is converted to the Laptop subclass object but as per the casting we need to convert the l=T;

Driver.java

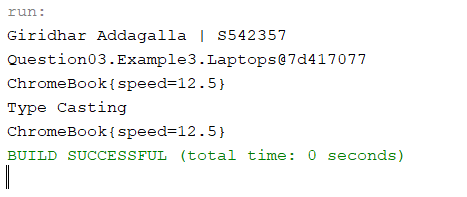
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question03.Example3;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  Laptops l = new Laptops("ch1200");  System.out.println(l);  ChromeBook c = new ChromeBook(12.5, "ch12000");  System.out.println(c);  l = c;  //typecasting  ChromeBook c1 = (ChromeBook) l;  System.out.println("Type Casting\n" + c1);  }  } |

Laptops.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 Question03.Example3;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Laptops {  private String nameOfLaptop;  public Laptops(String laptopName) {  this.nameOfLaptop = nameOfLaptop;  }  public String getNameOfLaptop() {  return nameOfLaptop;  }  public void setNameOfLaptop(String nameOfLaptop) {  this.nameOfLaptop = nameOfLaptop;  }  } |

ChromeBook.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 Question03.Example3;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ChromeBook extends Laptops {  private double speed;  public ChromeBook(String laptopName) {  super(laptopName);  }  @Override  public String toString() {  return "ChromeBook{" + "speed=" + speed + '}';  }  public ChromeBook(double speed, String laptopName) {  super(laptopName);  this.speed = speed;  }  public double getSpeed() {  return speed;  }  public void setSpeed(double speed) {  this.speed = speed;  }  } |



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:**

Legal, Here fruit instance is Fruit reference type. In this program we are storing the GoldenDelicous class instance in fruit reference variable.GoldenDelicious is subclass of Fruit

1. Is fruit instanceof Orange?

**Answer**:

Illegal, here fruit is storing the GoldenDelicous class instance in fruit reference variable. There is no is -a relationship for classes GoldenDelicious and Orange. So , fruit is not a instance of Orange.

1. Is fruit instanceof Apple?

**Answer:**

Legal ,Here fruit instance is Fruit reference type. In this program we are storing the GoldenDelicous class instance in fruit reference variable.GoldenDelicious is subclass of Apple and Apple is sub class of Fruit.

1. Is fruit instanceof GoldenDelicious?

**Answer:**

Legal ,here fruit instance is Fruit reference type. In the fruit reference we are storing the GoldenDelicious instance. So , the fruit is of type GoldenDelicious.

1. Is fruit instanceof McIntosh?

**Answer:**

Illegal , Fruit is of type GoldenDelicious. GoldenDelicious and McIntosh are the sub classes of Apple. There is no is a relation between GoldenDelicious class and McIntosh class.so , fruit is not an instance of type McIntosh.

1. Is orange instanceof Orange?

**Answer:**

Legal, here we are instantiating the orange reference variable with Orange class instance.

g. Is orange instanceof Fruit?

**Answer:**

Legal, here orange is a reference variable of type Orange.And Fruit is the super class of Orange.So ,orange can be an instance of type of Fruit.

h. Is orange instanceof Apple?

**Answer**:

Illegal, We will get a compilation error while checking the instance of orange with type Apple like incompatible types orange cannot be converted to Apple.

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

**Answer:**

We can invoke the makeAppleCider() by using the below statement

((GoldenDelicious)fruit).makeAppleCider();

Here we are defining Fruit fruit=new GoldenDelicious();

First JVM checks whether the method is available in GoldenDelicious or not .In our program method is not available in GoldenDelicious class. It will check the any override methods in its super class with name makeAppleCider();.Here we are not overriding makeAppleCider() method.so , if we directly use the fruit. makeAppleCider() will get compilation error .So . we are converting the fruit reference variable of type Fruit to GoldenDelicous and invoking makeAppleCider() method. By using casting we can invoke the makeAppleCider().

We cannot invoke the makeAppleCider() by using orange reference.

There is no is-a relation between the Apple class and Orange class. Which will generate a compilation error.

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

**Answer**:

Yes ,orange can invoke the makeOrangeJuice().orange reference storing the instance of type Orange. By using fruit reference, we cannot invoke the makeOrangeJuice().because there is no is-a relation between these two classes.

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

**Answer**:

Illegal, we will get a compilation error like ,incompatible types Apple cannot be converted to Orange. Because there is no is-a relation between the Orange class and Apple class. So , the polymorphic substitution is not allowed.

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

**Answer:**

Illegal, Here McIntosh is sub class and Apple is super class. Sub class reference cannot store the super class instance variable directly .To make this statement legal , we must perform the casting i.e. converting the Apple to McIntosh type. Like : McIntosh p=(McIntosh)new Apple();

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

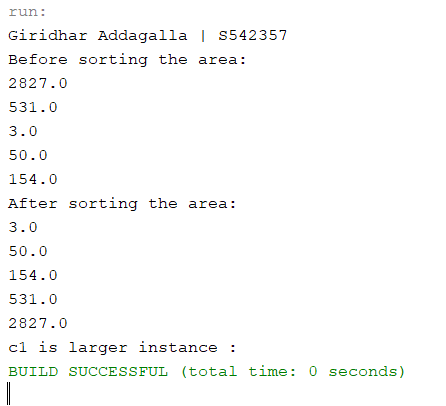
**Answer**:

Legal, here we are using polymorphic substitution . So , the statement is legal. Here Apple is super class and McIntosh is sub class.

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:**

|  |
| --- |
| Circle.java  package Question05;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Circle {  private double radius;  public Circle(double r) {  this.radius = r;  }  public double getRadius() {  return radius;  }  public double getArea() {  return Math.round(Math.PI \* getRadius() \* getRadius());  }  }  ComparableCircle.java  package Question05;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ComparableCircle extends Circle  implements Comparable {  public ComparableCircle(double r) {  super(r);  }  @Override  public int compareTo(Object arg0) {  Circle circle = (Circle) arg0;  return Double.compare(getArea(), circle.getArea());  }  }  ComparableCircleDriver.java  package Question05;  import java.util.ArrayList;  import java.util.Collections;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ComparableCircleDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  ArrayList<Double> al = new ArrayList<>();  ComparableCircle circle1 = new ComparableCircle(30);  ComparableCircle circle2 = new ComparableCircle(13);  ComparableCircle circle3 = new ComparableCircle(1);  ComparableCircle circle4 = new ComparableCircle(4);  ComparableCircle circle5 = new ComparableCircle(7);  al.add(circle1.getArea());  al.add(circle2.getArea());  al.add(circle3.getArea());  al.add(circle4.getArea());  al.add(circle5.getArea());  System.out.println("Before sorting the area:");  al.forEach((al1) -> {  System.out.println(al1);  });  Collections.sort(al);  System.out.println("After sorting the area:");  al.forEach((al1) -> {  System.out.println(al1);  });  if (circle1.compareTo(circle2) < 0) {  System.out.println("c2 is the larger instance");  } else if (circle1.compareTo(circle2) > 0) {  System.out.println("c1 is larger instance :");  } else {  System.out.println("Both are equal");  }  }  } |



**UML Diagram**

**Diagram

Description automatically generated**

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

**Answer:**

**Exception:** An exception is problem that arises during the execution of the problem. The exception disturbs the program while executing

They are two types of exceptions:

1. Checked exception.
2. Un checked exception.

**Checked Exception:**

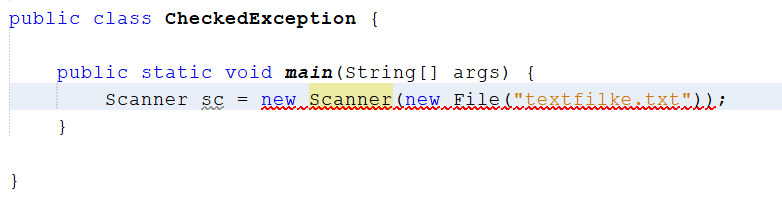
The checked exception is the checked by the compiler in the compile time it is also called as the compile time exception. All subclasses of Exception except Run time Exception and its subclass are checked exceptions. By using the throws keyword, we can throw the checked exception at last try-catch is used to handle the checked exception.

Few of the exception are

1. I O Exception
2. File Not Found Exception
3. E O F Exception

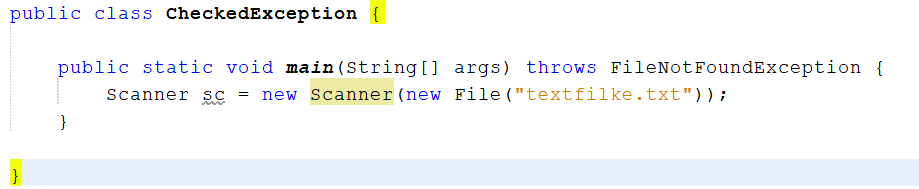
In the below code FileNotFoundException , we can use its super classes also like IOException or Exception. For checked exception we have to specify the correct exception

|  |
| --- |
| public static void main(String[] a)throws FileNotFoundException:  //body of the method;  } |



In the above example we are trying to access the file then it raises exception saying unreported exception. FileNotFoundException might be caught or declared to be thrown.

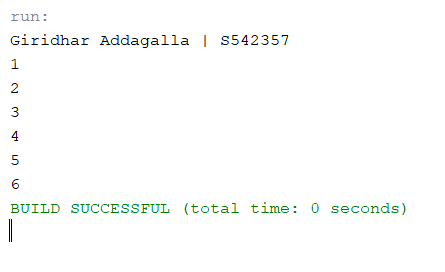
**The above can be fixed using the throws FileNotFoundException**



CheckedException.java

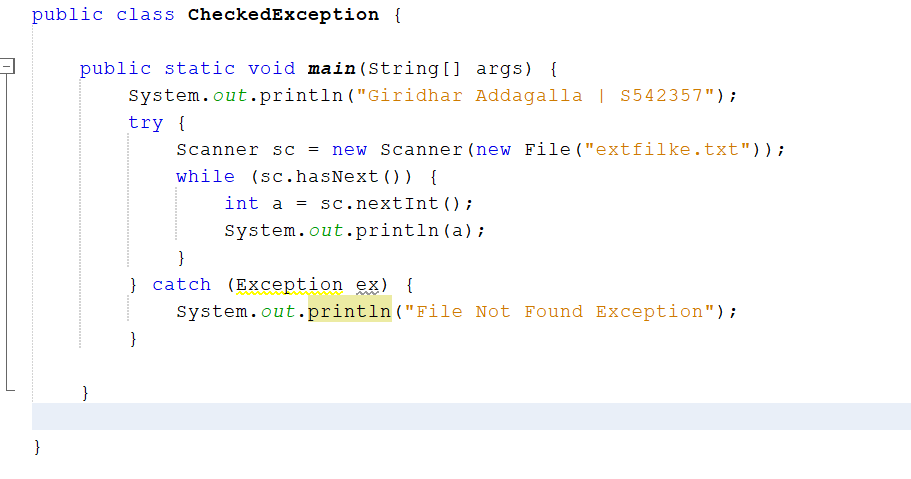
|  |
| --- |
| package Question06;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class CheckedException {  public static void main(String[] args) throws FileNotFoundException {  Scanner sc = new Scanner(new File("textfilke.txt"));  while (sc.hasNext()) {  int a = sc.nextInt();  System.out.println(a);  }  }  } |

**Output:**

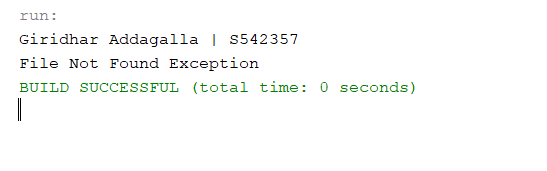


**2. Checked Exception: (Try-Catch Block)**

|  |
| --- |
| package Question06.CheckedUsingTry;  import java.io.File;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class CheckedException {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  try {  Scanner sc = new Scanner(new File("extfilke.txt"));  while (sc.hasNext()) {  int a = sc.nextInt();  System.out.println(a);  }  } catch (Exception ex) {  System.out.println("File Not Found Exception");  }  }  } |



In the above code I have removed the file name and the file name is not found in the location then the exception is raised caught by the catch block.



**Un Checked Exception:**

Un checked exception is the exception raised in the runtime. They are not advertised while writing the code. Runtime Exception and its subclasses are unchecked exception. Some of the unchecked exceptions are NullPointerException ,IndexOutOfBoundsException, Arithmetic Exception etc. This exception mostly occurred due to bad coding or incorrect logic. The developer can fix the exception and handle them correctly.

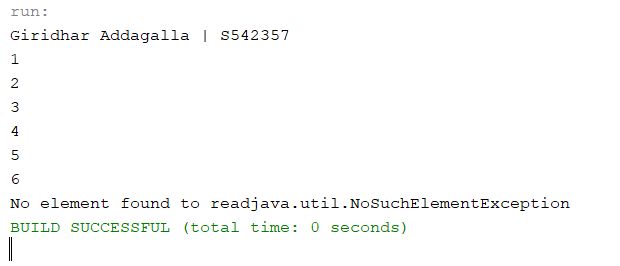


The above exception is handled using the catch exception in the below code.

**UncheckedException.java**

|  |
| --- |
| package Question06.UnCheckedException;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class UnCheckedException {  public static void main(String[] args) {  Scanner sc;  try {  sc = new Scanner(new File("textfilke.txt"));  int a = sc.nextInt();  while (true) {  System.out.println(a);  a = sc.nextInt();  }  } catch (FileNotFoundException ex) {  System.out.println("File not found Exception" + ex.getMessage());  } catch (java.util.NoSuchElementException ex) {  System.out.println("No element found to read" + ex);  }  }  } |

Output:

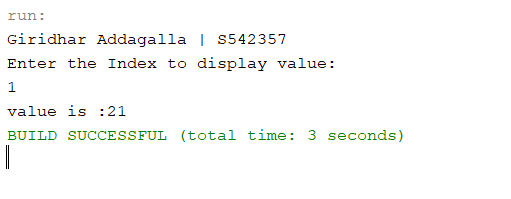


1. (10-Points) Write a program that meets the following requirements:

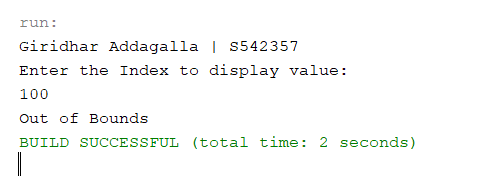
* Creates an array with 100 randomly chosen integers.
* Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds.

RandomNumberArray.java

|  |
| --- |
| package Question07;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class RandomNumberArray {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  int randomArr[] = new int[100];  for (int i = 0; i < randomArr.length; i++) {  randomArr[i] = (int) (Math.random() \* 100);  }  Scanner sc = new Scanner(System.in);  try {  System.out.println("Enter the Index to display value:");  int index = sc.nextInt();  System.out.println("value is :" + randomArr[index]);  } catch (ArrayIndexOutOfBoundsException exp) {  System.out.println("Out of Bounds");  }  }  } |



**Out of Bound:**



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 the exception is to handle the different types of the run time errors. And the program will be terminated .On that situation by using exception we can handle Errors; termination is not necessary. For example, divide by a number , our program will run for every number except 0 , divide by 0 is undefined , so the system doesn’t know the answer and it gives a run time error. To avoid this situation, we will use expectation we throws an arithmetic exception.

|  |
| --- |
| public int divide() throws ArithmeticException{  //body of the method;  } |

We can declare the multiple exception in the method header. These exceptions are separated by commas.

|  |
| --- |
| Public int divide() throws Arithmetic Exception , Exception{  //body of the method  } |

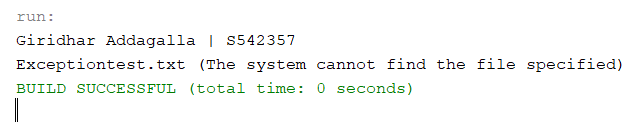
**FilentFound.java**

|  |
| --- |
| package Question08;  import java.io.\*;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class FilentFound {  public void getDetails() throws FileNotFoundException {  Scanner sc = new Scanner(new File("test.txt"));  }  } |

**DriverClass.java**

|  |
| --- |
| package Question08;  import java.io.FileNotFoundException;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class DriverClass {  public static void main(String[] args) throws FileNotFoundException {  FilentFound fc = new FilentFound();  try {  fc.getDetails();  } catch (Exception ex) {  System.out.println("Exception" + ex.getMessage());  }  }  } |

In the above example I have written the display method under try and catch block because it will help to catch the exception in the runtime, this helps to keep the program alive from breaking in between. When the program executed then the program validated the file and check the file present or not then if the file is found then it will continue the process else the program will hit the catch block and print the statement in the catch block.

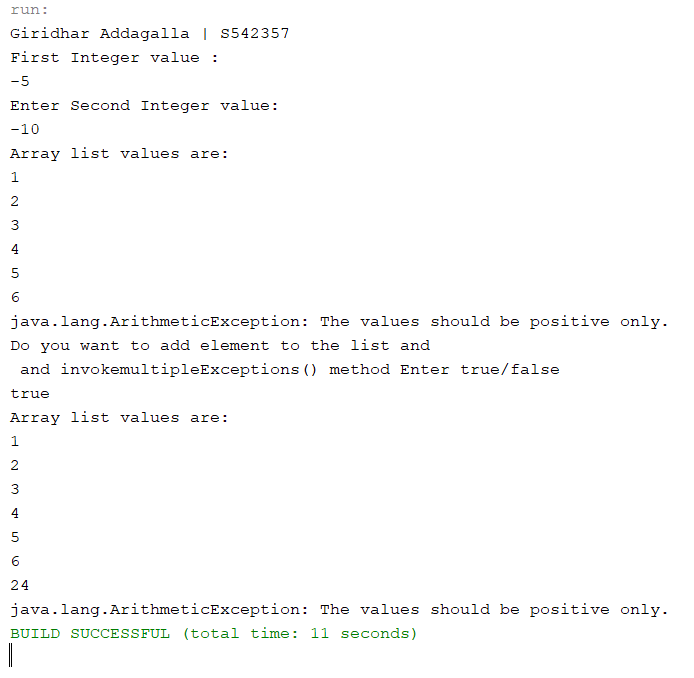


Example: 2

|  |
| --- |
| package Question08.ArithAndArrayIndex;  import java.util.ArrayList;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Arithmatic {  public void twoExceptions(ArrayList<Integer> listOne, int int2, int int3) throws ArithmeticException, IndexOutOfBoundsException {  if (listOne.size() > 9) {  throw new IndexOutOfBoundsException();  } else {  System.out.println("Array list values are:");  listOne.forEach((a1) -> {  System.out.println(a1);  });  }  if (int2 <= 0 || int3 <= 0) {  throw new ArithmeticException("The values should be positive only.");  } else {  System.out.println("Division :" + int2 / int3);  }  }  } |

Driver.java

|  |
| --- |
| package Question08.ArithAndArrayIndex;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.ArrayList;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  public static void main(String[] args) throws FileNotFoundException {  System.out.println("Giridhar Addagalla | S542357");  ArrayList<Integer> intList = new ArrayList<>();  Scanner sc = new Scanner(new File("textfilke.txt"));  Scanner s = new Scanner(System.in);  while (sc.hasNext()) {  int a = sc.nextInt();  intList.add(a);  }  System.out.println("First Integer value :");  int int2 = s.nextInt();  System.out.println("Enter Second Integer value:");  int int3 = s.nextInt();  Arithmatic ex = new Arithmatic();  try {  ex.twoExceptions(intList, int2, int3);  } catch (ArithmeticException | IndexOutOfBoundsException ex1) {  System.out.println(ex1);  }  try {  System.out.println("Do you want to add element to the list and\n and invoke"  + "multipleExceptions() method Enter true/false");  boolean d = s.nextBoolean();  if (d == true) {  intList.add(24);  ex.twoExceptions(intList, int2, int3);  }  System.out.println("Done !!");  } catch (ArithmeticException | IndexOutOfBoundsException ex2) {  System.out.println(ex2);  }  }  } |



NotValidException.java

|  |
| --- |
| package Question08.ArithAndArrayIndex;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class NotValidException extends Exception {  public NotValidException() {  }  public NotValidException(String msg) {  super(msg);  }  } |

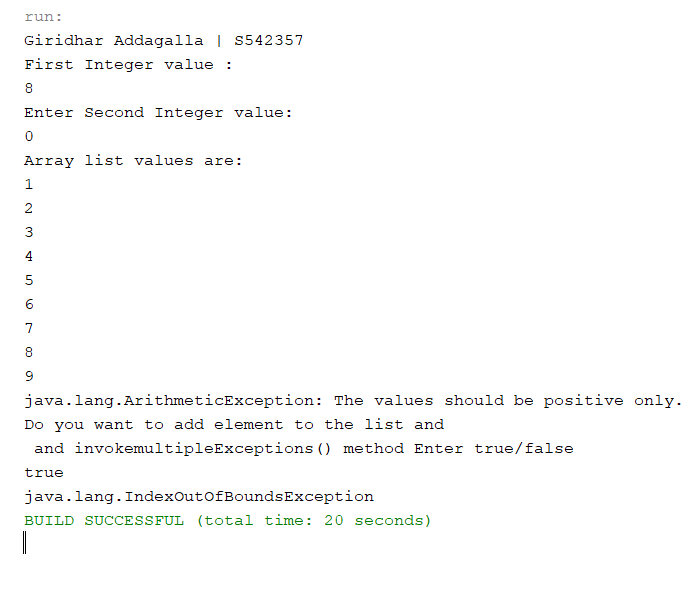
In the above example we are throwing the multiple exception, we have created the method name twoExceptions() this may throw ArithmeticException, IndexOutOfBoundsException, NotValidException. In this next we are checking the list size which is greater than 9 or not.I have updated the Arithmetic exception class which will handle the 0 by division and make sure the code catches the 0 value.

**Arithmatic.java**

|  |
| --- |
| package Question08.ArithAndArrayIndex;  import java.util.ArrayList;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Arithmatic {  public void twoExceptions(ArrayList<Integer> listOne, int int2, int int3) throws ArithmeticException, IndexOutOfBoundsException, NotValidException {  if (listOne.size() > 9) {  throw new IndexOutOfBoundsException();  } else {  System.out.println("Array list values are:");  listOne.forEach((a1) -> {  System.out.println(a1);  });  }  if (int2 <= 0 || int3 <= 0) {  throw new ArithmeticException("The values should be positive only.");  } else if (int2 > int3) {  throw new NotValidException("int3 value must be less than int2 ");  } else {  System.out.println("Division :" + int2 / int3);  }  }  } |

**Driver.java**

|  |
| --- |
| package Question08.ArithAndArrayIndex;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.ArrayList;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  public static void main(String[] args) throws FileNotFoundException, NotValidException {  System.out.println("Giridhar Addagalla | S542357");  ArrayList<Integer> intList = new ArrayList<>();  Scanner sc = new Scanner(new File("textfilke.txt"));  Scanner s = new Scanner(System.in);  while (sc.hasNext()) {  int a = sc.nextInt();  intList.add(a);  }  System.out.println("First Integer value :");  int int2 = s.nextInt();  System.out.println("Enter Second Integer value:");  int int3 = s.nextInt();  Arithmatic ex = new Arithmatic();  try {  ex.twoExceptions(intList, int2, int3);  } catch (ArithmeticException | IndexOutOfBoundsException | NotValidException ex1) {  System.out.println(ex1);  }  try {  System.out.println("Do you want to add element to the list and\n and invoke"  + "multipleExceptions() method Enter true/false");  boolean d = s.nextBoolean();  if (d == true) {  intList.add(24);  ex.twoExceptions(intList, int2, int3);  }  System.out.println("Done !!");  } catch (ArithmeticException | IndexOutOfBoundsException | NotValidException ex2) {  System.out.println(ex2);  }  }  } |

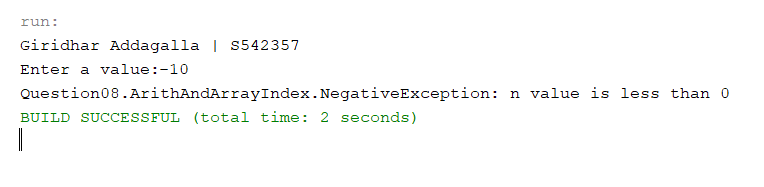
+

**NegativeException.java**

|  |
| --- |
| package Question08.ArithAndArrayIndex;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  class NegativeException extends Exception {  public NegativeException(String msg) {  super(msg);  }  public NegativeException() {  }  } |

**NegativeExceptionDriver.java**

|  |
| --- |
| package Question08.ArithAndArrayIndex;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class NegativeExceptionDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  Scanner sc = new Scanner(System.in);  System.out.print("Enter a value:");  int integer = sc.nextInt();  if (integer < 0) {  try {  throw new NegativeException("n value is less than 0");  } catch (NegativeException ex) {  System.out.println(ex);  }  } else {  System.out.println("Square is :" + (integer \* integer));  }  }  } |



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.

**Throw Keyword:**

It is used to throw an exception for a block of code or inside a method. By using this keyword, we can throw our own exception. By using the throw, we can throw on exception at a time.

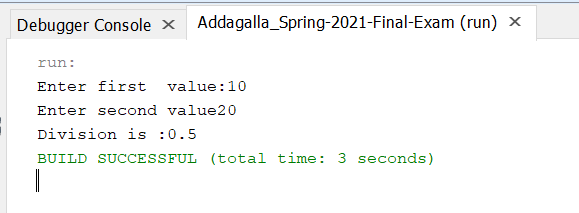
By using the throws, we can throw multiple exception using comma separated multiple exceptions. The keyword used for announcing an exception might occurs while running the code. The throws keyword followed by the class name.

No, we cannot throw multiple exception with one throw statements. We can only throw single exception at a time.

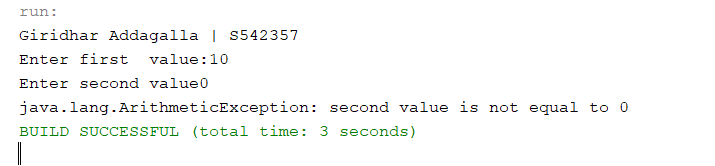
Throwsdriver.java

|  |
| --- |
| package Question09;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class ThrowsDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.print("Enter first value:");  double x = sc.nextInt();  System.out.print("Enter second value");  double y = sc.nextInt();  if (y == 0) {  try {  throw new ArithmeticException("second value is not equal to 0");  } catch (ArithmeticException ex) {  System.out.println(ex);  }  } else {  System.out.println("Division is :" + (x / y));  }  }  } |

Output:



Output:



NotValidException.java

|  |
| --- |
| package Question08.ArithAndArrayIndex;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class NotValidException extends Exception {  public NotValidException() {  }  public NotValidException(String msg) {  super(msg);  }  } |

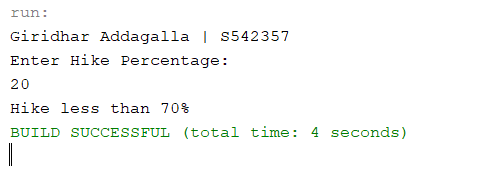
HikePercentageDriver.java

|  |
| --- |
| package Question09.SalaryHike;  import Question08.ArithAndArrayIndex.NotValidException;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class HikePercentageDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter Hike Percentage:");  int n = sc.nextInt();  try {  CheckHike sg = new CheckHike(n);  if (sg.getHikePercentage() < 0 || sg.getHikePercentage() > 100) {  throw new NotValidException("Invalid Hike Percentage");  }  if (sg.getHikePercentage() >= 90 && sg.getHikePercentage() <= 100) {  System.out.println("Highest Hike");  } else if (sg.getHikePercentage() < 90 && sg.getHikePercentage() >= 80) {  System.out.println("Mid level Hike");  } else if (sg.getHikePercentage() < 80 && sg.getHikePercentage() >= 70) {  System.out.println("Hike Level: 70-80");  } else {  System.out.println("Hike less than 70%");  }  } catch (NotValidException ex) {  System.out.println(ex);  }  }  } |

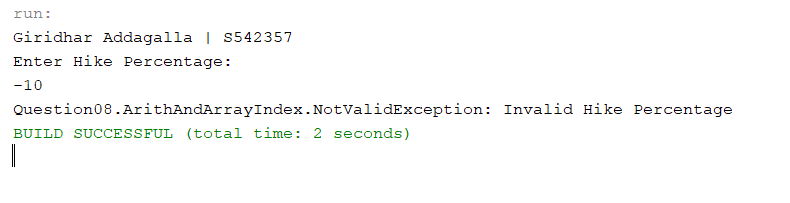
CheckHike.java

|  |
| --- |
| package Question09.SalaryHike;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class CheckHike {  private double hikePercentage;  public CheckHike(double hikePercentage) {  this.hikePercentage = hikePercentage;  }  public double getHikePercentage() {  return hikePercentage;  }  } |

Output



**Exception**



**Explanation:**

In the above example we are verifying the integer value, The integer value is used as the hike percentage, The hike percentage should be greater than 0, If it is negative value then it raises the exception and move to the catch block.

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:**

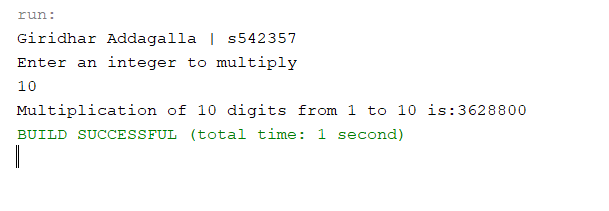
**Recursive Method:** A method invoke itself directly or indirectly is called recursion and particular method is called recursive method.

Infinite Recursion: The recursive methods involves continuously and there is no way to stop the recursive call is called the infinity recursive.

**Direct Recursive Example:**

|  |
| --- |
| package Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class DirectRec {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | s542357");  Scanner sc = new Scanner(System.in);  System.out.println("Enter an integer to multiply");  int value = sc.nextInt();  System.out.println("Multiplication of " + value + " digits from 1 to " + value + " is:" + multiply(value));  }  private static int multiply(int value) {  if (value > 1) {  return value \* multiply(value - 1);  } else {  return 1;  }  }  } |

**Output:**



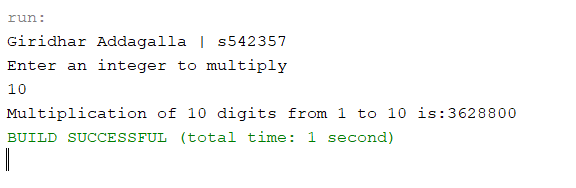
**Explanation:**

In the above example there is a method called multiplication(), this method logic is keep on recursing and producing the result based on the validation present. If the validation is not there, then it will be an infinity recursive. Until and unless the value is less than 1 the loop will keep on going.

**Example for Indirective Recursive:**

|  |
| --- |
| package Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class IndirectRec {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | s542357");  Scanner sc = new Scanner(System.in);  System.out.println("Enter an integer to multiply");  int value = sc.nextInt();  System.out.println("Multiplication of " + value + " digits from 1 to " + value + " is:" + multiply(value));  }  private static int multiply(int value) {  if (value > 1) {  return value \* multiply1(value - 1);  } else {  return 1;  }  }  private static int multiply1(int value) {  if (value > 1) {  return value \* multiply(value - 1);  } else {  return 1;  }  }  } |

**Output:**



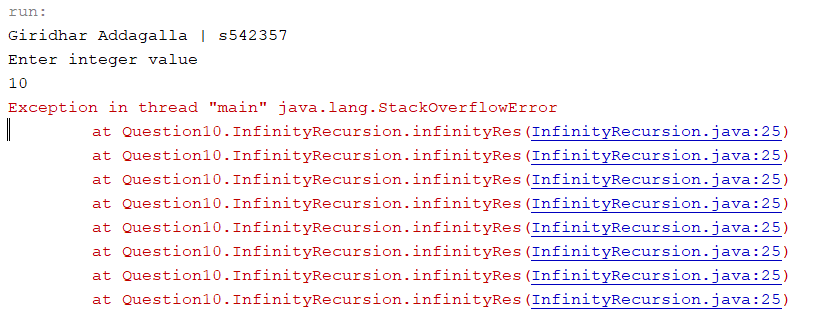
**Explanation:**

In the about the code we are using two methods multiplication(), multiplication1() have written the code in the indirect recursive way this helps us to get the logic indirectly recuring and move to the next method in order to the result. The program will stop execution once it reaches the validation point. The validation point will check weather value is less than 1 or not. If the value is less than 1 then the method will stop execution.

Infinity Recursive:

|  |
| --- |
| package Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class InfinityRecursion {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | s542357");  Scanner sc = new Scanner(System.in);  System.out.println("Enter integer value");  int value = sc.nextInt();  try {  System.out.println(infinityRes(value));  } catch (Exception ex) {  }  }  private static String infinityRes(int value) {  if (1 == 1) {  return "infinite recursion" + "\n" + infinityRes(value + 1);  } else {  return "";  }  }  } |

**Output:**



**Explanation**:

In the above example I defined an method which has a validation statement is always true, This validation statement will not verify with real term values but it compromise the code from execution. And allow the code running with out validation, So in this case the loop will not stop and continuously running and create exceptions. In the next Example I will explain the similar program with try and catch blocks.

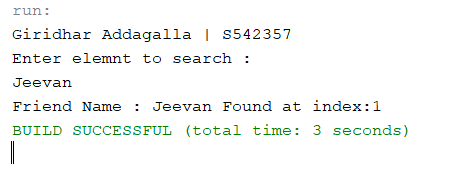
**Friends.java**

|  |
| --- |
| package Question10;  import java.util.ArrayList;  import java.util.NoSuchElementException;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Friends {  private ArrayList<String> list1;  private static int i = 0;  public Friends(ArrayList<String> list1) {  this.list1 = list1;  }  public int findMyFriend(String n) {  if (list1.size() < i + 1) {  throw new NoSuchElementException(n + " Name is not available in the list");  }  if (list1.get(i).equals(n)) {  return i;  } else {  i = i + 1;  return findMyFriend(n);  }  }  } |

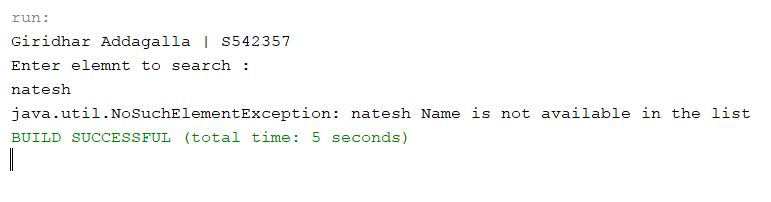
**FindmyFriend.Java**

|  |
| --- |
| package Question10;  import java.util.ArrayList;  import java.util.NoSuchElementException;  import java.util.Scanner;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class FindMyFriendDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  ArrayList<String> friendList = new ArrayList<String>();  friendList.add("Vasavi");  friendList.add("Jeevan");  friendList.add("Raghava");  friendList.add("Prasamsha");  System.out.println("Enter elemnt to search :");  try {  Scanner sc = new Scanner(System.in);  String e = sc.next();  Friends fr = new Friends(friendList);  System.out.println("Friend Name : " + e + " Found at index:" + fr.findMyFriend(e));  } catch (NoSuchElementException ex) {  System.out.println(ex);  }  }  } |

**Output**:



**Exception**:



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

Answer:

**Equals()**: This method is used from java lang object, it is used to compare two objects weather they are same or not. It compares the both values.

**Hash code**(): The Object class in Java contains a method named **hashCode** that returns a hash code value for an object. It returns the same hash value when called on two objects , they are equal according to the equals method(). If the object is different it returns different hash value.

If two objects are the same as per the equal(object) method, then if we call the hashcode() method on the each of two object. As per the documentation both the method should be overridden to get complete equality mechanism, using equals() alone is not sufficient. It means, if we override the equals(), we must override the hashcode() method.

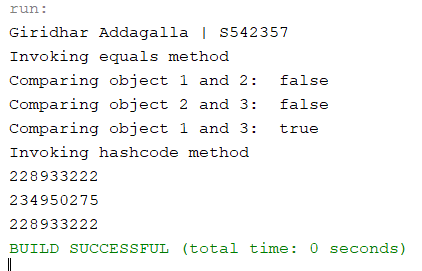
Desktop.java

|  |
| --- |
| /\*  \* 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 Giridhar Addagalla  \*/  public class Desktop {  private double price;  private String model;  public Desktop(double price, String model) {  this.price = price;  this.model = model;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  public String getModel() {  return model;  }  public void setModel(String model) {  this.model = model;  }  @Override  public int hashCode() {  int hash = 4;  hash = 14 \* hash + (int) (Double.doubleToLongBits(this.price) ^ (Double.doubleToLongBits(this.price) >>> 32));  hash = 14 \* hash + Objects.hashCode(this.model);  return hash;  }  @Override  public boolean equals(Object object) {  if (this == object) {  return true;  }  if (object == null) {  return false;  }  if (getClass() != object.getClass()) {  return false;  }  final Desktop other = (Desktop) object;  if (Double.doubleToLongBits(this.price) != Double.doubleToLongBits(other.price)) {  return false;  }  return Objects.equals(this.model, other.model);  }  } |

Driver.java

|  |
| --- |
| package Question11;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class Driver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  Desktop desktop1 = new Desktop(250, "DES3546");  Desktop desktop2 = new Desktop(350, "DES1376");  Desktop desktop3 = new Desktop(250, "DES3546");  System.out.println("Invoking equals method");  System.out.println("Comparing object 1 and 2: " + desktop1.equals(desktop2));  System.out.println("Comparing object 2 and 3: " + desktop2.equals(desktop3));  System.out.println("Comparing object 1 and 3: " + desktop3.equals(desktop1));  System.out.println("Invoking hashcode method");  System.out.println(desktop1.hashCode());  System.out.println(desktop2.hashCode());  System.out.println(desktop3.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.

**Answer:**

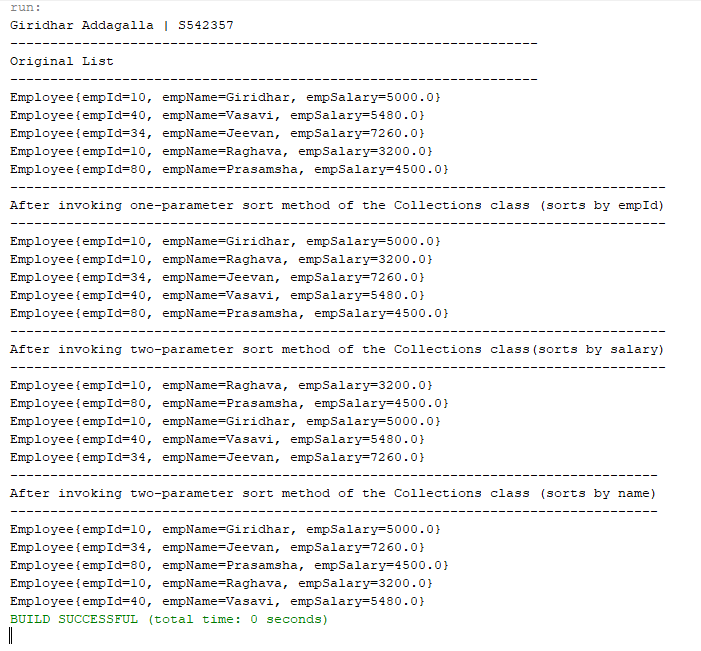
Employee.java

|  |
| --- |
| package Question11;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  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{" + "empId=" + empId + ", empName=" + empName + ", empSalary=" + empSalary + '}';  }  @Override  public int compareTo(Employee e1) {  return Integer.compare(empId, e1.getEmpId());  }  } |

EmployeeDriver.java

|  |
| --- |
| package Question11;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  import java.util.List;  /\*\*  \*  \* @author Giridhar Addagalla  \*/  public class EmployeeDriver {  public static void main(String[] args) {  System.out.println("Giridhar Addagalla | S542357");  ArrayList<Employee> employees = new ArrayList<>();  employees.add(new Employee(10, "Giridhar", 6000));  employees.add(new Employee(40, "Vasavi", 5870));  employees.add(new Employee(34, "Jeevan", 8750));  employees.add(new Employee(10, "Raghava", 4500));  employees.add(new Employee(80, "Prasamsha", 5500));  System.out.println("------------------------------------------------------------------");  System.out.println("Original List");  System.out.println("------------------------------------------------------------------");  printlist(employees);  Collections.sort(employees);  System.out.println("----------------------------------------------------------------------------------");  System.out.println("After invoking one-parameter sort method of the Collections class (sorts by empId)");  System.out.println("----------------------------------------------------------------------------------");  printlist(employees);  System.out.println("----------------------------------------------------------------------------------");  System.out.println("After invoking two-parameter sort method of the Collections class(sorts by salary)");  System.out.println("----------------------------------------------------------------------------------");  Collections.sort(employees, new Comparator<Employee>() {  @Override  public int compare(Employee e1, Employee e2) {  return Double.compare(e1.getEmpSalary(), e2.getEmpSalary());  } // end compareTo  });  printlist(employees);  Collections.sort(employees, new Comparator<Employee>() {  @Override  public int compare(Employee e1, Employee e2) {  return e1.getEmpName().compareTo(e2.getEmpName());  } // end compareTo  }  );  System.out.println("---------------------------------------------------------------------------------");  System.out.println("After invoking two-parameter sort method of the Collections class (sorts by name)");  System.out.println("---------------------------------------------------------------------------------");  printlist(employees);  }  private static void printlist(List<Employee> employees) {  employees.forEach((s) -> {  System.out.println(s);  });  }  } |

Output:



GitHub: [giridhar196/Addagalla\_Spring-2021-Final-Exam (github.com)](https://github.com/giridhar196/Addagalla_Spring-2021-Final-Exam)