**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.

1(A) Abstract Class

* A class which has the abstract keyword in its declaration is called abstract class.
* Abstract classes should have at least one abstract method.
* An abstract class permits you to make functionality that subclasses can implement or override.
* A class can extend only one abstract class.

Keyword:

An abstract class can be extended using keyword "extends".

Example:

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon)

abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, then the class itself must be declared abstract:

|  |
| --- |
| public abstract class Shape{  public abstract void draw();  } |

Interface

* 1.In interface only permits you to state functionality but not to implement it.
* 2.while a class can implement multiple interfaces
* 3.Interfaces are used to achieve abstraction.
* 4.Designed to support dynamic method resolution at run time

keyword:

The interface keyword is used to declare interface.

Example:

An interface is a completely "**abstract class**" that is used to group related methods with empty bodies.

|  |
| --- |
| public interface Drawable{  void draw();  } |

Similarities between Interface and Abstract Class:

1. 1.object cannot be created for both.
2. All the abstract methods must be overridden by subclass.
3. Both Don’t have default constructor.
4. dynamic polymorphism can be used.
5. Both can contain static and final variable .

Difference between Interface and Abstract Class:

|  |  |
| --- | --- |
| ABSTRACT CLASS | INTERFACE |
| Abstract class can have abstract and non-abstract method | Interface can only have abstract method. |
| Abstract class does not support mutiple inheritance | Interface supports multiple inheritance |
| Abstract class can have final ,non-final ,static and non-static variable. | Interface has only static and final variable. |
| Abstract class can provide the implementation of interface. | Interface can’t provide the implementation of abstract class. |
| An abstract class can extend another java class and implement multiple java interfaces. | Interface can extend another java interface only. |
| It can have class members like private ,protected ,etc. | Members are public by default. |
| Abstract class can not be instantiated but can be invoked | Interface can not be instantiated or invoked |
| Abstract class has parametrised and non-parametrised constructor but don’t have default constructor | Interface does not support the concept of constructor |

Interfaces are preferred over abstract classes:

1.Cannot be instantiated.

2.Implementation of all members of the interface occurs in the base class. It is not possible to implement only some members within the implementing class.

3.Extending interfaces with additional members breaks the version compatibility.

4.Cannot store any data. Fields can be specified only on the deriving classes. The workaround for this is to define properties, but without implementation.

5.All members are automatically virtual and cannot include any implementation.

6.Although no default implementation can appear, classes implementing interfaces can continue to derive from one another.

Abstract class example:

Question1Abstract1

Abstract\_a.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 Question1Abstract1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public abstract class Abstract\_a {  private int value;  public Abstract\_a(int value) {  this.value = value;  }  public int getValue() {  return value;  }  public String valSquare() {  return "Square Value of X is :" + (getValue() \* getValue());  }  public abstract void display();  } |

Abstract\_b.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 Question1Abstract1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract\_b extends Abstract\_a {  public Abstract\_b(int val) {  super(val);  }  @Override  public void display() {  System.out.println("Implementing the Display method in Abstract\_b class");  }  } |

Abstract1Driver.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 Question1Abstract1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract1Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Abstract\_b a = new Abstract\_b(10);  System.out.println(a.valSquare());  a.display();  }  } |

In the above model , we are characterizing a theoretical class with one conceptual technique i.e. in Abstract\_a class we are announcing case variable and instating the variable with a defined constructor. Furthermore, characterizing a solid strategy for example strategy with body which will print the square of the case variable , and we are pronouncing a theoretical technique names as display. In Abstract\_b class , we are instated a defined constructor to call super class constructor and we are carrying out the theoretical strategy show with a print proclamation .In driver class we are making object for Abstract\_b class and summoning the both valSquare() and display().We are not making the item for AbstractEx1 class since we proclaimed class as abstract. If we attempt to make an article for Abstract\_a will get assemblage blunder.

Output:

|  |
| --- |
|  |

Abstract without any abstract methods:

Question1Abstract2

Abstract\_a.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 Question1Abstract2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public abstract class Abstract\_a {  public void print() {  System.out.println("print method in Abstract\_a class");  }  public void getValue(int y) {  System.out.println("The value of x is :" + y);  }  } |

Abstract\_b.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 Question1Abstract2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract\_b extends Abstract\_a {  @Override  public void getValue(int y) {  System.out.println("The value of X is :" + y + " in Abstract\_b class");  }  } |

Abstract2Driver.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 Question1Abstract2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract2Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Abstract\_b ab = new Abstract\_b();  ab.getValue(8);  Abstract\_a ab1 = new Abstract\_b();  ab1.print();  ab1.getValue(10);  }  } |

In the model , we are characterizing a class as theoretical with no theoretical techniques .In the Abstract\_a class we are characterizing two solid strategies and characterizing the class as conceptual. In Abstract\_b class we are superseding the getValue(x).In driver class we are making the item for Abstract\_b class and conjuring the getValue(x) strategy .and In the following line we are making the article for Abstract\_a class by utilizing polymorphic replacement and summoning the strategies print() and getValue(x).

Output:

|  |
| --- |
|  |

Abstract class with all abstract methods :

Question1Abstract3

Abstract\_a.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 Question1Abstract3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public abstract class Abstract\_a {  public abstract int sum();  public abstract int multiply();  } |

Abstract\_b.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 Question1Abstract3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract\_b extends Abstract\_a {  private int x;  public Abstract\_b(int z) {  this.x = x;  }  @Override  public int sum() {  return x + x;  }  @Override  public int multiply() {  return x \* x;  }  } |

Abstract3Driver.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 Question1Abstract3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Abstract3Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Abstract\_b ab = new Abstract\_b(4);  System.out.println("Sum is :" + ab.sum());  System.out.println("Multiplication is :" + ab.multiply());  }  } |

In this model , we are characterizing a class as abstract with every abstract technique and we are carrying out those abstract strategies in its subclass. Alongside that execution we are announced a variable and by utilizing defined constructor we instated the instance variable.

In driver class we are made an object for sub class named Abstract\_b and invoking the methods sum() and increase.

Output:

|  |
| --- |
|  |

Interface example :

Question1Interface1

Interface\_a.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 Question1Interface1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public interface Interface\_a {  int i = 10;  int sum();  int multiply();  int subtract();  } |

Interface\_b.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 Question1Interface1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface\_b implements Interface\_a {  private final int x;  public Interface\_b(int x) {  this.x = x;  }  @Override  public int sum() {  return i + x;  }  @Override  public int multiply() {  return i \* x;  }  @Override  public int subtract() {  return i - x;  }  public void displayResults() {  System.out.println("Sum of :" + i + ", " + x + " is :" + sum() + "\n"  + "Multiplication of :" + i + ", " + x + " is :" + multiply() + "\n"  + "Subtraction of :" + i + ", " + x + " is :" + subtract());  }  } |

Interface1Driver.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 Question1Interface1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface1Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Interface\_b in = new Interface\_b(4);  System.out.println("Sum is:" + in.sum());  System.out.println("Multiplication is :" + in.multiply());  System.out.println("Subtraction is :" + in.subtract());  System.out.println("Invoking the display method:");  in.displayResults();  }  } |

In this model .We are making an interface with one static variable of type int and proclaiming 3 abstract methods names sum(),multiply() and subtract().Next we are carrying out the interface and giving conduct to all abstract methods alongside abstract methods we are characterized a solid strategy, names displayresults().In driver class we are making an item for Interface\_b class and invoking all strategies around there. Naturally all techniques in interface are public , abstract we can preclude these watchwords while characterizing an interface.

Output:

|  |
| --- |
|  |

Interface using default method :

Question1Interface2

Interface\_a.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 Question1Interface2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public interface Interface\_a {  default void print() {  System.out.println("default method in InterfaceEx2 class");  }  public abstract void display();  } |

Interface\_b.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 Question1Interface2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public abstract class Interface\_b implements Interface\_a {  private final int x;  private final int y;  public int getX() {  return x;  }  public int getY() {  return y;  }  public Interface\_b(int x, int y) {  this.x = x;  this.y = y;  }  public int Multiply() {  return x \* y;  }  } |

Interface\_c.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 Question1Interface2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface\_c extends Interface\_b implements Interface\_a {  public Interface\_c(int x, int y) {  super(x, y);  }  @Override  public void print() {  System.out.println("in Interface\_c class");  System.out.println("calling default print() in Interface\_a");  super.print(); //To change body of generated methods, choose Tools | Templates.  }  @Override  public void display() {  System.out.println("Value of X is :" + getX() + "Y is :" + getY());  }  } |

Interface2Driver.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 Question1Interface2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface2Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Interface\_c ex = new Interface\_c(3, 8);  System.out.println("Multiplication is :" + ex.Multiply());  ex.display();  ex.print();  }  } |

In this model we are making an interface with one default technique and one abstract strategy. Next, we are embedding the interface and not carrying out the abstract technique in the interface .So , we are characterizing the class as abstract and characterizing a solid strategy in Interface\_b class. In Next class we are executing the interface Interface\_a and expanding the class Interface\_b and abrogating the print() and carrying out the display().In print() we are invoking the default print strategy in Interface\_a .In driver class we are making object for Interface\_c class and invoking all strategies.

Output:

|  |
| --- |
|  |

Class implementing multiple Interfaces :

Question1Interface3

Interface\_a.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 Question1Interface3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public interface Interface\_a {  void display();  } |

Interface\_b.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 Question1Interface3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public interface Interface\_b {  void print();  } |

Interface\_c.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 Question1Interface3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface\_c implements Interface\_b, Interface\_a {  @Override  public void print() {  System.out.println("declared in Interface\_b and implementing in Interface\_c");  }  @Override  public void display() {  System.out.println("declared in Interface\_a and implementing in Interface\_c");  }  } |

Interface3Driver.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 Question1Interface3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Interface3Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Interface\_c in = new Interface\_c();  in.display();  in.print();  }  } |

In this model, we are announced an interface with abstract method print(). In second interface we are pronounced a abstract method display().In next class we are executing the different interfaces and characterized the conduct for print() and display().In driver class we made item for Interface\_c class and invoked the methods.

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.

Question2

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 Gullapalli Sai Satyanarayana  \*/  public interface Colorable {  void howToColor();  } |

GeometricObject.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 Gullapalli Sai Satyanarayana  \*/  public abstract class GeometricObject {  public abstract double getArea();  public void howToColor() {  System.out.println("howToColor");  }  } |

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 Gullapalli Sai Satyanarayana  \*/  public class Square extends GeometricObject implements Colorable {  private double side1;  public Square(double side1) {  this.side1 = side1;  }  public double getSide1() {  return side1;  }  @Override  public double getArea() {  return getSide1() \* getSide1();  }  @Override  public void howToColor() {  System.out.println("Color all four sides");  }  } |

Question2Driver.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 Gullapalli Sai Satyanarayana  \*/  public class Question2Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  GeometricObject[] geo = new GeometricObject[5];  geo[0] = new Square(7);  geo[1] = new Square(10);  geo[2] = new Square(34.6);  geo[3] = new Square(15);  geo[4] = new Square(15);  for (int i = 0; i < geo.length; i++) {  System.out.println("Area of is :" + geo[i].getArea());  System.out.print("Invoking howtoColor():");  geo[i].howToColor();  }  }  } |

Output:

|  |
| --- |
|  |

UML Diagram:

|  |
| --- |
|  |

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

3(A) **Casting**: Casting means converting an object from one type to another type.

Different types of casting are :

1.Up Casting

2.Down Casting

Up Casting: Up Casting is the sort casting of sub class object to super class object. Up casting verifiably performed by compiler .It is firmly identified with the inheritance. The Is-A relationship between super class and sub class permitted us to utilize a sub class object anyplace that a super object would be permitted.

Question3UpCasting

Person.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 Question3UpCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Person {  private String firstName, lastName;  private int age;  public Person(String firstName, String lastName, int age) {  this.firstName = firstName;  this.lastName = lastName;  this.age = age;  }  public String getfirstName() {  return firstName;  }  public void setfirstName(String firstName) {  this.firstName = firstName;  }  public String getlastName() {  return lastName;  }  public void setlastName(String lastName) {  this.lastName = lastName;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String getPersonDetails() {  return getfirstName() + " " + getlastName();  }  @Override  public String toString() {  return "Human{" + "fName=" + firstName + ", lName=" + lastName + ", age=" + age + '}';  }  } |

Engineer.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 Question3UpCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Engineer extends Person {  private String eng\_branch, job\_Title, company\_name;  public Engineer(String eng\_branch, String job\_Title, String company\_name, String fName, String lName, int age) {  super(fName, lName, age);  this.eng\_branch = eng\_branch;  this.job\_Title = job\_Title;  this.company\_name = company\_name;  }  public String getEng\_branch() {  return eng\_branch;  }  public void setEng\_branch(String eng\_branch) {  this.eng\_branch = eng\_branch;  }  public String getJob\_Title() {  return job\_Title;  }  public void setJob\_Title(String job\_Title) {  this.job\_Title = job\_Title;  }  public String getCompany\_name() {  return company\_name;  }  public void setCompany\_name(String company\_name) {  this.company\_name = company\_name;  }  @Override  public String getPersonDetails() {  return super.getPersonDetails() + getJob\_Title() + " Engineer";  }  @Override  public String toString() {  return super.toString() + "Engineer{" + "eng\_branch=" + eng\_branch + ", job\_Title=" + job\_Title + ", company\_name=" + company\_name + '}';  }  } |

Teacher.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 Question3UpCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Teacher extends Person {  private String subject, college\_name, prefix;  private long salary;  public Teacher(String subject, String college\_name, long salary, String prefix, String fName, String lName, int age) {  super(fName, lName, age);  this.subject = subject;  this.college\_name = college\_name;  this.salary = salary;  this.prefix = prefix;  }  public String getSubject() {  return subject;  }  public void setSubject(String subject) {  this.subject = subject;  }  public String getCollege\_name() {  return college\_name;  }  public void setCollege\_name(String college\_name) {  this.college\_name = college\_name;  }  public String getPrefix() {  return prefix;  }  public void setPrefix(String prefix) {  this.prefix = prefix;  }  public long getSalary() {  return salary;  }  public void setSalary(long salary) {  this.salary = salary;  }  @Override  public String getPersonDetails() {  return getPrefix() + "." + super.getPersonDetails() + " Teacher";  }  @Override  public String toString() {  return super.toString() + "Teacher{" + "subject=" + subject + ", college\_name=" + college\_name + ", salary=$" + salary + '}';  }  } |

UpcastingDriver.java

|  |
| --- |
| /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class UpcastingDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Person p = new Person("Satya Chowdary", "Gullapalli", 24);  System.out.println(p.getPersonDetails());  Person s = new Engineer("CSE", " Software Developer", "Strateology", "Satya Chowdary", "Gullapalli", 20);  System.out.println(s.getPersonDetails());  Teacher t = new Teacher("C", "SIUE", 12000, "Dr", "Rakesh", "Chandra", 25);  System.out.println(t.getPersonDetails());  Person per = new Teacher("Database", "UHCL", 11000, "Dr", "Prudhvi", "Raju", 30);  System.out.println(per.getPersonDetails());  }  } |

In this model we are making three classes specifically Person , Engineer and Teacher. Here Person is Super class and Teacher and Engineer are sub classes for Person class. In driver class we are making an instance for individual class and invoking the getPersonDetails().In the following line , we are making a reference variable of type Person and putting away the reference to a case of Engineer class. Which is Up Casting. By utilizing that parent class reference we are summoning the getPersonDetails() of Engineer .Next we are starting up the Teacher class. Also, invoking getPersonDetails().In the following line we are performing Up Casting.

Output:

|  |
| --- |
|  |

Down Casting: Down casting is the type casting of super class reference to a sub class object. Software engineer needs to play out the down casting expressly. Prior to performing down casting we need to perform polymorphic substitution for two objects . Otherwise will get a ClassCastException at run time.

Question3DownCasting

Person.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 Question3DownCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Person {  private String firstName, lastName;  private int age;  public Person(String firstName, String lastName, int age) {  this.firstName = firstName;  this.lastName = lastName;  this.age = age;  }  public String getfirstName() {  return firstName;  }  public void setfirstName(String firstName) {  this.firstName = firstName;  }  public String getlastName() {  return lastName;  }  public void setlastName(String lastName) {  this.lastName = lastName;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  public String getPersonDetails() {  return getfirstName() + " " + getlastName();  }  @Override  public String toString() {  return "Human{" + "fName=" + firstName + ", lName=" + lastName + ", age=" + age + '}';  }  } |

Engineer.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 Question3DownCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Engineer extends Person {  private String eng\_branch, job\_Title, company\_name;  public Engineer(String eng\_branch, String job\_Title, String company\_name, String fName, String lName, int age) {  super(fName, lName, age);  this.eng\_branch = eng\_branch;  this.job\_Title = job\_Title;  this.company\_name = company\_name;  }  public String getEng\_branch() {  return eng\_branch;  }  public void setEng\_branch(String eng\_branch) {  this.eng\_branch = eng\_branch;  }  public String getJob\_Title() {  return job\_Title;  }  public void setJob\_Title(String job\_Title) {  this.job\_Title = job\_Title;  }  public String getCompany\_name() {  return company\_name;  }  public void setCompany\_name(String company\_name) {  this.company\_name = company\_name;  }  @Override  public String getPersonDetails() {  return super.getPersonDetails() + getJob\_Title() + " Engineer";  }  @Override  public String toString() {  return super.toString() + "Engineer{" + "eng\_branch=" + eng\_branch + ", job\_Title=" + job\_Title + ", company\_name=" + company\_name + '}';  }  } |

Teacher.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 Question3DownCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Teacher extends Person {  private String subject, college\_name, prefix;  private long salary;  public Teacher(String subject, String college\_name, long salary, String prefix, String fName, String lName, int age) {  super(fName, lName, age);  this.subject = subject;  this.college\_name = college\_name;  this.salary = salary;  this.prefix = prefix;  }  public String getSubject() {  return subject;  }  public void setSubject(String subject) {  this.subject = subject;  }  public String getCollege\_name() {  return college\_name;  }  public void setCollege\_name(String college\_name) {  this.college\_name = college\_name;  }  public String getPrefix() {  return prefix;  }  public void setPrefix(String prefix) {  this.prefix = prefix;  }  public long getSalary() {  return salary;  }  public void setSalary(long salary) {  this.salary = salary;  }  @Override  public String getPersonDetails() {  return getPrefix() + "." + super.getPersonDetails() + " Teacher";  }  @Override  public String toString() {  return super.toString() + "Teacher{" + "subject=" + subject + ", college\_name=" + college\_name + ", salary=$" + salary + '}';  }  } |

DowncastingDriver.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 Question3DownCasting;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class DowncastingDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Person p = new Person("Satya Chowdary", "Gullapalli", 24);  System.out.println(p.getPersonDetails());  Engineer eng = new Engineer("CSE", " Software Developer", "Strateology", "Satya Chowdary", "Gullapalli", 23);  System.out.println(eng.getPersonDetails());  p = eng;  eng = (Engineer) p;  System.out.println(eng.getPersonDetails());  }  } |

In this model we are making three classes specifically Person , Engineer and Teacher. Here Person is Super class and Teacher and Engineer are sub classes for Person class. In driver class we are making an instance for individual class and invoking the getPersonDetails().In the following line , we are making a case for Engineer class and summoning the getPersonDetails().In the following line we are playing out the polymorphic substitution. In the following line we are down casting the person reference type to Engineer type. Here we are changing over the super class example to sub class case type prior to changing over we are utilizing the class type. In the event that we didn’t make reference to the sub class type will get a blunder like inconsistent sorts. Also, invoking the getPersonDetails().which will print the getPersonDetails() in Engineer class.

Output:

|  |
| --- |
|  |

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?

|  |
| --- |
| is fruit instanceof Fruit ------ legal  Explanation : fruit is a reference variable of type Fruit , which is pointing to its subclass object.Here GoldenDelicious class is extending Fruit class.So this is true |

1. Is fruit instanceof Orange?

|  |
| --- |
| is fruit instanceof Orange --- illegal  Explanation : here the reference variable fruit is pointing to the GoldenDelicious class object.Which is not subclass of Orange. |

1. Is fruit instanceof Apple?

|  |
| --- |
| is fruit instanceof Apple --- legal  Explanation : here the reference variable fruit is pointing to the GoldenDelicious (which is the sub class of Apple class )class object. |

1. Is fruit instanceof GoldenDelicious?

|  |
| --- |
| fruit instanceof GoldenDelicious ---- legal  Explanation : fruit is a reference variable of type Fruit which is pointing to its sub class Type object GoldenDelicious. Here GoldenDelicious class is extending Fruit class.So this is true |

1. Is fruit instanceof McIntosh?

|  |
| --- |
| is fruit instanceof McIntosh ---- illegal  Explanation : fruit is a reference variable of type Fruit which is not pointing to its sub class Type object McIntosh. |

1. Is orange instanceof Orange?

|  |
| --- |
| is orange instanceof Orange ---- legal  Explanation : Orange is a reference variable of type Orange which is pointing to the Orange class object |

1. Is orange instanceof Fruit?

|  |
| --- |
| is orange instanceof Fruit ---- legal  Explanation : Orange is a reference variable of type Orange which is pointing to the Orange class(Which is a sub class of Fruit type) object. |

1. Is orange instanceof Apple?

|  |
| --- |
| is orange instanceof Apple -- illegal  Explanation : Orange is a reference variable of type Orange which is not pointing to the Apple class |

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

|  |
| --- |
| We can invoke the makeAppleCider() by utilizing the beneath statement  ((GoldenDelicious)fruit).makeAppleCider();  Here we are characterizing Fruit fruit=new GoldenDelicious();  First JVM checks if the method is accessible in GoldenDelicious .In our program method isn't accessible in GoldenDelicious class. It will check the any override techniques in its super class with name makeAppleCider();.Here we are not overriding makeAppleCider() method. So, in the event that we straight forwardly utilize the fruit. makeAppleCider() will get compilation error. So, we are changing over the fruit reference variable of type Fruit to GoldenDelicous and invoking makeAppleCider() method. By utilizing projecting we can invoke the makeAppleCider().  We can't invoke the makeAppleCider() by utilizing orange reference.  There is no is-a connection between the Apple class and Orange class. Which will create an compilation error. |

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

|  |
| --- |
| Yes, Orange can invoke the makeOrangeJuice().orange reference putting away the case of type Orange.  By utilizing fruit reference we can't invoke the makeOrangeJuice().because there is no is-a connection between these two classes. |

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

|  |
| --- |
| Is the statement Orange p = new Apple() --- Illegal ,  Explanation : We will get an compilation error like, inconsistent sorts Apple can't be changed over to Orange. Since there is no is-a connection between the Orange class and Apple class. So, the polymorphic substitution isn't permitted. |

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

|  |
| --- |
| Is the statement McIntosh p = new Apple() --- Illegal  Explanation: Here McIntosh is sub class and Apple is super class. Sub class reference can't store the super class occurrence variable straight forwardly. To offer this expression legitimate, we need to play out the casting for example changing the Apple over to McIntosh type.  Like : McIntosh p=(McIntosh)new Apple(); |

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

|  |
| --- |
| Is the statement Apple p = new McIntosh() --- legal  Explanation : Here we are utilizing polymorphic substitution. So, the assertion 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.

5(A) Question5

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 Question5;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Circle {  private double radius;  public Circle(double radius) {  this.radius = radius;  }  public double getRadius() {  return radius;  }  public double getArea() {  return Math.round(Math.PI \* getRadius() \* getRadius());  }  } |

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 Gullapalli Sai Satyanarayana  \*/  public class ComparableCircle extends Circle implements Comparable {  public ComparableCircle(double radius) {  super(radius);  }  @Override  public int compareTo(Object ob) {  Circle c = (Circle) ob;  return Double.compare(getArea(), c.getArea());  }  } |

ComparableCircleDriver.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;  import java.util.ArrayList;  import java.util.Collections;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class ComparableCircleDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ArrayList<Double> list = new ArrayList<>();  ComparableCircle com1 = new ComparableCircle(5);  ComparableCircle com2 = new ComparableCircle(10);  ComparableCircle com3 = new ComparableCircle(15);  ComparableCircle com4 = new ComparableCircle(20);  ComparableCircle com5 = new ComparableCircle(25);  list.add(com1.getArea());  list.add(com2.getArea());  list.add(com3.getArea());  list.add(com4.getArea());  list.add(com5.getArea());  System.out.println("Before sorting the area:");  for (Double d : list) {  System.out.println(d);  }  Collections.sort(list);  System.out.println("After sorting the area:");  for (Double al1 : list) {  System.out.println(al1);  }  if (com1.compareTo(com2) < 0) {  System.out.println("c2 is the larger instance");  } else if (com1.compareTo(com2) > 0) {  System.out.println("c1 is larger instance :");  } else {  System.out.println("Both are equal");  }  }  } |

Output:

|  |
| --- |
|  |

UML Diagram:

|  |
| --- |
|  |

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

6(A) Exception: An exception is an unexpected event that occurs during the execution of a program that disrupts the normal flow of instructions.

Exception handling: We have two choices when an exception object is created in our application.

a)Either we will handle it within the method.

b)Or we can pass it to the caller method to let it handle.

There are two types of exceptions:

a) checked exception.

b) unchecked exception.

difference between checked and unchecked exception:

1.checked exceptions are checked at compile-time.

2.unchecked exceptions are checked at runtime.

1. checked exceptions:

1.Checked exceptions are checked at compile-time.

2. It handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

Question6

1. By throwing FileNotFoundException

CheckedException1.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 Question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class CheckedException1 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException {  // TODO code application logic here  Scanner scannner;  scannner = new Scanner(new File("gullapalli.txt"));  while (scannner.hasNext()) {  int a = scannner.nextInt();  System.out.println(a);  }  }  } |

In this model , we are throwing a FileNotFoundException after the mark of main().and perusing the qualities from the myfile.txt and printing the qualities.

Output:

|  |
| --- |
|  |

1. By using try – catch block

CheckedException2.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 Question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class CheckedException2 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Checked Exception class");  Scanner sc;  try {  sc = new Scanner(new File("gullapalli.txt"));  while (sc.hasNext()) {  int a = sc.nextInt();  System.out.println(a);  }  } catch (FileNotFoundException ex) {  System.out.println("File not found Exception");  }  }  } |

In this model, we are utilizing try - catch block. We are keeping the error produced code in attempt block and writing a catch statement to get the thrown exception.

Output:

|  |
| --- |
|  |

CheckedException3.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 Question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class CheckedException3 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner sc;  try {  sc = new Scanner(new File("gulapalli.txt"));  while (sc.hasNext()) {  int a = sc.nextInt();  System.out.println(a);  }  } catch (FileNotFoundException ex) {  System.out.println("File not found Exception");  }  }  } |

In this model, I am attempting to peruse a document, which isn't accessible in its root organizer. So, the compiler throws an exception and catch statement gets the special case and print the statement in the catch().

Output:

|  |
| --- |
|  |

CheckedException4.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 Question6;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class CheckedException4 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws InterruptedException {  // TODO code application logic here  try {  Thread.sleep(600);  } catch (InterruptedException ie) {  System.out.println(ie);  }  System.out.println("here the thread is slept.");  }  } |

In this model, we need to thrown the InterruptedException or, in all likelihood we need to we have keep the code in try-catch block to stay away from these promoted Exception.

Output:

|  |
| --- |
|  |

b)Unchecked Exception:

(a)The classes which inherit RuntimeException are known as unchecked exceptions.

e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

(b)Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

UncheckedException1.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 Question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class UncheckedException1 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scanner;  try {  scanner = new Scanner(new File("gullapalli.txt"));  int abc = scanner.nextInt();  while (true) {  System.out.println(abc);  abc = scanner.nextInt();  }  } catch (FileNotFoundException ffe) {  System.out.println("File not found Exception");  } catch (java.util.NoSuchElementException nsee) {  System.out.println("No element found to read");  }  }  } |

In this model , we are perusing the qualities from myfile.txt and printing the qualities in record. In while statement we are not checking if the document has next value or not. If we were arrived at the finish of the record, we will get a mistake at run time which is no such element exception.

Output:

|  |
| --- |
|  |

UncheckedException2.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 Question6;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class UncheckedException2 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner sc = new Scanner(System.in);  try {  System.out.println("Enter a Value:");  int a = sc.nextInt();  System.out.println("Enter d value:");  double d = sc.nextDouble();  System.out.println("Enter b value:");  boolean b = sc.nextBoolean();  System.out.println("Values of a:" + a + ", d : " + d + " , b :" + b);  } catch (java.util.InputMismatchException ex) {  System.out.println("check the input type: " + ex);  }  }  } |

In this model, we are perusing the qualities by utilizing scanner, in run time there might be an opportunity to enter diverse datatype values which will cause the InputMismatchexception.

Output:

|  |
| --- |
|  |

UncheckedException3.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 Question6;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.ArrayList;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class UncheckedException3 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException {  // TODO code application logic here  Scanner scan = new Scanner(System.in);  Scanner scannner = new Scanner(new File("gullapalli.txt"));  ArrayList<Integer> list = new ArrayList<>();  while (scannner.hasNext()) {  int a = scannner.nextInt();  list.add(a);  }  try {  System.out.print("Enter the Index to print the value:");  int n = scan.nextInt();  System.out.println("Value is :" + list.get(n));  } catch (java.lang.Exception ex) {  System.out.println(ex);  }  }  } |

In this model , we are perusing an array values from a document. By utilizing scanner we are giving list number to print specific value of the index. In this model, there is opportunity to enter the worth which is more than the length of array.

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.

7(A) Question7

RandomIntegers.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 Question7;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class RandomIntegers {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  int randomArray[] = new int[100];  for (int i = 0; i < randomArray.length; i++) {  randomArray[i] = (int) (Math.random() \* 100);  }  Scanner sc = new Scanner(System.in);  try {  System.out.println("Enter the Index :");  int index = sc.nextInt();  System.out.println("value is :" + randomArray[index]);  } catch (ArrayIndexOutOfBoundsException exp) {  System.out.println("Out of Bounds");  }  }  } |

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.

8(A) The purpose behind declaring Exceptions is to deal with the various types of run time errors. On the off chance that we compose the program with right logic and tested everything went effectively. Yet, in certain situations our logic will come up short. Furthermore, the program will ended .On that circumstance by utilizing special case we can deal with Errors ,end isn't required. For instance, partition by a number , our program will run for each number with the exception of 0 , partition by 0 is indistinct , so the framework don't have the idea about the appropriate response and it gives a run time mistake . to stay away from these circumstance will utilize Exceptions we tosses an Arithmetic Exception. Some different situations are , on the off chance that we arrived at the End of the record it will throws an exception.

We can announce an exception toward the finish of the method signature by utilizing the throws keyword. Like,

public int divide() throws ArithmeticException{

//body of the method;

}

Indeed , we can proclaim various exceptions in method header . Every one of the exemptions are isolated by commas.

Public int divide() throws Arithmetic Exception , Exception{

//body of the method

}

1.Exceptions are events that occurs in the code.

2.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.

examples:

NullPointerException – When you try to use a reference that points to null.

ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.

Question8Exception1

Exception1.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 Question8Exception1;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Exception1 {  public void display() throws FileNotFoundException {  Scanner scan = new Scanner(new File("gullapalli.txt"));  }  } |

Exception1Driver.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 Question8Exception1;  import java.io.FileNotFoundException;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Exception1Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException {  // TODO code application logic here  Exception1 ex = new Exception1();  try {  ex.display();  } catch (Exception exc) {  System.out.println(exc);  }  }  } |

In this model, we are throwing a FileNotFoundException ,In the Exception1 class we are perusing a record by using scanner.Name of the document is file.txt, which isn't accessible in the root envelope. So, compiler will throws an error. So, compiler will check is there any catch statement to get the exception. First it will check for the FileNotFoundException, it isn't accessible , later it will check for its super class. In our model we referenced Exception, which is super class for all exceptions.

Output:

|  |
| --- |
|  |

Question8Exception2

Exception2.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 Question8Exception2;  import java.util.ArrayList;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Exception2 {  void multipleExceptions(ArrayList<Integer> a, int b, int c) throws ArithmeticException, IndexOutOfBoundsException {  if (a.size() > 9) {  throw new IndexOutOfBoundsException();  } else {  System.out.println("Prininting array list values:");  for (Integer a1 : a) {  System.out.println(a1);  }  }  if (b <= 0 || c <= 0) {  throw new ArithmeticException("Value should be greater than zero");  } else {  System.out.println("Division is of b ,c is :" + b / c);  }  }  } |

InvalidException.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 Question8Exception2;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class InvalidException extends Exception {  public InvalidException() {  }  public InvalidException(String message) {  super(message);  }  } |

Exception2Driver.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 Question8Exception2;  import java.io.File;  import java.io.FileNotFoundException;  import java.util.ArrayList;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Exception2Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws FileNotFoundException {  // TODO code application logic here  ArrayList<Integer> list = new ArrayList<>();  Scanner scanner = new Scanner(new File("gullapalli.txt"));  Scanner scan = new Scanner(System.in);  while (scanner.hasNext()) {  int i = scanner.nextInt();  list.add(i);  }  System.out.println("Enter b value :");  int b = scan.nextInt();  System.out.println("Enter c value:");  int c = scan.nextInt();  Exception2 ex = new Exception2();  try {  ex.multipleExceptions(list, b, c);  } 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");  boolean d = scan.nextBoolean();  if (d == true) {  list.add(24);  ex.multipleExceptions(list, b, c);  }  System.out.println("Done !!");  } catch (Exception ex2) {  System.out.println(ex2);  }  }  } |

In this model we are throwing the numerous exceptions, we have made a method named multipleException(list) and throwing ArithmeticException,IndexOutOfBoundsException, NotValidException. In the following line we are checking the list size which is greater than 9 or not .If the rundown is more than 9 then it will throw IndexOutOfBoundsException or probably we are printing the components in the list. In the following condition we are checking b, c qualities which is not exactly or equivalent to 0 or not. On the off chance that b,c anybody of the component is not exactly or equivalent 0 , we are throwing the ArithmeticException with a error message . In the following condition we are checking b is not exactly c or not if its actual we are throwing NotValidException which is a user defined Exception or, more than likely partitioning the b,c. In the driver class we referenced the catch statements to get all exceptions.

Output:

|  |
| --- |
|  |

|  |
| --- |
|  |

Question8Exception3

NegativeException.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 Question8Exception3;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class NegativeException extends Exception {  public NegativeException() {  }  public NegativeException(String message) {  super(message);  }  } |

NegativeExceptionDriver.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 Question8Exception3;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class NegativeExceptionDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scan = new Scanner(System.in);  System.out.print("Enter a value:");  int n = scan.nextInt();  if (n < 0) {  try {  throw new NegativeException(" value is less than 0");  } catch (NegativeException ne) {  System.out.println(ne);  }  } else {  System.out.println("Square is :" + (n \* n));  }  }  } |

In this model, we are perusing a n value from framework and checking if the worth is not more than 0.If the worth is more greater than 0 we are printing square of the number or, in all likelihood we are throwing a user defined exception with a error which is NegativeException and characterized the catch statement to get the NegativeException.

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.

9(A) Throw Keyword: Throw keyword is utilized to throw an exception for a square of code or inside a method. By utilizing throw keyword we can throw our own Exceptions. By utilizing throw we can throw just a single exception at a time. Throw keyword is trailed by instance variable.

Throws Keyword: By utilizing Throws Keyword we can thrown an exception in the mark of the method. Which is utilized to report an exception may happen while running the code. By utilizing throws we can throw numerous exceptions which are isolated by commas. Throws keyword is trailed by exception class name.

No, we can't throw various throw exceptions in one throw statement. By utilizing throw keyword we can throw single Exception at a time in one explanation.

Question9Throw1

Grade.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 Question9Throw1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Grade {  private double marks;  public Grade(double marks) {  this.marks = marks;  }  public double getMarks() {  return marks;  }  } |

InvalidException.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 Question9Throw1;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class InvalidException extends Exception {  public InvalidException() {  }  public InvalidException(String msg) {  super(msg);  }  } |

GradeDriver.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 Question9Throw1;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class GradeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  System.out.println("Enter student marks:");  int n = scanner.nextInt();  try {  Grade gr = new Grade(n);  if (gr.getMarks() < 0 || gr.getMarks() > 100) {  throw new InvalidException("Invalid Marks");  }  if (gr.getMarks() >= 90 && gr.getMarks() <= 100) {  System.out.println("Result is :A Grade");  } else if (gr.getMarks() < 90 && gr.getMarks() >= 80) {  System.out.println("Result is :B Grade");  } else if (gr.getMarks() < 80 && gr.getMarks() >= 70) {  System.out.println("Result is :C Grade");  } else {  System.out.println("Result is :Fail");  }  } catch (InvalidException ex) {  System.out.println(ex);  }  }  } |

In this model, we are checking the evaluation of a student. If the marks are under 0 or more than 100, it will throw a NotValidexception. Which is a user defined exception. In the driver class we characterized a catch statement to get the exception.

Output:

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Question9Throw2

Throw1Driver.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 Question9Throw2;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Throw1Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  System.out.print("Enter x value :");  double x = scanner.nextInt();  System.out.print("Enter y value :");  double y = scanner.nextInt();  if (y == 0) {  try {  throw new ArithmeticException("y value is not equal to 0");  } catch (ArithmeticException ex) {  System.out.println(ex);  }  } else {  System.out.println("Division is :" + (x / y));  }  }  } |

In this model we are throwing the ArithmeticException by utilizing throw keyword. We are checking the y value which is equivalent to 0 then we are throwing ArithmeticException with an error message. What’s more, getting the statement, or else we are printing the division of x,y.

Output:

|  |
| --- |
|  |

|  |
| --- |
|  |

Throw2Driver.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 Question9Throw2;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Throw2Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  System.out.print("Enter n value :");  int a = scanner.nextInt();  try {  System.out.println("Square of given number: " + a + " is :" + square(a));  } catch (ArithmeticException ae) {  System.out.println(ae);  }  }  public static int square(int n) throws ArithmeticException {  if (n == 0) {  throw new ArithmeticException("enter the value greater than 0");  } else {  return n \* n;  }  }  } |

In this model, we are throwing ArithmeticException . on the off chance that the entered value is 0 we are throwing the exception or, in all likelihood we are figuring out the given number.

Output:

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Throw3Driver.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 Question9Throw2;  import java.util.InputMismatchException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class Throw3Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws InputMismatchException {  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  try {  System.out.println("Enter a integer value :");  int i = scanner.nextInt();  System.out.println("Enter a boolean value :");  boolean boo = scanner.nextBoolean();  } catch (InputMismatchException ie) {  System.out.println(ie);  }  }  } |

In this model, we are throwing an InputMismatchException , we are proclaimed the exception in the mark of main(). We kept the mistake produced code in try block and characterized the catch block for InputMismatchException.

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.

10(A) 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.

Question10

Recursive Method : Example : Direct Recursion

DirectSum.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 Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class DirectSum {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scanner = new Scanner(System.in);  System.out.print("Enter n value:");  int a = scanner.nextInt();  System.out.println("Sum of " + a + " digits from 1 to " + a + " is:" + Sum(a));  }  private static int Sum(int n) {  if (n > 1) {  return n + Sum(n - 1);  } else {  return 1;  }  }  } |

In this model, we are characterized a sum(int) and adding the n digits. In sum(n) we are checking the condition if the n value is more prominent than 1. On the off chance that n is more prominent than 1 we are returning n+sum(n-1) for example we are calling a similar method once more .This interaction will proceed until the n value is 1 or under 1.If n is under 1 or equivalent to 1 we are returning 1 and not invoking the sum(n).In main method we are printing the amount of n digits.

Output:

|  |
| --- |
|  |

Indirect recursion

InDirectSum.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 Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class InDirectSum {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner sc = new Scanner(System.in);  System.out.print("Enter n value:");  int n = sc.nextInt();  System.out.println("Sum of " + n + " digits from 1 to " + n + " is:" + Sum(n));  }  private static int Sum(int n) {  if (n > 1) {  return n + Sum1(n - 1);  } else {  return 1;  }  }  private static int Sum1(int n) {  if (n > 1) {  return n + Sum(n - 1);  } else {  return 1;  }  }  } |

In this model, we are utilizing two static methods sum,sum1.In the two techniques we are checking the condition n greater than 1 or not. In sum(int n) we are checking n greater than n or not ,if the condition is genuine we are returning n and invoking the sum1(int n) method. In sum1(int n) it checks if the condition is valid .If condition is genuine it will return the worth of n and invoking the sum(int n).these two methods will invoke consistently until the n more prominent than 1 condition falls flat. Here we are calling sum(int n),sum1(int n) in a roundabout way and computing the amount of n digits.

Output:

|  |
| --- |
|  |

Infinite recursion

InfiniteRecursion.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 Question10;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class InfiniteRecursion {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.print("Enter n :");  Scanner scanner = new Scanner(System.in);  int a = scanner.nextInt();  try {  System.out.println(infiniteRec(a));  } catch (java.lang.StackOverflowError ex) {  System.out.println("infinite recursion" + ex);  }  }  private static String infiniteRec(int a) {  if (true) {  return "infinite recursion" + "\n" + infiniteRec(a + 1);  } else {  return "";  }  }  } |

In this model, we are characterizing a static method infiniteRec(int).And we are characterizing the condition which is in every case valid. Here we are characterizing a condition to stop the cycle. At the point when we are utilizing the infinite recursion java throws StackOverFlowError. Because , java powers to run the method too often and put the neighborhood factors and boundaries on the stack too often on the degrees stack runs out of its space so, it throws error.

Output:

|  |
| --- |
|  |

search (element) in a list using recursion

SearchElement.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 Question10;  import java.util.ArrayList;  import java.util.NoSuchElementException;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class SearchElement {  private ArrayList<Integer> list;  private static int i = 0;  public SearchElement(ArrayList<Integer> list) {  this.list = list;  }  public int search(int n) {  if (list.size() < i + 1) {  throw new NoSuchElementException(n + " Element is not available in the list");  }  if (list.get(i) == n) {  return i;  } else {  i = i + 1;  return search(n);  }  }  } |

SearchElementDriver.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 Question10;  import java.util.ArrayList;  import java.util.NoSuchElementException;  import java.util.Scanner;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class SearchElementDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ArrayList<Integer> list = new ArrayList<>();  list.add(10);  list.add(11);  list.add(15);  list.add(31);  list.add(2);  list.add(7);  try {  System.out.println("Enter element to search :");  Scanner scanner = new Scanner(System.in);  int e = scanner.nextInt();  SearchElement search = new SearchElement(list);  System.out.println("Element: " + e + " Found at index:" + search.search(e));  } catch (NoSuchElementException ex) {  System.out.println(ex);  }  }  } |

Output:

|  |
| --- |
|  |

|  |
| --- |
|  |

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

11(A) equals() and hashCode() :

Java equals() and hashCode() methods are present in Object class. So every java class gets the default implementation of equals() and hashCode()

equals(Object obj):

1.A method provided by java.lang.Object that indicates whether some other object passed as an argument is "equal to" the current instance.

2.The default implementation provided by the JDK is based on memory location — two objects are equal if and only if they are stored in the same memory address.

Some principles of equals() method of Object class :

1.Reflexive : for any reference value a, a.equals(a) should return true.

2.Symmetric : for any reference values a and b, if a.equals(b) should return true then b.equals(a) must return true.

3.Transitive : for any reference values a, b, and c, if a.equals(b) returns true and b.equals(c) returns true, then a.equals(c) should return true.

4.Consistent : for any reference values a and b, multiple invocations of a.equals(b) consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.

Java hashCode():

Java Object hashCode() is a native method and returns the integer hash code value of the object.

Question11

HashcodeEquals.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 Gullapalli Sai Satyanarayana  \*/  public class HashcodeEquals {  public String name;  public int id;  HashcodeEquals(String name, int id) {  this.name = name;  this.id = id;  }  @Override  public boolean equals(Object obj) {  // checking if both the object references are  // referring to the same object.  if (this == obj) {  return true;  }  // it checks if the argument is of the  // type equal by comparing the classes  // of the passed argument and this object.  // if(!(obj instanceof equal)) return false;  if (obj == null || obj.getClass() != this.getClass()) {  return false;  }  // type casting of the argument.  HashcodeEquals geek = (HashcodeEquals) obj;  // comparing the state of argument with  // the state of 'this' Object.  return (geek.name == this.name && geek.id == this.id);  }  @Override  public int hashCode() // This method returns the hash code value for the object on which this method is invoked.  {  // We are returning the equal\_id as a hashcode value.  // we can also return some other calculated value or may  // be memory address of the Object on which it is invoked.  // it depends on how you implement hashCode() method.  return this.id;  }  } |

HashCodeDriver.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 Gullapalli Sai Satyanarayana  \*/  public class HashCodeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  // creating the Objects of Equal class.  HashcodeEquals g1 = new HashcodeEquals("aa", 1);  HashcodeEquals g2 = new HashcodeEquals("aa", 1);  // comparing above created Objects.  if (g1.hashCode() == g2.hashCode()) {  if (g1.equals(g2)) {  System.out.println("Both Objects are equal. ");  } else {  System.out.println("Both Objects are not equal. ");  }  } else {  System.out.println("Both Objects are not equal. ");  }  }  } |

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.

12(A) Question12

Employee.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 Question12;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  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 empId + " " + empName + " " + empSalary;  }  @Override  public int compareTo(Employee emp) {  return Integer.compare(empId, emp.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.

EmployeeDriver.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 Question12;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  /\*\*  \*  \* @author Gullapalli Sai Satyanarayana  \*/  public class EmployeeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  ArrayList<Employee> empList = new ArrayList<Employee>();  Employee empobj1 = new Employee(1, "SaiNath", 1111.1);  empList.add(empobj1);  Employee empobj2 = new Employee(2, "Satya", 2222.2);  empList.add(empobj2);  Employee empobj3 = new Employee(3, "Vishal", 3333.3);  empList.add(empobj3);  Employee empobj4 = new Employee(4, "Bharath", 4444.4);  empList.add(empobj4);  Employee empobj5 = new Employee(5, "Manoj", 5555.5);  empList.add(empobj5);  Employee empobj6 = new Employee(6, "Vishnu", 6666.6);  empList.add(empobj6);  System.out.println("Displaying original Employee List Elements(before sorting):");  for (Employee employeeList1 : empList) {  System.out.println(employeeList1);  }  Collections.sort(empList);  System.out.println("Sorted the list by using empId(Natural Ordering):");  for (Employee employeeList1 : empList) {  System.out.println(employeeList1);  }  Collections.sort(empList, new Comparator<Employee>() {  @Override  public int compare(Employee emp1, Employee emp2) {  if (emp1.getEmpSalary() < emp2.getEmpSalary()) {  return -1;  }  if (emp1.getEmpSalary() > emp1.getEmpSalary()) {  return 1;  } else {  return 0;  }  }  });  System.out.println("Sorting the Employee details by Salary:");  for (Employee employeeList : empList) {  System.out.println(employeeList);  }  Collections.sort(empList, new Comparator<Employee>() {  @Override  public int compare(Employee emp1, Employee emp2) {  if (emp1.getEmpName().compareTo(emp2.getEmpName()) == 0) {  return emp1.getEmpName().compareTo(emp2.getEmpName());  } else {  return emp1.getEmpName().compareTo(emp2.getEmpName());  }  }  });  System.out.println("Sorting the Employee details by name:");  for (Employee employeeList1 : empList) {  System.out.println(employeeList1);  }  }  } |

Output:

|  |
| --- |
|  |

|  |
| --- |
| GITHUB Link : <https://github.com/Satyachowdary97/OOP-Spring-2021-FinalExam> |