**Final Exam Instructions**

**OBJECT-ORIENTED PROG**

* This is a take-home exam. You can use any resources that are available for you to finish this exam, except
  + Outsourcing the exam to any person or to any third party websites
  + Copying from other students work
  + Copying direct quotes from the books or internet
* Do not lose your opportunity to learn while working on the exam. Understand the concept and write answers on your own.
* Usually, in life, we have several choices. Unfortunately, you don’t have any choice on this exam. You have to answer all the questions and each part of the problem.
* All the topics on this exam were discussed in class . So, you cannot claim that the questions are out of the syllabus!
* Refer to Microsoft Word tutorials for proper formatting
* Points will be deducted for grammatical and spelling mistakes
* No two brains think alike unless you are soulmates. Definitely your answers will not be same as other students.
* Read the code of academic integrity before you start the exam. <https://www.nwmissouri.edu/policies/academics/Academic-Integrity.pdf>
* Push your source code to GitHub and provide your GitHub link at the end of the document and in the comment section.
* Don’t use examples that already explained in class or worksheets.
* Provide the input and output screenshots for every program.

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

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

**Answer for the Question1**:

Data abstraction is the procedure of hiding the implementation and displaying limited useful data to the user. Abstraction can be accomplished with either abstract classes or interfaces.

An **Abstract class** is a class which is declared with the keyword Abstract. It is a restricted class which cannot be used for creating objects.

A boundary or interconnection between two entities is referred to as an **interface**. Interfaces specify what a class should do but not how. It is the blueprint of the class. In an interface, all the methods are abstract and public. Interfaces can have constants but no instance variables. Recently, from java 1.8 default methods are introduced in the interfaces. A class implementing an interface, inherits the abstract methods of the interface.

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Similarities:   1. Abstract classes have abstract methods. 2. Abstract classes cannot be instantiated. 3. Don’t have default constructor. 4. All the abstract methods must be overridden by subclass. | Similarities:   1. Interfaces can have abstract methods. 2. Interface can not be instantiated. 3. Don’t have default constructor. 4. All the methods of the interface must be overridden by the implementing sub class. |
| Differences:   1. an abstract class can be extended using the keyword “extends”. 2. Abstract class does not support multiple inheritance as we cannot extend more than one class. 3. Abstract classes can have normal non-abstract methods. 4. No methods are created default with the word ‘default’. 5. An abstract class may contain non-final variables. 6. Abstract class can have final, non-final, static and non-static variables. 7. Abstract class can provide the implementation of the interface. | Differences:   1. A Java interface can be implemented using the keyword “implements”. 2. We can achieve multiple inheritance using interface as we can implement multiple interfaces to any class. 3. Interfaces cannot have non-abstract methods. 4. Interfaces can have default methods created with the word ‘default’. 5. Variables declared in a Java interface are by default final. 6. The interface has only static and final variables. 7. Interface can’t provide the implementation of an abstract class. |

Multiple inheritance isn't supported in Java, so you can't have a class that implements two abstract classes at the same time. When you want several classes to implement the interface and don't want to inherit default behavior, an interface is **preferable** to an abstract class.  **Interfaces** are better in situations in which you do not need to inherit implementation from a base class. In addition, abstract classes cannot provide data abstraction 100% where as interfaces provide 100% abstraction.

**Code for Example 1:**

ItIndustry Interface

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public interface ItIndustry {  public static final double hours=20.0;  public abstract double calculatesalary();        } |

Abstract Class Tester

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public abstract class Tester implements ItIndustry  {  private String fname,lname;  public Tester(String fname, String lname) {  this.fname = fname;  this.lname = lname;  }  public String getFname() {  return fname;  }  public void setFname(String fname) {  this.fname = fname;  }  public String getLname() {  return lname;  }  public void setLname(String lname) {  this.lname = lname;  }  @Override  public String toString() {  return "Tester{" + "fname=" + fname + ", lname=" + lname + '}';  }    public abstract void testingType();  } |

Administrative Tester

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class AdministrativeTester extends Tester{  @Override  public void testingType() {  System.out.println("Testing Type: Administrative Testing");  }  public AdministrativeTester(String fname,String lname) {  super(fname,lname);  }  @Override  public double calculatesalary() {  return ItIndustry.hours\*30.0;  }    } |

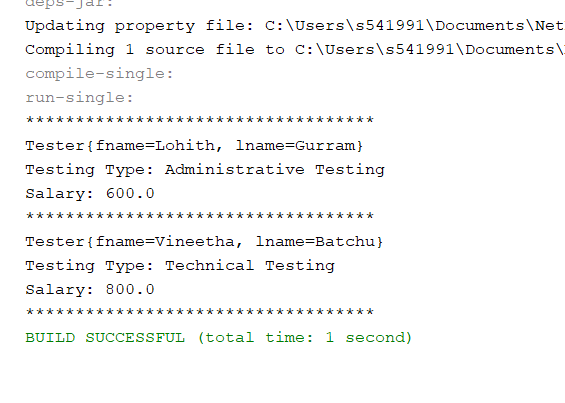
Technical Tester

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class TechnicalTester extends Tester{  @Override  public void testingType() {  System.out.println("Testing Type: Technical Testing");  }  public TechnicalTester(String fname,String lname) {  super(fname,lname);  }  @Override  public double calculatesalary() {  return ItIndustry.hours\*40.0;  }    } |

Driver Class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class TesterDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Tester t1=new AdministrativeTester("Lohith", "Gurram");  System.out.println(t1);  t1.testingType();  System.out.println("Salary: "+t1.calculatesalary());  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    Tester t2=new TechnicalTester("Vineetha", "Batchu");  System.out.println(t2);  t2.testingType();  System.out.println("Salary: "+t2.calculatesalary());  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");      }    } |

**Output for Example1 in Question1:**



**Explanation for Example1 in Question1:**

In this Example,

* An Interface named ItIndustry is created with a constant named hours and an abstract method calculatesalary().
* A new abstract class named Tester is created which implements interface ItIndustry with two String variables fname,lname and it has an abstract method testingType().
* Sub Classes AdministrativeTester,TechnicalTester which extends Tester the class an implements calculatesalary() and testingType().
* Now in the Driver Class, objects for both the sub class are created. If we invoke calculatesalary() and testingType()methods, sub class will be invoked because implementation for these provide in the sub classes.

**Example2 for Interfaces:**

**Code**

Interface1

|  |
| --- |
| /\*  \* 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 question1Example2Interface;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public interface Dress {  default void clothingtype(){  System.out.println("Clothing Type: Dress");  }  } |

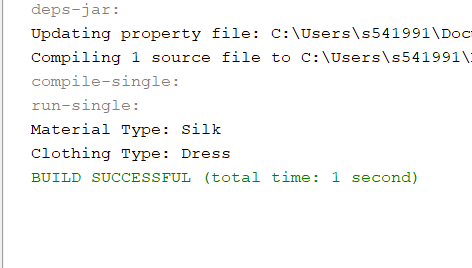
Interface2

|  |
| --- |
| /\*  \* 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 question1Example2Interface;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public interface MaterialSilk {  default void materialtype(){  System.out.println("Material Type: Silk");  }  } |

Driver Class implementing both the interfaces

|  |
| --- |
| /\*  \* 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 question1Example2Interface;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class SilkDress implements MaterialSilk,Dress{  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  SilkDress s=new SilkDress();  s.materialtype();  s.clothingtype();  }    } |

**Output for Example2 Interfaces:**



**Explanation for Example 2 Interfaces:**

This is an example of multiple inheritance which allows a class to implement two interfaces. In this Example, Dress and MaterialSilk are the interfaces with default methods clothingtype,materialtype() resectively. Now SilkDress class will implemement both the interfaces. In the main method when we create an object both the method implements in the class SilkDress are invoked.

**Example2 for Abstract Classes:**

**Code**

Abstract Class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1Example2AbstractClass;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public abstract class Music {  public void display() {  System.out.println("In Music Class");  }  public abstract void getMusicType();  } |

Subclass 1

|  |
| --- |
| /\*  \* 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 question1Example2AbstractClass;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Rock extends Music {  @Override  public void getMusicType() {  System.out.println("Music Type : Rock");  }  } |

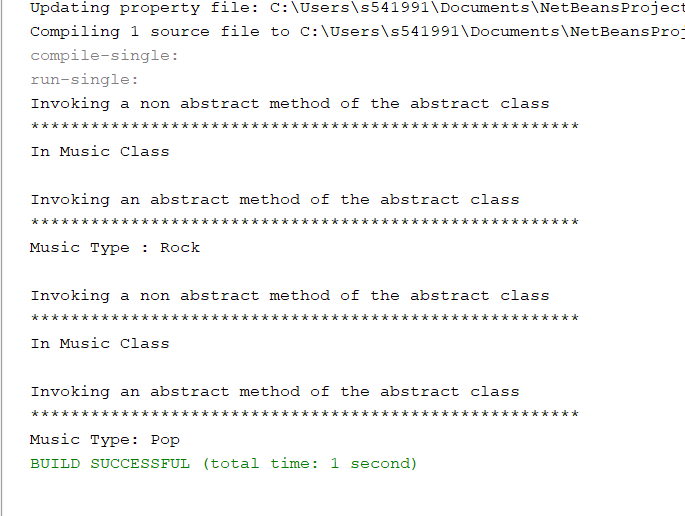
SubClass 2

|  |
| --- |
| /\*  \* 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 question1Example2AbstractClass;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Pop extends Music{  @Override  public void getMusicType() {  System.out.println("Music Type: Pop");  }    } |

Driver Class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question1Example2AbstractClass;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Rock r1=new Rock();  System.out.println("Invoking a non abstract method of the abstract class");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  r1.display();  System.out.println("\nInvoking an abstract method of the abstract class");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  r1.getMusicType();      //from another sub class  Pop p1=new Pop();  System.out.println("\nInvoking a non abstract method of the abstract class");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  p1.display();  System.out.println("\nInvoking an abstract method of the abstract class");  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  p1.getMusicType();  }    } |

**Output for Example 2 Abstract Classes:**



**Explanation for Example2 Abstract Classes:**

* Create an abstract class named Music with a normal method display() and an abstract method getMusicType ().
* Create sub classes Pop and Rock using extends key word and implement the abstract method getMusicType ().
* Now in the main class, we will be creating objects for the sub classes(Pop, Rock) and invoke the non-abstract methods of super class (as abstract classes cannot be instantiated) and abstract methods that were implemented in the sub class.

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

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

**Answer for the Question2:**

Colorable Interface

|  |
| --- |
| /\*  \* 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 Vineetha Batchu  \*/  public interface Colorable {  void howToColor();    } |

Geometric Object abstract class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question2;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public abstract class GeometricObject {  public abstract double area();  } |

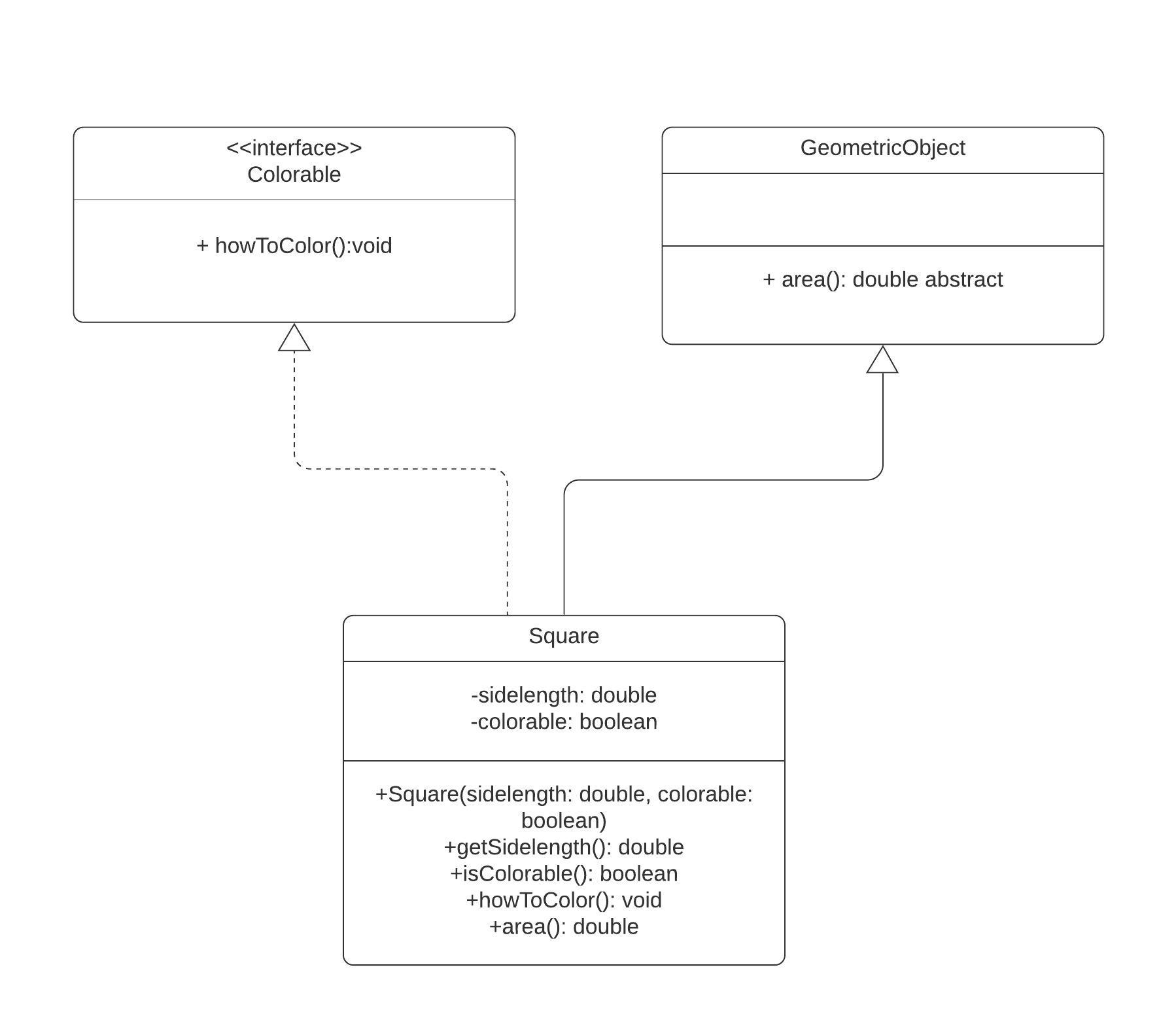
Square Class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question2;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Square extends GeometricObject implements Colorable {  private double sidelength;  private boolean colorable;  public Square(double sidelength, boolean colorable) {  this.sidelength = sidelength;  this.colorable = colorable;  }  public double getSidelength() {  return sidelength;  }  public boolean isColorable() {  return colorable;  }  @Override  public void howToColor() {  System.out.println("Color all four sides");  }  @Override  public double area() {  return sidelength \* sidelength;  }  } |

Driver Class

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question2;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  GeometricObject[] squares = {new Square(8, true), new Square(20, false),  new Square(25, true), new Square(30, true), new Square(55, false)};  for (GeometricObject o : squares) {  System.out.println("-----------------------------");  System.out.println("Area is " + o.area());  Square s = (Square) o;  if (s.isColorable()) {  s.howToColor();  }  System.out.println("-----------------------------");    }  }  } |

**UMLdiagram for Question 2:**



**Output Screenshot for Question 2:**

****

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

**Answer for question3:**

**Definition:** The process of converting the value of one data type to another data type is called type casting. There are two types of casting

1. **Widening Type Casting**
2. **Narrowing Type Casting**

**Widening Type Casting:** Widening style casting is the process of converting a lower data type to a higher one. It's also known as casting down or implicit conversion. It's done automatically. It is safe because there is no risk of data loss.

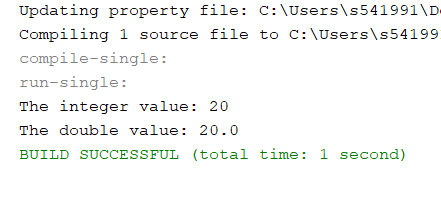
**Example for Widening Type Casting:** In this example int is converted to double and this is done automatically.

**Explanation for the below example:** In this example, an integer variable ‘number’ is created. Now this integer is converted into double by assigning. Java does this conversion automatically because it is widening type.

Code

|  |
| --- |
| /\*  \* 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 question3widening;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class WideningDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  // create integer type variable  int number = 20;  System.out.println("The integer value: " + number);  // convert integer into double type  double newdouble = number;  System.out.println("The double value: " + newdouble);  }  } |

**Output for the above example1:**

****

**Narrowing Type Casting:** Narrowing type casting is the process of converting a higher data type to a lower one. It's also known as casting up or explicit conversion. The programmer performs this task manually. The compiler will give a compile-time error if we do not perform casting.

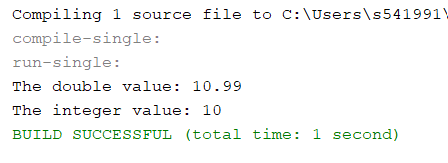
**Example for Narrowing Type Casting(Example 2)** In this example double is converted to int.

**Explanation for the below example:** In this example, a double variable ‘number’ is created. Now this double is converted into int by assigning using paranthesis int convertedNumber = (int) number.

Code

|  |
| --- |
| /\*  \* 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 quetion3Narrowing;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class NarrowingDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  double number = 10.99;  System.out.println("The double value: " + number);  // convert into int type  int convertedNumber = (int) number;  System.out.println("The integer value: " + convertedNumber);  }  } |

Output for the Example2:



Example 3: This type casting is applicable for objects too. Let us consider the following example where a super class object is converted into sub class object through typecasting.

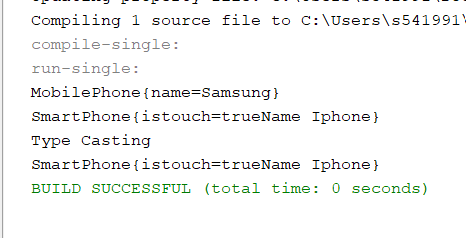
**Explanation for Example3**: In this example, SmartPhone s1=(SmartPhone)m; using this statement sub class object m is converted to (SmartPhone) sub class object. But for casting m=s; this assigning statement is mandatory. If we assign any random class object to the super class, then Class Cast Exception will occur.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question3;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class MobilePhone {  private String name;  public MobilePhone(String name) {  this.name = name;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public String toString() {  return "MobilePhone{" + "name=" + name + '}';  }      } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question3;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class SmartPhone extends MobilePhone {  private boolean istouch;  public SmartPhone(boolean istouch, String name) {  super(name);  this.istouch = istouch;  }  public boolean isIstouch() {  return istouch;  }  public void setIstouch(boolean istouch) {  this.istouch = istouch;  }  @Override  public String toString() {  return "SmartPhone{" + "istouch=" + istouch + "Name "+super.getName()+ '}';  }      } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question3;  /\*\*  \*  \* @author Vineetha Batchu  \*/  public class Driver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  MobilePhone m=new MobilePhone("Samsung");  System.out.println(m);  SmartPhone s=new SmartPhone(true, "Iphone");  System.out.println(s);  m=s;  SmartPhone s1=(SmartPhone)m;  System.out.println("Type Casting");    System.out.println(s1);  }  } |

**Output for the Example3**:



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.

a. Is fruit instanceof Fruit? True

b. Is fruit instanceof Orange? False

c. Is fruit instanceof Apple? True

d. Is fruit instanceof GoldenDelicious? True

e. Is fruit instanceof McIntosh? False

f. Is orange instanceof Orange? True

g. Is orange instanceof Fruit? True

h. Is orange instanceof Apple? False

i. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method? Fruit can invoke the makeAppleCider() because fruit is an instance of Apple, but orange cannot invoke makeAppleCider() because orange is not an instance of apple.

j. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method? Orange can invoke this method, because orange is an instance of Orange

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

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

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

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.
2. (15-Points) What is an exception? What are checked and unchecked exceptions? Explain and demonstrate with examples.
3. (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.

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.
2. (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.
3. (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.
4. (10-Points) Write a java program that illustrates how equals() and hashCode() methods work? Explain your code in comments.
5. (15-Points) Design Employee class and Employee driver class as follows:
6. **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.