**Final Exam Instructions**

**OBJECT-ORIENTED PROG**

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

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

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

**Abstract Classes:**

**Classes** that are declared using **abstract** keyword are known as **abstract classes**. It can have **abstract** methods (methods without body) as well as concrete methods (regular methods with body). An **abstract class** cannot be instantiated, which means you are not allowed to create an object of it.

In general, if we want to implement/declare some method which is used commonly used among the other classes we will declare it in abstract class. A normal class (non-abstract class) cannot have abstract methods. since we do not have a constructor, it is not allowed to create an object of abstract class. A class derived from the abstract class must implement all those methods that are declared as abstract in the parent class. If a child does not implement all the abstract methods of the abstract parent class, then the child class must need to be declared abstract as well.

**Interfaces:**

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

An interface is one of the ways to achieve abstraction which is hiding the implementation details and showing only functionality. Like a class, an Interface can have fields, method declarations and methods, but cannot be instantiated. The methods declared in an interface are by default abstract. To declare an interface, use the keyword interface. All the methods in an interface are public and all fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use keyword called **implements**.

**Similarities:**

* We cannot create object for both abstract classes and interfaces.
* All the abstract methods declared in the abstract and interfaces must be overridden by subclass/child Class.
* Both abstract and interface will not have default constructor.
* By using both we can go for dynamic polymorphism.
* Both abstract and interface can contain static and final variable.
* Both abstract and interface provide static method implementation.

**Differences:**

|  |  |
| --- | --- |
| **ABSTRACT CLASS** | **INTERFACE CLASS** |
| Abstract class can have abstract and non-abstract methods | An Interface can only have abstract method. Since java 8, it can have default and static method |
| Abstract class does not support multiple inheritance | Interface supports multiple inheritance |
| Abstract class can have final, non-final, static, and non-static variables. | Interface has only static and final variables. |
| Abstract class can provide the implementation of interface. | Interface cannot 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 of an interface are public by default. |
| Abstract class cannot be instantiated but can be invoked | Interface cannot be instantiated or invoked |
| Abstract class has parametrized and non-parametrized constructor but do not have default constructor | Interface does not support the concept of constructor |

**Why Interfaces are preferred over abstract classes:**

Java does not support multiple inheritance which means you cannot have a **class** that implements two **abstract classes** at once. An **interface** is **better than** abstract **class** when you want multiple **classes** to implement that **interface** and when you do not have to inherit default behavior.

**Interfaces** are more flexible than base **classes** because you can define a single implementation that can **implement** multiple **interfaces**. **Interfaces** are better in situations in which you do not need to inherit implementation from a base **class**. **Interfaces** are useful in cases where you cannot **use class** inheritance.

**Examples:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class GroceryOrder implements GroceryStorePayments, MexicanFood {  @Override  public void acceptOnlinePayment() {  System.out.println("Thanks for Paying through Online");  }  @Override  public void acceptChequePayment() {  System.out.println("Thanks for Paying through Cheque");  }  @Override  public void acceptCashPayment() {  System.out.println("Thanks for Paying through cash");  }  @Override  public void getBeef() {  System.out.println("Thanks for ordering Beef");  }  @Override  public void getChedderChesse() {  System.out.println("Thanks for ordering Chedder Cheese");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface GroceryStorePayments {  public void acceptOnlinePayment();  public void acceptChequePayment();  public void acceptCashPayment();  } |

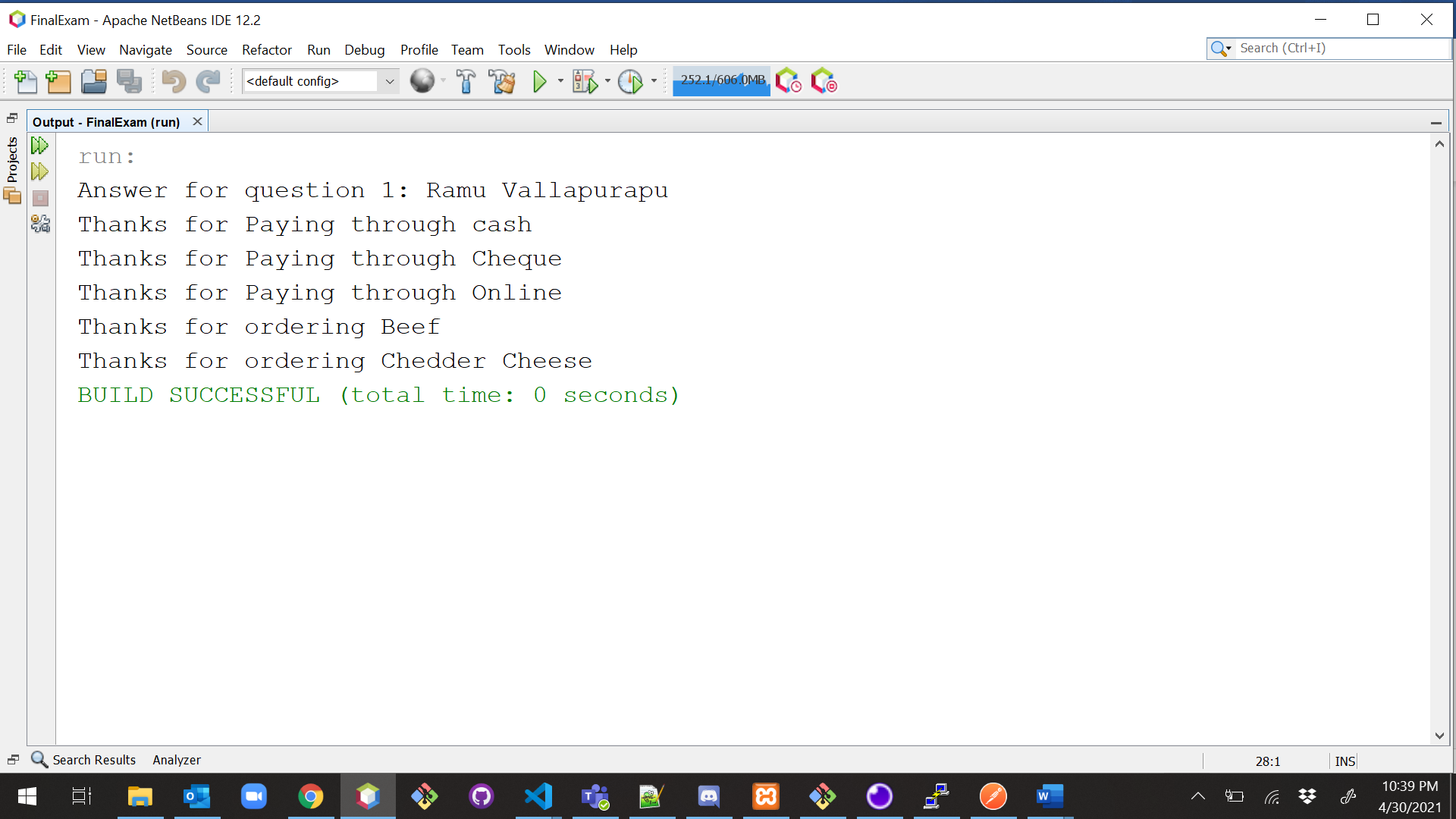
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface MexicanFood {  public void getBeef();  public void getChedderChesse();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class GroceryStoreDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question 1: Ramu Vallapurapu");  GroceryOrder g = new GroceryOrder();  g.acceptCashPayment();  g.acceptChequePayment();  g.acceptOnlinePayment();  g.getBeef();  g.getChedderChesse();  }  } |

**Explanation:**

In the above example, we have created two interfaces called **MexicanFood** and **GroceryStorePayments** and now it was implemented by class **GroceryOrder**. I have totally declared 5 methods as part of the interfaces, now the class which implements that interface must implement all the methods declared in that interface. Now we have implementation of methods declared in some other class and other classes can also implements that interface and can have their own implementation. In the above example we have achieved **multiple inheritance** which is not achieved normal classes.

**OutPut:**



**Example2:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  abstract class MyTest {  void calculate(int a, int b) {  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface Sports {  void play();  } |

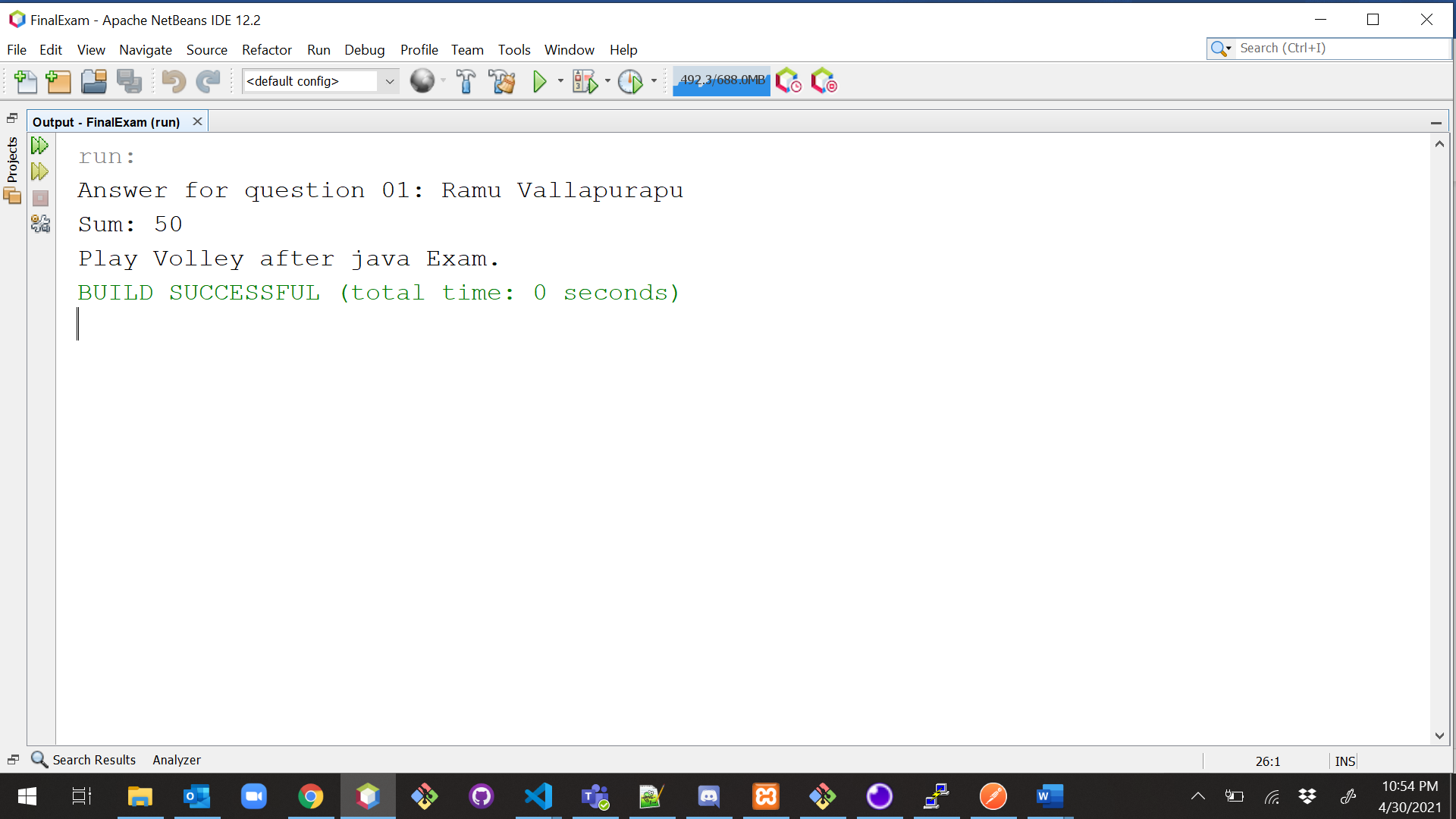
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class DailyRoutine extends MyTest implements Sports {  @Override  void calculate(int a, int b) {  int x = a + b;  System.out.println("Sum: " + x);  }  @Override  public void play() {  System.out.println("Play Volley after java Exam.");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question01;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class DailyRoutineDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question 01: Ramu Vallapurapu");  DailyRoutine a = new DailyRoutine();  a.calculate(20, 30);  a.play();  }  } |

**Explanation:**

In the above example, I have declared an abstract class with an abstract method (only method declaration) now it was extended by the DailyRoutine class since it is not an abstract class now class DailyRoutine must implement the method. With the help of the abstract class, we will achieve abstraction (data hiding) and as interfaces a class cannot extends multiple classes so interfaces are more **preferred** than abstract class

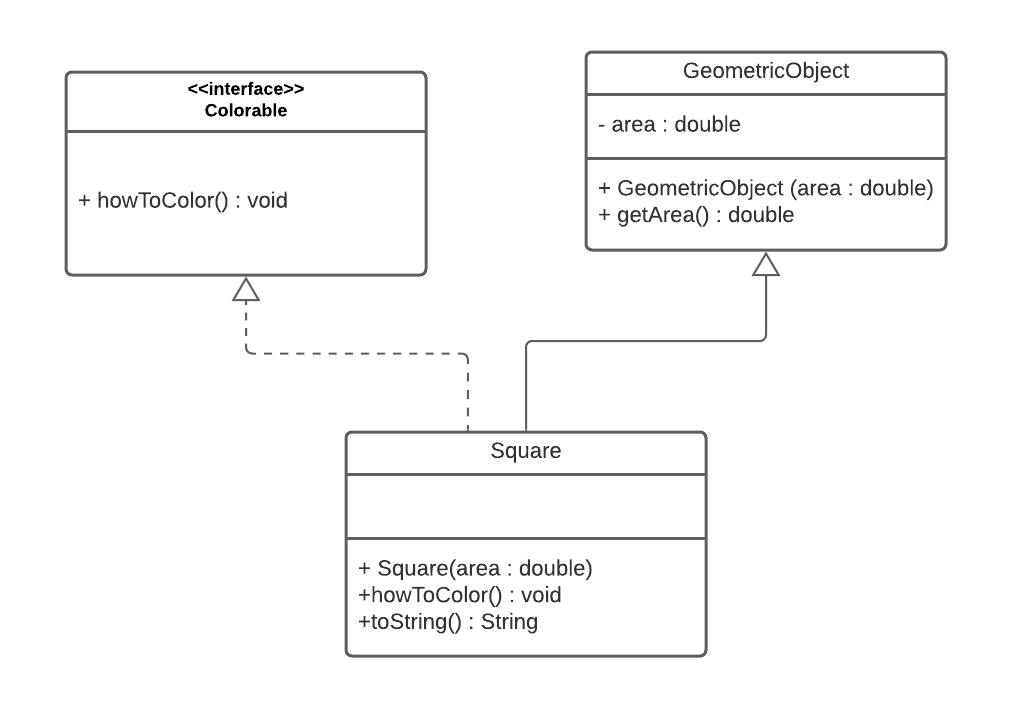
**Output:**



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.

**UML:**

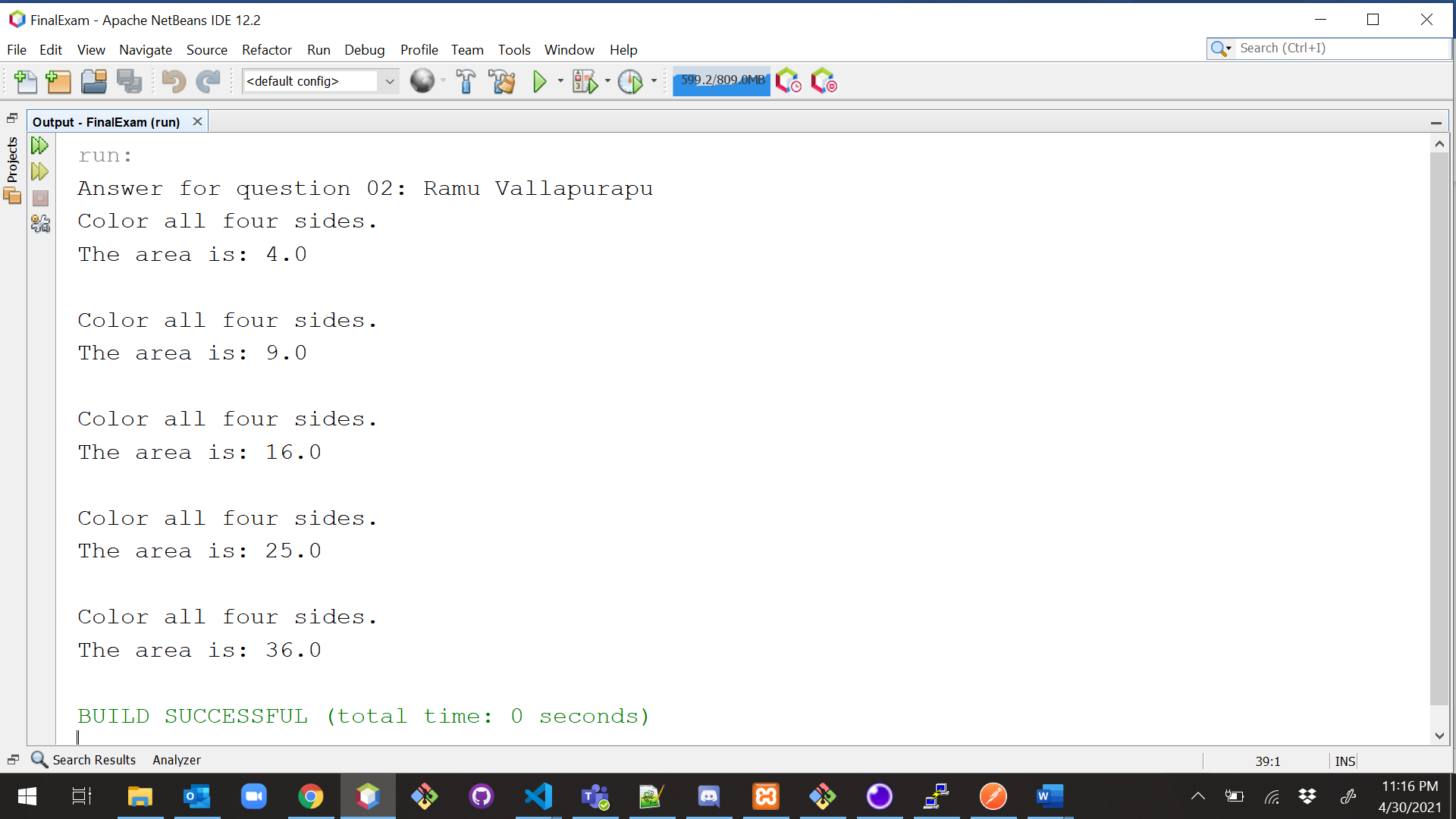


|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question02;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public interface Colorable {  public void howToColor();  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question02;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  abstract class GeometricObject {  private double area;  public double getArea() {  return area;  }  GeometricObject(double area) {  this.area = area;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question02;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Square extends GeometricObject implements Colorable {  public Square(double area) {  super(area);  }  @Override  public void howToColor() {  System.out.println("Color all four sides.");  }  @Override  public String toString() {  this.howToColor();  return ("The area is: " + this.getArea() + "\n");  }  } |

|  |
| --- |
| package question02;  /\*  \* 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.  \*/  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class TestDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question 02: Ramu Vallapurapu");  Square s1 = new Square(4);  Square s2 = new Square(9);  Square s3 = new Square(16);  Square s4 = new Square(25);  Square s5 = new Square(36);  GeometricObject[] objects = new GeometricObject[5];  objects[0] = s1;  objects[1] = s2;  objects[2] = s3;  objects[3] = s4;  objects[4] = s5;  for (GeometricObject obj : objects) {  obj.getArea();  System.out.println(obj.toString());  }  }  } |



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

**Casting** is the action of converting between two different data types such as converting an **int** to a **double** and vice versa. When we cast object only the reference type of the object is changed but not the actual object type.

Type casting is used to convert an **object** or **variable** of one type into another.

This is done in two ways.

A) Implicit conversion

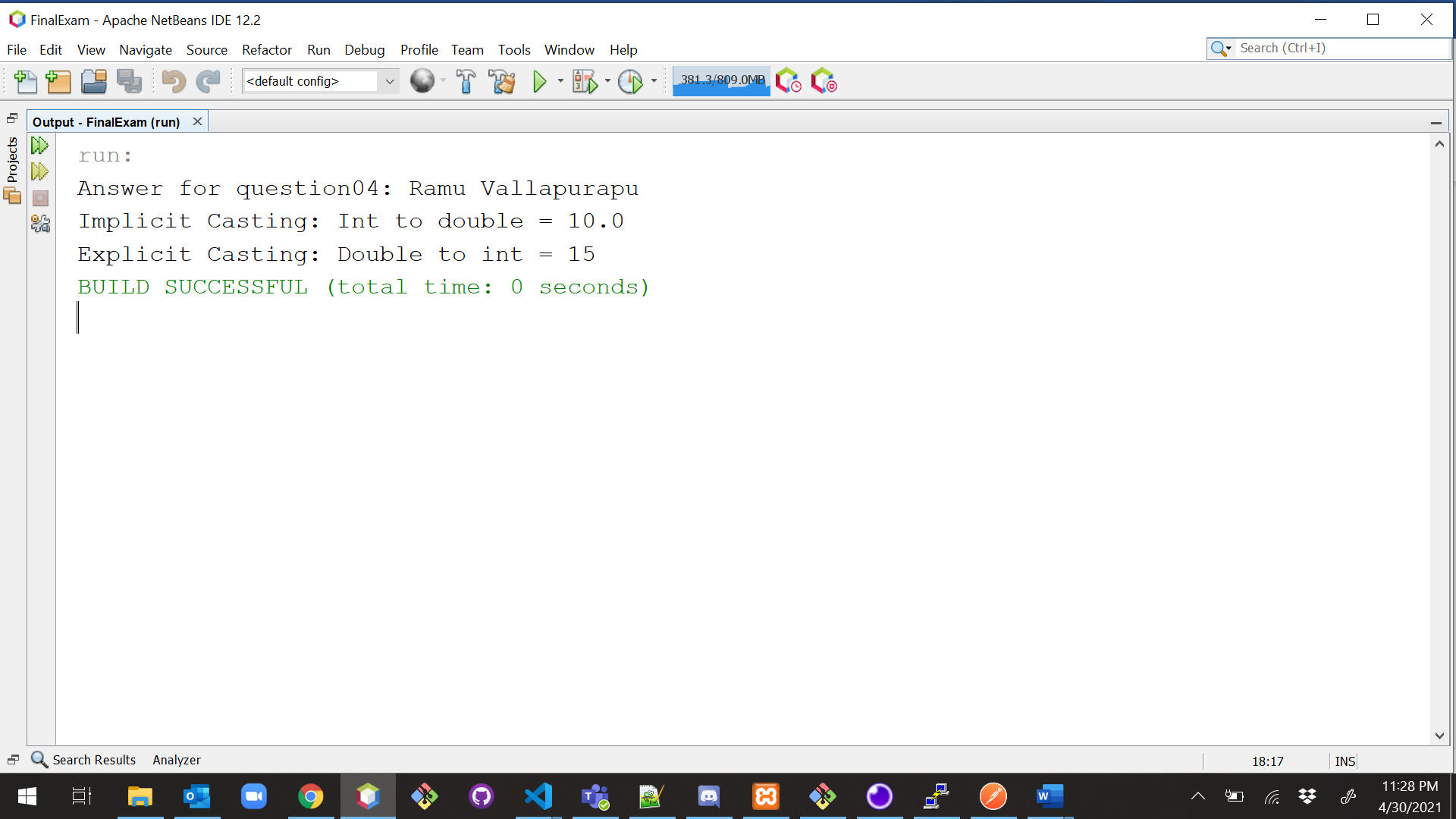
B) Explicit conversion

**Implicit Casting:** Upcasting is the typecasting **of a child object to a parent object**. Upcasting can be done implicitly. Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child class members using this feature. Instead of all the members, we can access some specified members of the child class. For instance, we can access the overridden methods.

**Explicit Casting:** If we want to assign a value of larger data type to a smaller data type, we perform explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.

**Examples:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class BasicCastingExample {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question04: Ramu Vallapurapu");  //Implicit Casting  int num = 10;  double d = num;  System.out.println("Implicit Casting: Int to double = " + d);  //Explicit Casting  int x = (int) 15.3;  System.out.println("Explicit Casting: Double to int = " + x);  }  } |



**Explanation:**

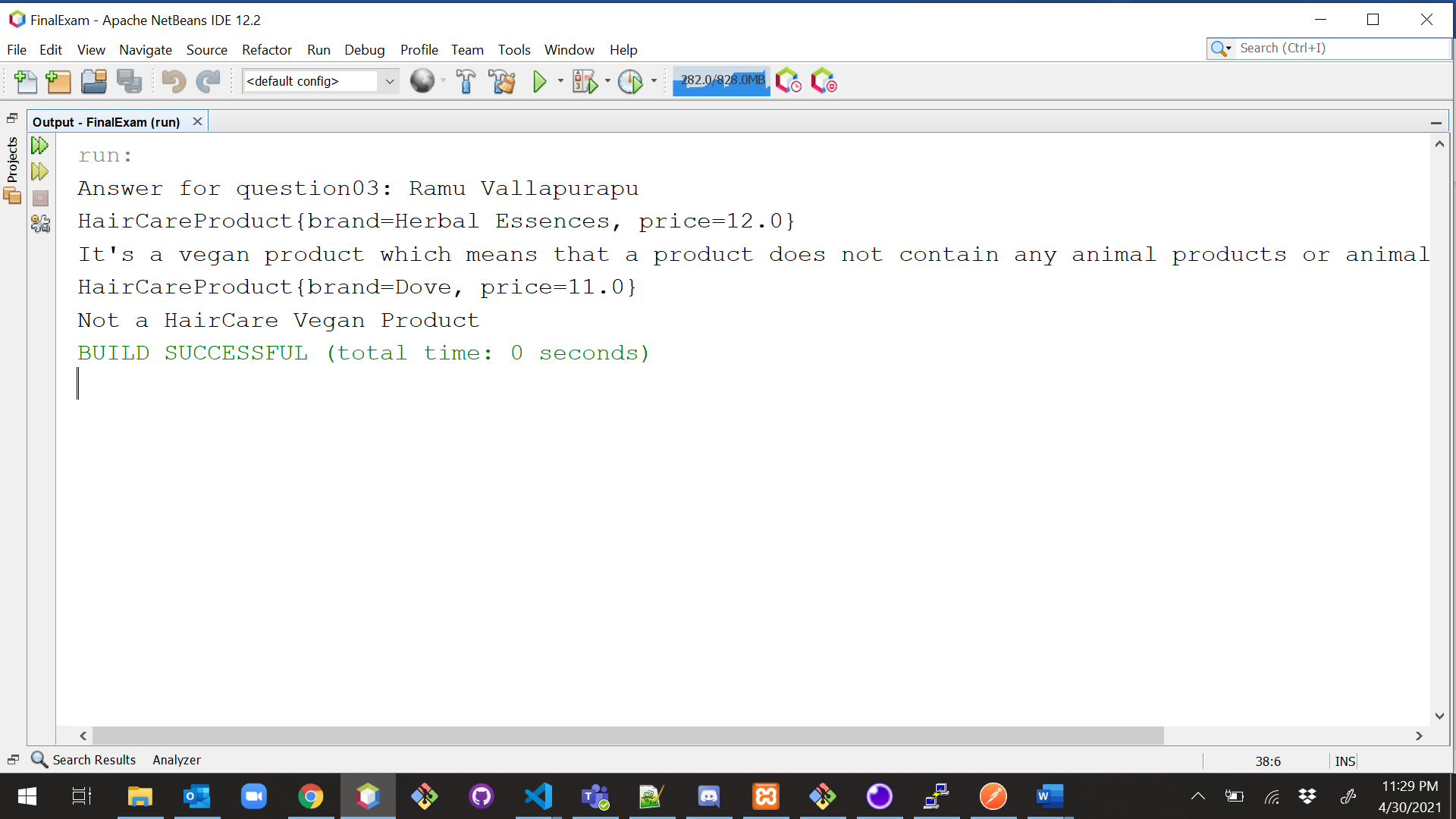
We just converted the double value 15.3 into an integer, which left only the whole number and the decimal places to be cut off. There is no rounding, it just ignores the numbers after the decimal point during casting. When you convert an int to a double, a decimal point will be added to match with the structure of a double as you see above, we have converted int value 10 to double the output is with a decimal point.

**Example2:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class HairCareProduct {  private String brand;  private double price;  public HairCareProduct(String brand, double price) {  this.brand = brand;  this.price = price;  }  public String getBrand() {  return brand;  }  public double getPrice() {  return price;  }  @Override  public String toString() {  return "HairCareProduct{" + "brand=" + brand + ", price=" + price + '}';  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class HairConditioner extends HairCareProduct {  public HairConditioner(String brand, double price) {  super(brand, price);  }  public String veganProduct() {  if (super.getBrand().equalsIgnoreCase("Herbal Essences")) {  return "It's a vegan product which means that a product does"  + " not contain any animal products or animal-derived ingredients.";  }  return "Not a HairCare Vegan Product ";//To change body of generated methods, choose Tools | Templates.  }  @Override  public double getPrice() {  return super.getPrice();  }  @Override  public String toString() {  return super.toString() + "\n" + veganProduct();  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question03;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class HairDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question03: Ramu Vallapurapu");  int price = 12;  double price2 = (double) price;  HairConditioner c = new HairConditioner("Herbal Essences", price2);  //upcasting  HairCareProduct h = c;  h = (HairCareProduct) c;  System.out.println(h.toString());  HairCareProduct h1 = new HairConditioner("Dove", 11);  if (h1 instanceof HairConditioner) {  //downcasting  ((HairConditioner) h1).getPrice();  ((HairConditioner) h1).veganProduct();  }  System.out.println(h1.toString());  }  } |



**Explanation:**

In the above example we have created object for Class HairConditioner HairCareProduct hcp = new HairConditioner(); We know that hcp variable refers to the instance of HairConditioner.

Here if we want to call hcp.veganProduct() since the method is not available in the HairCareProduct class, this would throw an error.

To overcome the above error we can downcast like

((HairConditioner) hcp).veganProduct();

Here adding (HairConditioner) before the hcp variable is down casting it and will allow us to access the methods specific to the HairConditioner class.

Usually we can use instanceof operator before downcasting to check if the object belongs to the specific type and n to run into any errors.

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?

b. Is fruit instanceof Orange?

c. Is fruit instanceof Apple?

d. Is fruit instanceof GoldenDelicious?

e. Is fruit instanceof McIntosh?

f. Is orange instanceof Orange?

g. Is orange instanceof Fruit?

h. Is orange instanceof Apple?

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

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

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

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

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

**Answers:**

**a. Is fruit instanceof Fruit?**

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

**b. Is fruit instanceof Orange?**

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

**c. Is fruit instanceof Apple?**

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

**d. Is fruit instanceof GoldenDelicious?**

Yes,fruit is an instance of GoldenDelicious

**e. Is fruit instanceof McIntosh?**

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

**f. Is orange instanceof Orange?**

Yes,orange is instance of Orange.

**g. Is orange instanceof Fruit?**

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

**h. Is orange instanceof Apple?**

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

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

Fruit can invoke method orange, but orange cannot as orange is not an instance of Apple whereas fruit is 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?**

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

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

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

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

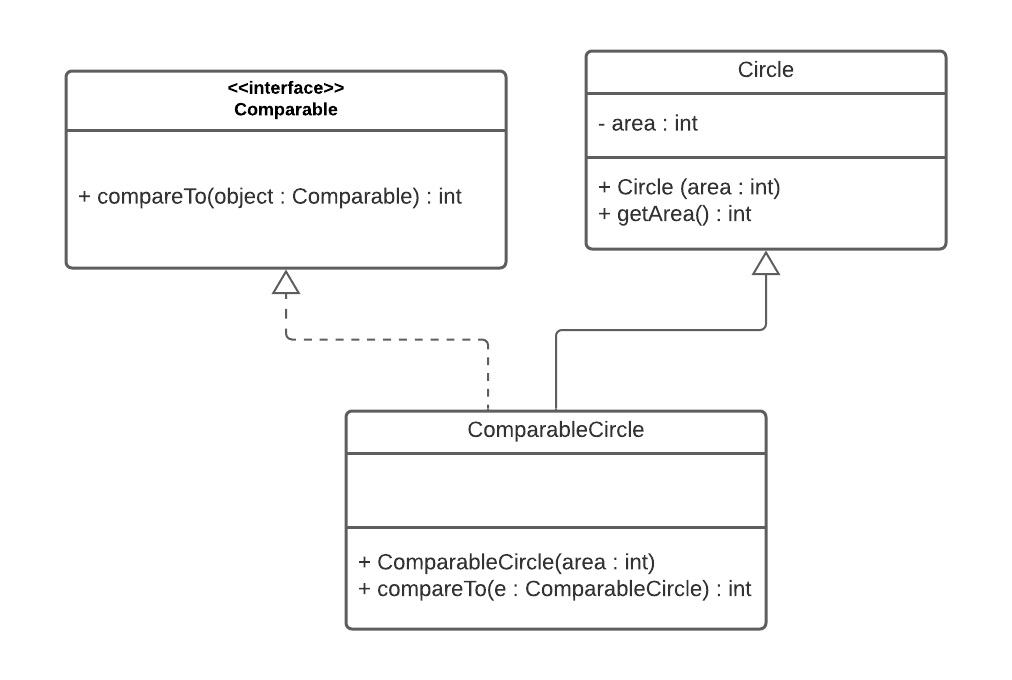
Illegal Statement,super class(Apple) instance cannot be assigned to sub-class reference(McIntosh)

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

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

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

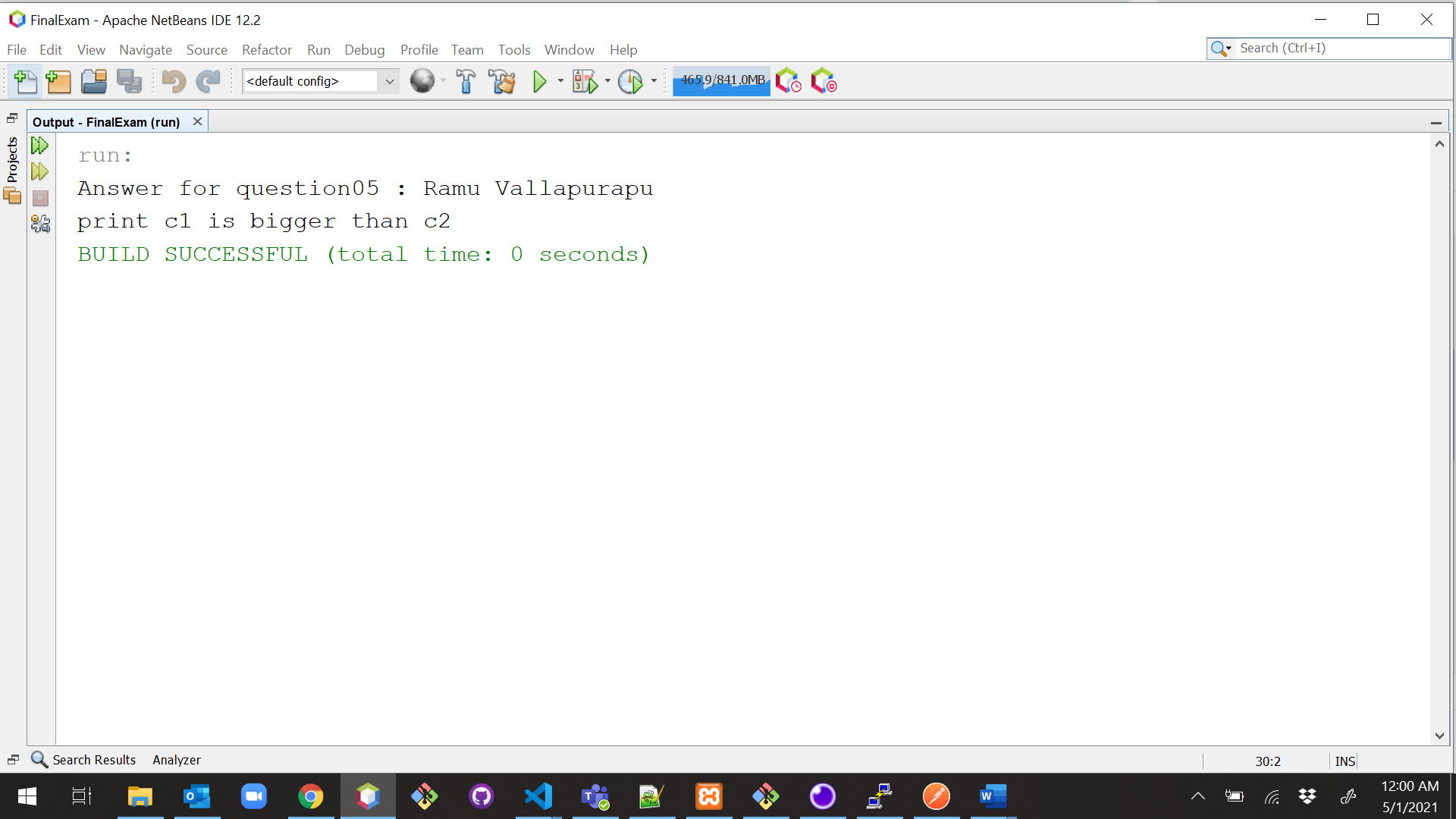
**UML:**



|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question05;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Circle {  private int area;  public Circle(int area) {  this.area = area;  }  public int getArea() {  return area;  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question05;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ComparableCircle extends Circle implements Comparable<ComparableCircle> {  public ComparableCircle(int area) {  super(area);  }  @Override  public int compareTo(ComparableCircle o) {  return Integer.compare(super.getArea(), o.getArea());  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question05;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class TestDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question05 : Ramu Vallapurapu");  ComparableCircle c1 = new ComparableCircle(30);  ComparableCircle c2 = new ComparableCircle(25);  int result = c1.compareTo(c2);  if (result == 1) {  System.out.println("print c1 is bigger than c2");  } else {  System.out.println("print c2 is bigger than c1");  }  }  } |



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

The general terminology of an exception is that it is an unexpected/unwanted occurrence of an event that disturbs the normal flow of a program/process. All exception classes are subtypes of the java.lang.Exception class. Exception is a subclass of throwable class. There is another subclass in throwable class, those are called errors. Errors cannot be handled by the user whereas exceptions can be handled. Exceptions can be handled by using try-catch block or throws keyword.

Advantages of Exception Handling in Java

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Meaningful Error Reporting



**Types of Exceptions:**

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

**Checked exceptions** are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](https://beginnersbook.com/2013/04/java-throws/), otherwise the program will give a compilation error.

Examples: ClassNotFoundException, IOException, SQLException etc

1. **ClassNotFoundException** is thrown when JVM tries to load a particular class but could not find the path. Class.forName()is used to load the class by passing String name of a class and it’s not found on the classpath.
2. **IOException** is thrown when the programmers use in the code to throw a failure in Input & Output operations.
3. **FileNotFound** exception occurs when the application failed attempt to open the file denoted by a specified pathname or when the application wants to write but the file is read-only.

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

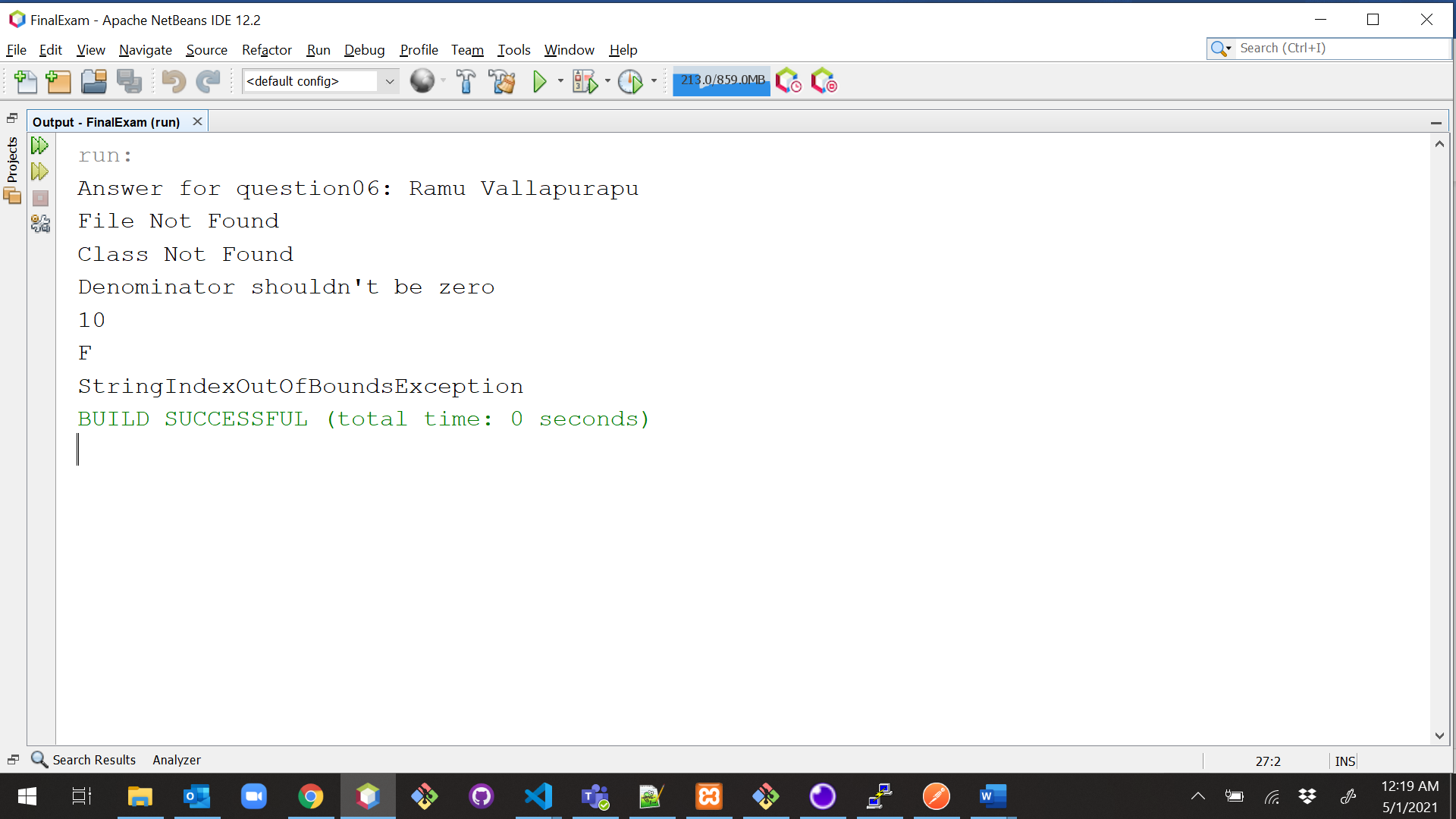
Examples Arithmetic Exception, ArrayStoreException etc

1. ArithmeticException: Arithmetic error, such as divide-by-zero.
2. ArrayIndexOutOfBoundsException: Array index is out-of-bounds.
3. ArrayStoreException: Assignment to an array element of an incompatible type.
4. NullPointerException occurs in code when we try to access/ modify an object which has not been initialized yet.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question06;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class CheckedExceptions {  public void fileNotFoundException() {  File file = new File("nofile.txt");  try {  FileInputStream sc1 = new FileInputStream(file);  } catch (FileNotFoundException e) {  //Any file not found exceptions by the Scanner sc1 will be caught here  System.out.println("File Not Found");  }  }  public void classNotException() {  try {  Class.forName("ques06.UncheckedException");  } catch (ClassNotFoundException ex) {  System.out.println("Class Not Found");  }  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question06;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class UncheckedException {  int a = 0;  int b = 0;  String s = null;  String s1 = null;  public UncheckedException(int a, int b, String s) {  this.a = a;  this.b = b;  this.s = s;  }  public void arithemeticException() {  try {  int output = a / b;  System.out.println("Result: " + output);  } catch (ArithmeticException e) {  System.out.println("Denominator shouldn't be zero");  }  }  public void stringIndexOutOfBoundsException() {  try {  //String str="beginnersbook";  System.out.println(s.length());  System.out.println(s.charAt(0));  System.out.println(s.charAt(23));  } catch (StringIndexOutOfBoundsException e) {  System.out.println("StringIndexOutOfBoundsException");  }  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question06;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ExceptionDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // Checked Exceptions  System.out.println("Answer for question06: Ramu Vallapurapu");  CheckedExceptions e = new CheckedExceptions();  e.fileNotFoundException();  e.classNotException();  UncheckedException e2 = new UncheckedException(30, 0, "Final Exam");  e2.arithemeticException();  e2.stringIndexOutOfBoundsException();  }  } |



**Explanation**:

In the CheckedExceptions class within the method fileNotFoundException(), we notice that we use the File while creating the object for FileInputStream, this requires the file to be present else it would throw FileNotFoundException. Any subsequent use of the FileInputStream object would depend on its creation in the first place hence it is required to either handle that using a catch or let the user know it by declaring using throws in the method declaration.

In the CheckedExceptions class within the method classNotException(), we notice that we use the Class.forName , this again throws the ClassNotFoundException as the system does not know how to proceed further hence we catch it after the try block.

All of the Checked exceptions will throw compiler errors if not handled by either try catch block or at the method declaration using the throws keyword

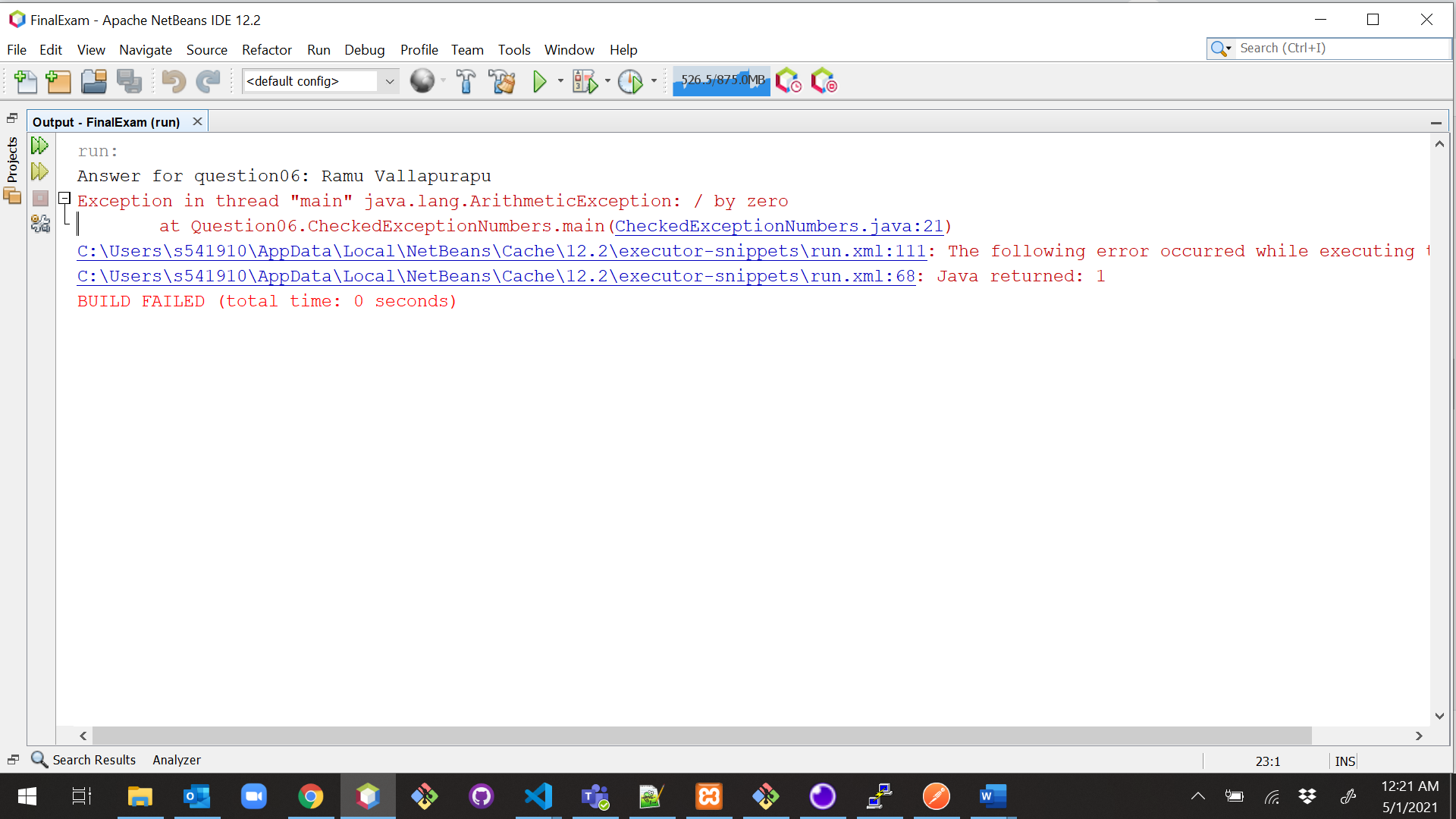
In the UncheckedException class within the method arithemeticException(), we notice that we use a/b where bis the denominator, if for suppose the value of b is 0 we run into the ArithmeticException. in some cases the value of b or the division itself is only determined at the runtime so we run into the Unchecked Exception.

In the UncheckedException class within the method stringIndexOutOfBoundsException(), we can have string of varied length and if trying to retrieve value at a specific index which is out of bounds it’ll run into StringIndexOutOfBoundsException. Here the string or the index can be provided by the user hence this is also an Unchecked Exception.

Unchecked Exceptions will never throw complier errors but can fail at runtime hence whenever possible if there is a possibility of unchecked exception user need to handle them either by try catch block or at the method declaration using the throws keyword.

**Example02:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question06;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class CheckedExceptionNumbers {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question06: Ramu Vallapurapu");  int numerator = 1;  int denominator = 0;  int result = numerator / denominator;  }  } |

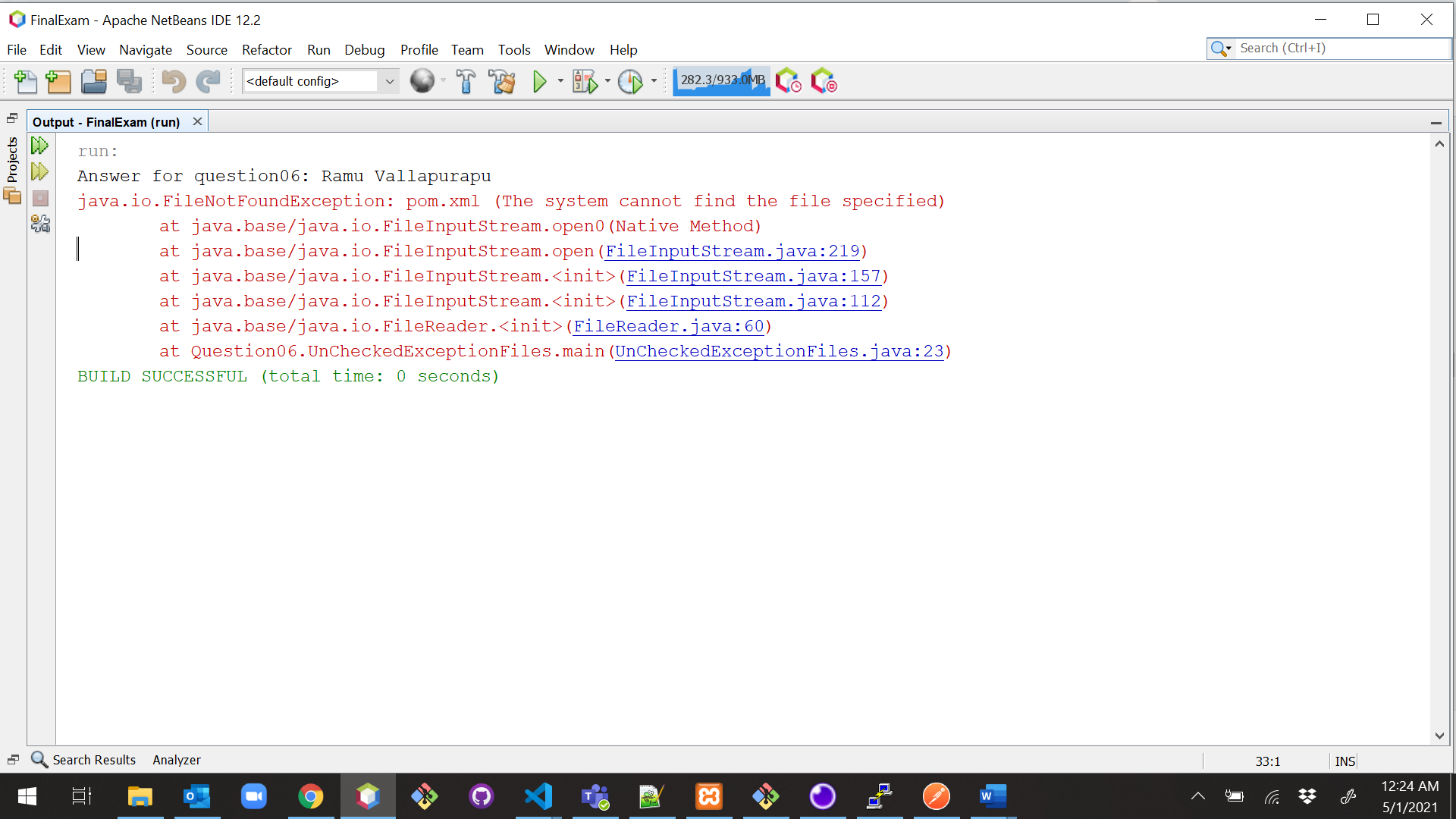


**Explanation:**

we do not have to declare unchecked exceptions in a method with the throws keyword. And although the above code does not have any errors during compile-time, it will throw Arithmetic Exception at runtime.

**Example03:**

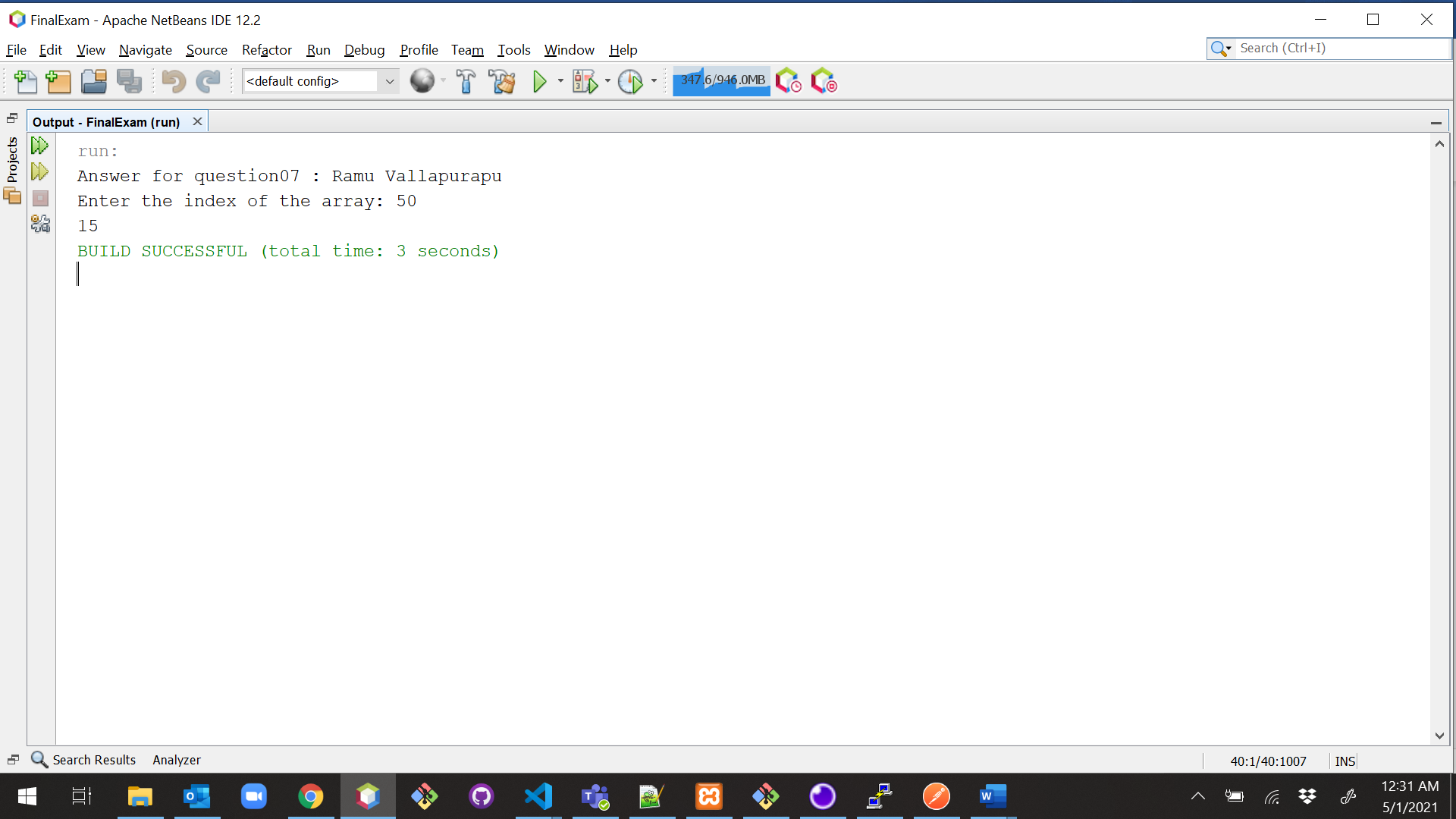
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package Question06;  import java.io.FileReader;  import java.io.IOException;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class UnCheckedExceptionFiles {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question06: Ramu Vallapurapu");  try {  FileReader file = new FileReader("pom.xml");  file = null;  file.read();  } catch (IOException e) {  //Alternate logic  e.printStackTrace();  }  }  } |

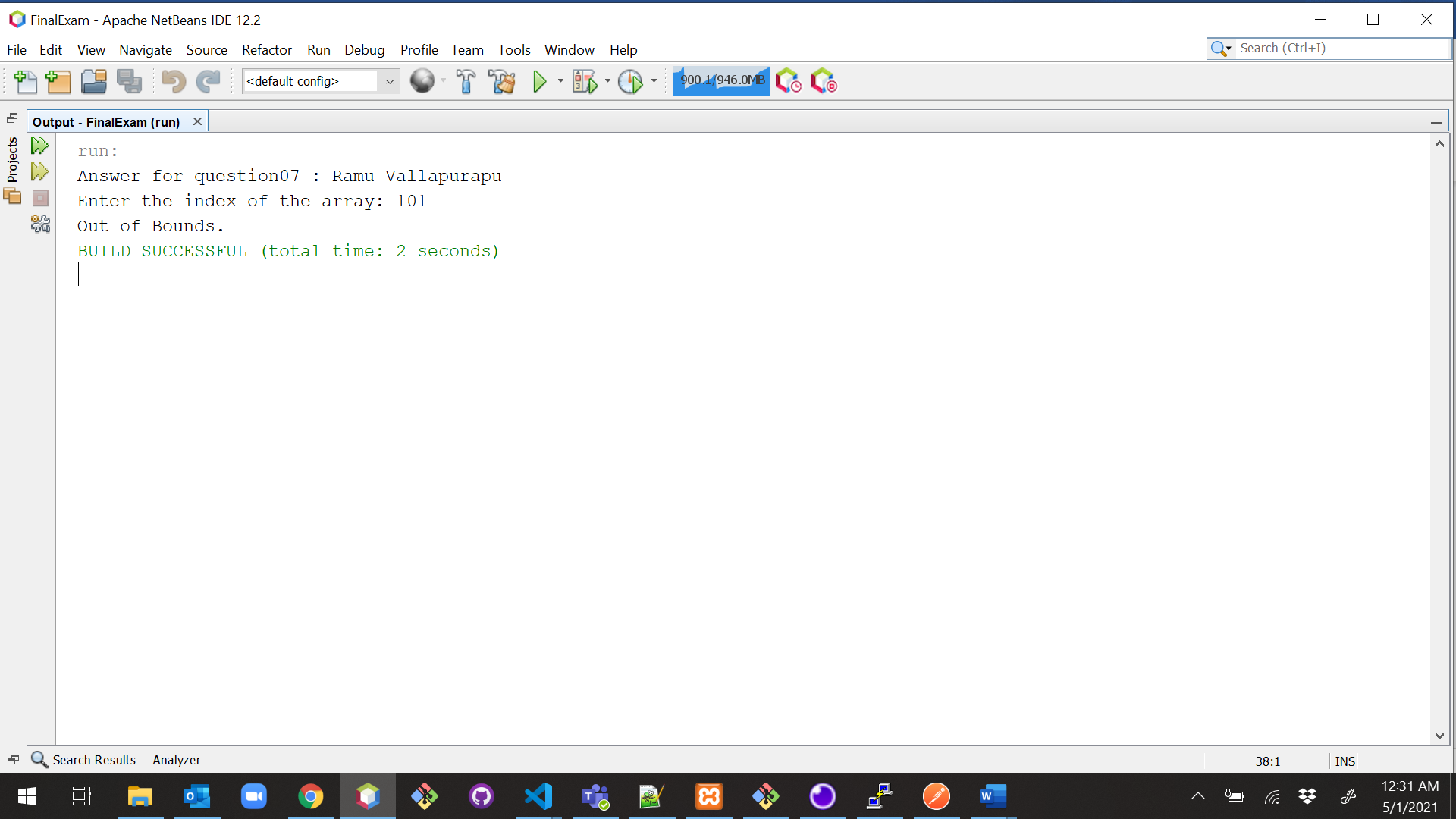


1. (10-Points) Write a program that meets the following requirements:

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

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question07;  import java.util.Scanner;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class randomarraynumber {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question07 : Ramu Vallapurapu");  int[] randArray = new int[100];  for (int i = 0; i < 100; i++) {  randArray[i] = (int) (Math.random() \* 100);  }  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the index of the array: ");  int temp = scanner.nextInt();  // try catch block if user specifies wrong value  try {  System.out.println(randArray[temp - 1]);  } catch (Exception e) {  System.out.println("Out of Bounds.");  }  }  } |





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

**Exception** usually means error, when we are writing code, some exceptions can be handled and some cannot be, but all these exceptions provide the user with a meaningful information to either point that there is something wrong that is happening or an unexpected flow the user is running into. An exception can occur for many different reasons it could be a code issue or a JVM issue or a connection issue.

* The purpose of declaring exceptions is to provide the user using that piece of code, to let them know of the possible exceptions that a user can run into.
* In Java one of the main concepts is code reusability, so for the code to be reusable the code must mention all the exceptions that could happen due to either bad user input or any other issue.
* The throws keyword is used to declare the exception that can occur during the program flow. It gives information about the exception to the programmer.
* It is always better to handle the exceptions so that there is no abrupt break in the flow.
* All exceptions should be Throwable.

**Declaration:**

Within code generally there are 3 places we use exceptions

* When creating a custom exception. You can have a checked exception by extending the Exception class. If you want to have an unchecked exception, we can extend Runtime Exception class.
* When catching the exception after try block in try catch. While throwing the exception you always throw a single exception in a single java statement but when catching the exception, you can catch multiple exceptions or also use the parent/ super class which has multiple exception subclasses.
* In the method header if the method throws an exception. Within the method for custom exceptions or user thrown exceptions you would use the throw keyword. Also, here you can declare multiple exceptions in the method header using the throws keyword.

**Yes**, we can declare multiple exceptions in method header but if there is an exception has occurred it will stop executing and will not go for remaining exceptions.

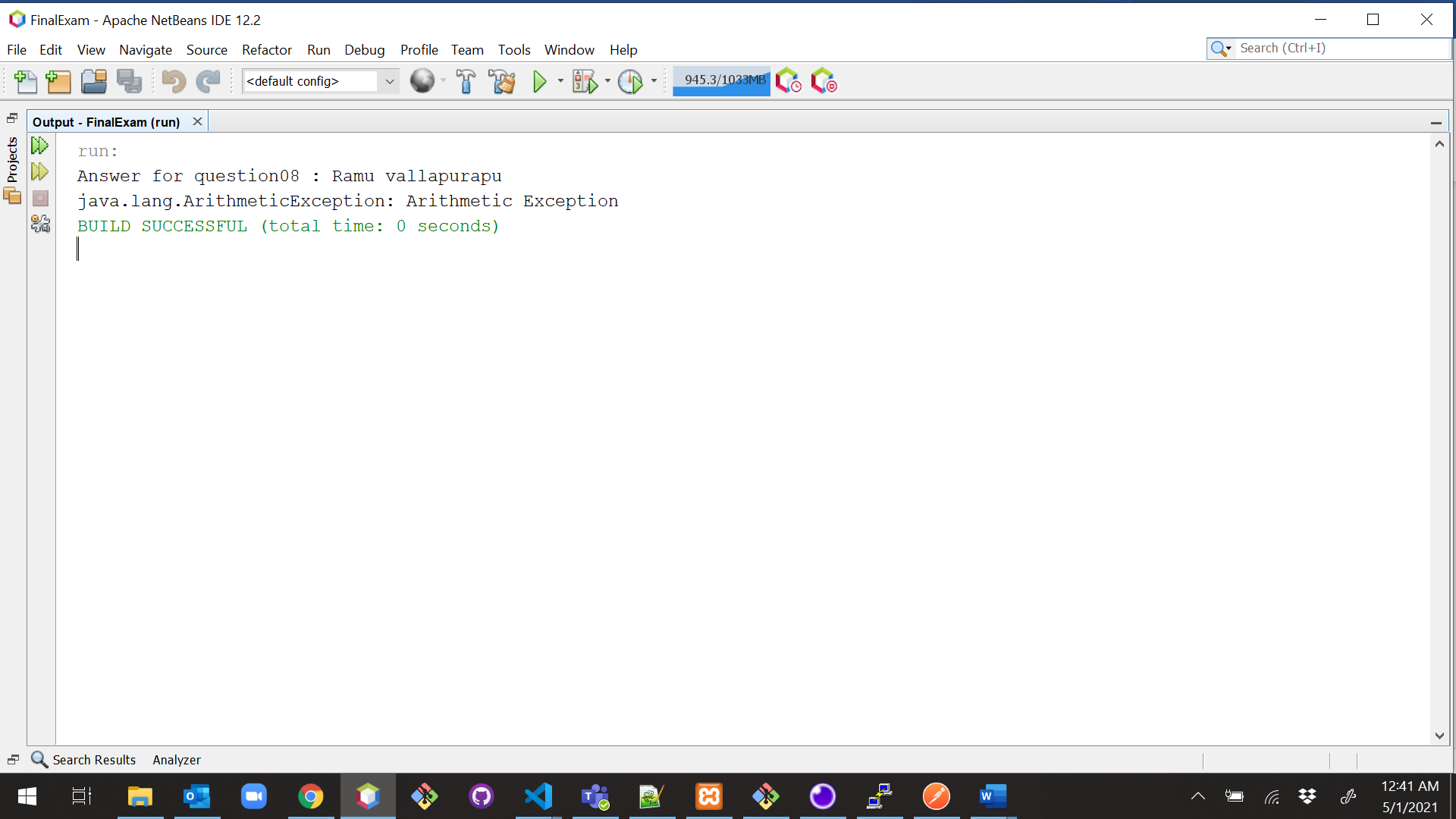
**Example Explanation:**

In the below example if you notice we only throw one exception at a time. Once the exceptions are thrown it stops the flow of execution, hence in our example we would either get Arithmetic Exception or NumberFormatException depending on the value of b.

However, during the method declaration you’d still need to mention all the exceptions that can be thrown from within that method hence we have.

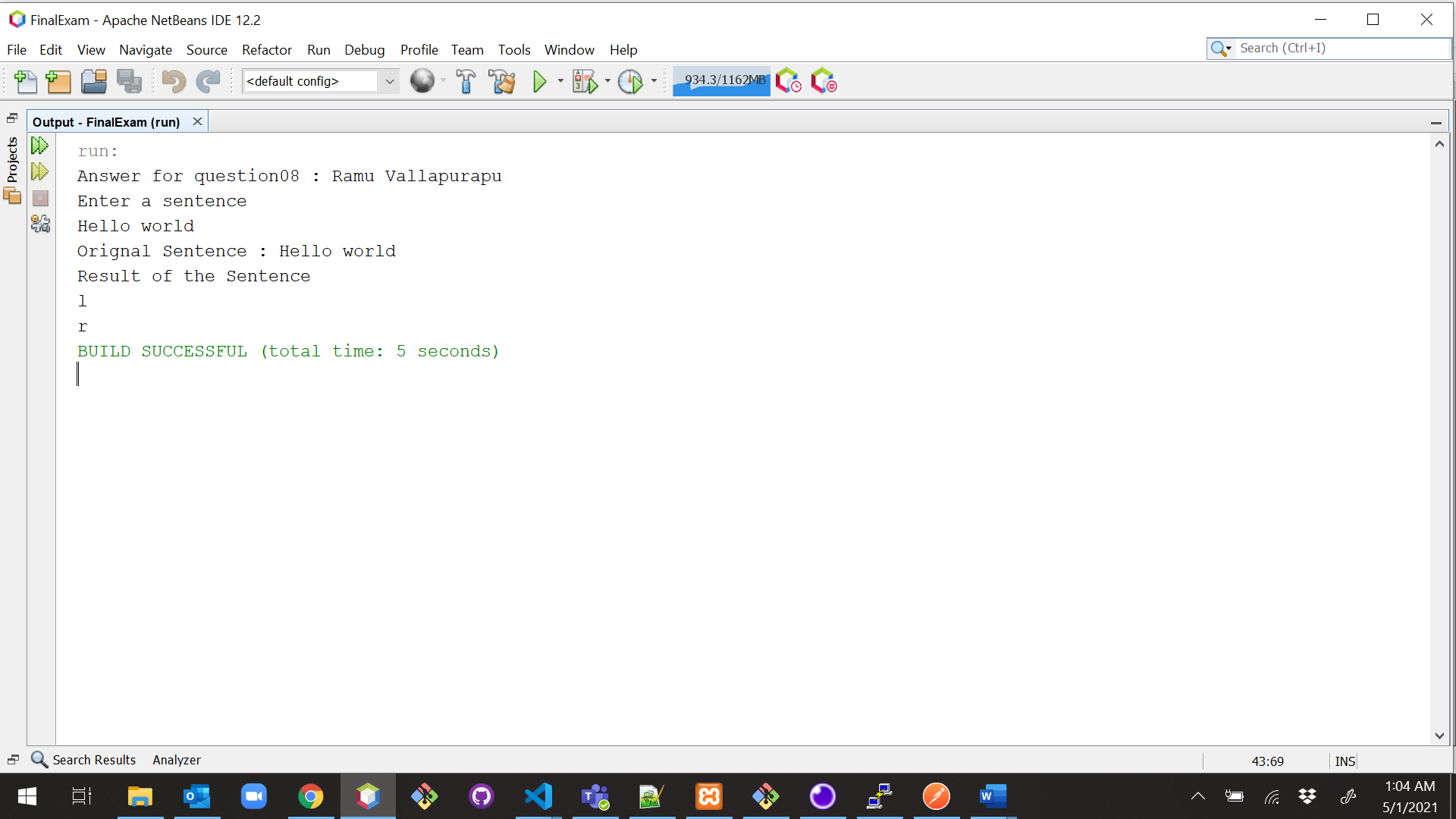
myMethod() throws ArithmeticException, NumberFormatException

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question08;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class MultiHeader {  static void myMethod() throws ArithmeticException, NumberFormatException {  int a = 9;  int b = 0;  if (b == 0) {  throw new ArithmeticException("Arithmetic Exception");  }  a = Integer.parseInt(null);  throw new NumberFormatException("NumberFormat Exception");  }  public static void main(String args[]) {  System.out.println("Answer for question08 : Ramu vallapurapu");  try {  myMethod();  } catch (Exception ex) {  System.out.println(ex);  }  }  } |



**Example02:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question08;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ExceptionExample {  static void MethodWithMultipleExc() throws IOException, ArrayIndexOutOfBoundsException {  BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  System.out.println("Enter a sentence");  String sentence = reader.readLine();  System.out.println("Orignal Sentence : " + sentence);  String arr[] = sentence.split(" ");  System.out.println("Result of the Sentence");  for (int i = 0; i < arr.length; i++) {  if (arr[i].length() > 2) {  System.out.print(arr[i].charAt(2));  System.out.println();  }  }  }  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  try {  MethodWithMultipleExc();  } catch (IOException ex) {  ex.printStackTrace();  } catch (ArrayIndexOutOfBoundsException ex) {  ex.printStackTrace();  }  }  } |



**Explanation**:

Here we are throwing two exceptions IOException, ArrayIndexOutOfBoundsException in the same method header. In this program we are trying to give a sentence and it gets stored in array after splitting.and it gives arr[2} i.e third letter of rvery word in the statement.it catches IOException, ArrayIndexOutOfBoundsException.

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.

Both throw and throws are the concepts of exception handing in which throw is used to explicitly throw an exception from a method or any block of code while throws are used in the signature of the method to indicate that this method might throw one of the listed type exceptions

Throw keyword is used to throw an exception, either a custom one or inbuilt java defined exception, we throw exceptions to let the user know that the current flow of code needs to be halted or cannot proceed further. While throwing the exception you always throw a single exception in a single java statement

We use **Throws** keyword in the method header if the method throws an exception or multiple exceptions. Here these exceptions could be user thrown exceptions or because of the code like in case of creating a File, by default the method using the file needs to throw the FileNotFoundException.

**No, we cannot throw multiple exceptions in a single throw statement** as once the exceptions are thrown it stops the flow of execution and there is no point in going to next exception. The whole point of exception is to let the user know of the error and narrow down the cause of the error. If multiple exceptions can be thrown from single place it would lose the meaning and it would be confusing. Instead of throwing multiple exceptions the use can throw Exception which is like the root class for all the exceptions.

Below are some of the sample codes where I was able to use the throw, throws keywords and display the use of exceptions.

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Checked {  public void fileNotFound() throws FileNotFoundException {  File file = new File("nofile.txt");  FileInputStream sc1 = new FileInputStream(file);  System.out.println("File Not Found");  }  public void classNotException() throws ClassNotFoundException {  Class.forName("ques09.Unchecked");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Unchecked {  int a = 0;  int b = 0;  String s = null;  public Unchecked(int a, int b, String s) {  this.a = a;  this.b = b;  this.s = s;  }  public void arithmetic() throws ArithmeticException {  int output = a / b;  System.out.print("Result: " + output);  }  public void stringIndexOutOfBound() throws StringIndexOutOfBoundsException {  //String str="beginnersbook";  System.out.println(s.length());  System.out.println(s.charAt(0));  System.out.println(s.charAt(23));  }  } |

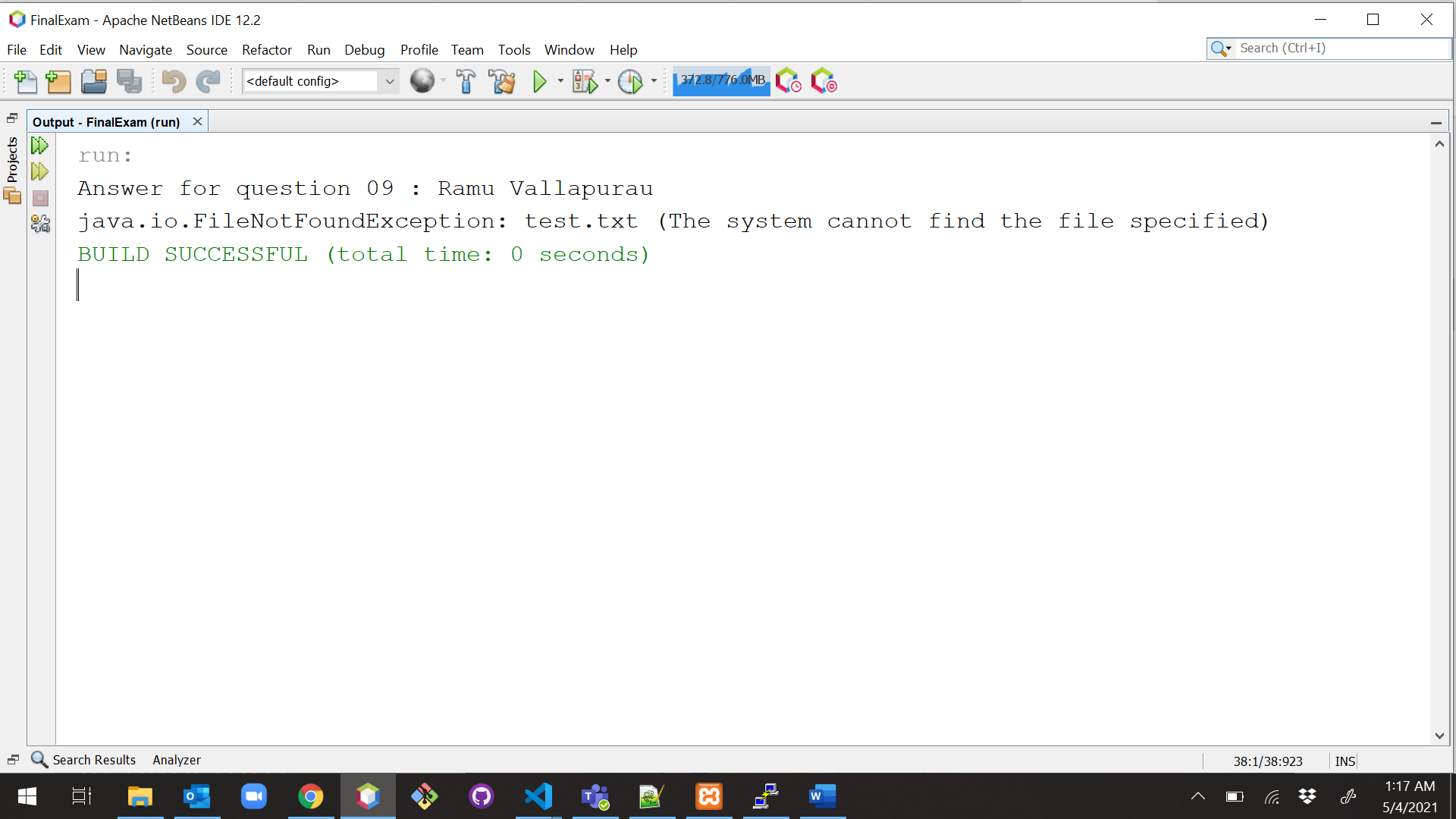
|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class GradeCheckWithThrow {  static void checkGrade(String grade) {  if (!grade.equalsIgnoreCase("A") || !grade.equalsIgnoreCase("B")) {  throw new ArithmeticException("Retake Course");  } else {  System.out.println("Good Job!");  }  }  public static void main(String[] args) {  checkGrade("B");  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  import java.io.FileNotFoundException;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ExceptionDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) throws ArithmeticException, StringIndexOutOfBoundsException, FileNotFoundException, ClassNotFoundException {  System.out.println("Answer for question 09: Ramu Vallapurapu");  // TODO code application logic here  Checked e = new Checked();  e.fileNotFound();  e.classNotException();  Unchecked e2 = new Unchecked(30, 0, "Final Exam");  e2.arithmetic();  e2.stringIndexOutOfBound();  }  } |



**Example02:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  import java.io.File;  import java.io.FileInputStream;  import java.io.IOException;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ThrowsExample01 {  public static void findFile() throws IOException {  // code that may produce IOException  File newFile = new File("test.txt");  FileInputStream stream = new FileInputStream(newFile);  }  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  System.out.println("Answer for question 09 : Ramu Vallapurau");  try {  findFile();  } catch (IOException e) {  System.out.println(e);  }  }  } |

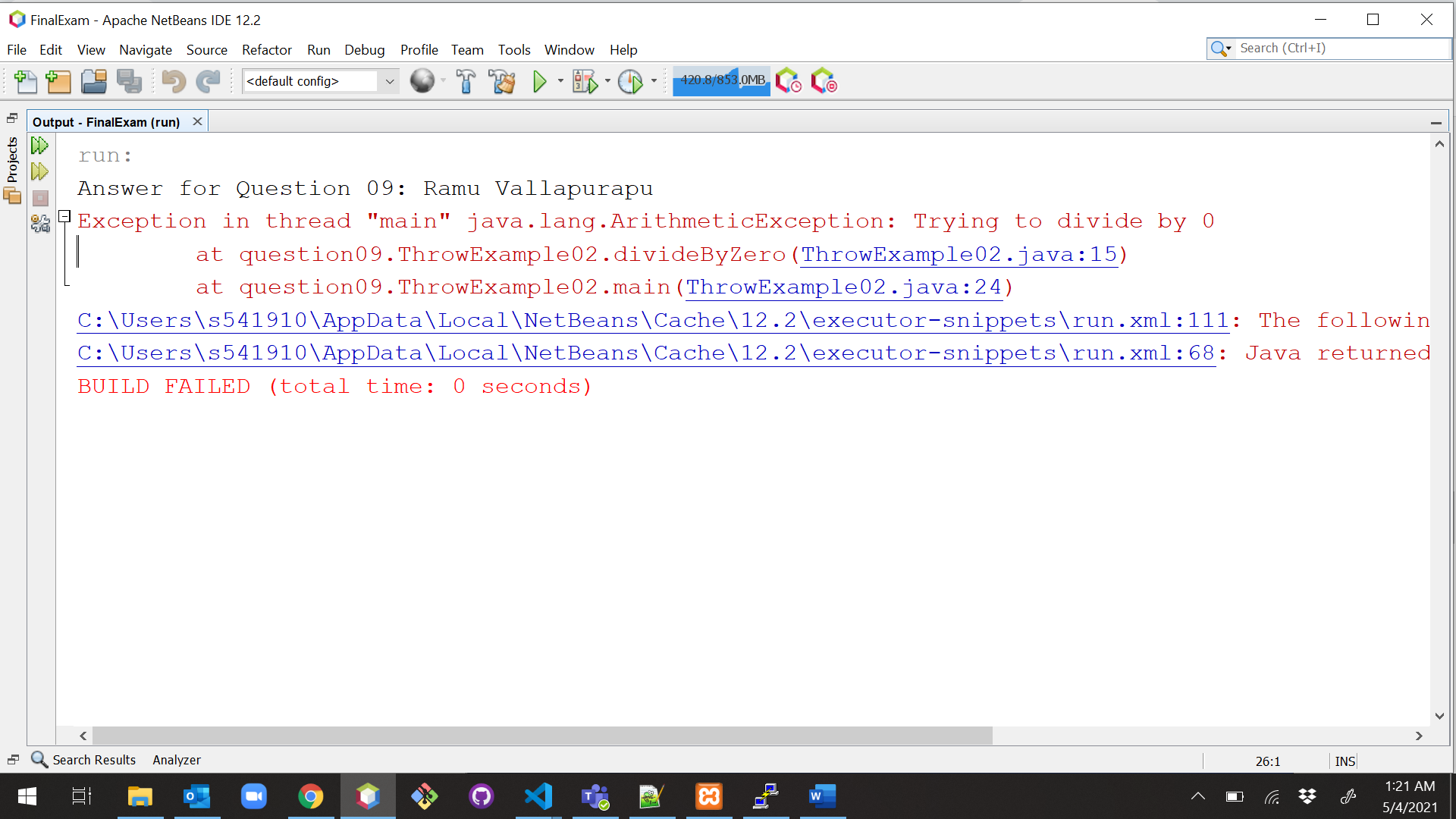


**Explanation:**

When we run this program, if the file test.txt does not exist, FileInputStream throws a FileNotFoundException which extends the IOException class. If a method does not handle exceptions, the type of exceptions that may occur within it must be specified in the throws clause so that methods further up in the call stack can handle them or specify them using throws keyword themselves. The findFile() method specifies that an IOException can be thrown. The main () method calls this method and handles the exception if it is thrown.

**Example03:**

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question09;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class ThrowExample02 {  public static void divideByZero() {  throw new ArithmeticException("Trying to divide by 0");  }  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for Question 09: Ramu Vallapurapu");  // TODO code application logic here  divideByZero();  }  } |



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.

**Recursion**: The technique in which the function calls itself is called recursion. Recursion helps to break complicated problems so that we have a simple solution.

There are two requirements for recursive function, they are.

1. **Stop Condition** - when some condition is satisfied, the function returns a value without a further recursive call.
2. **Recursive Call** – the function calls itself with an input which is a step closer to the stop condition.

**Infinite Recursion:** If a recursion never reaches a base case, it will go on making recursive

calls forever and the program will never terminate.

**Advantages:**

1. Reduces time complexity.
2. Adds more clarity and reduces the time for debugging the code.
3. Recursion is better at tree traversal.
4. More memory usage.

* Syntax

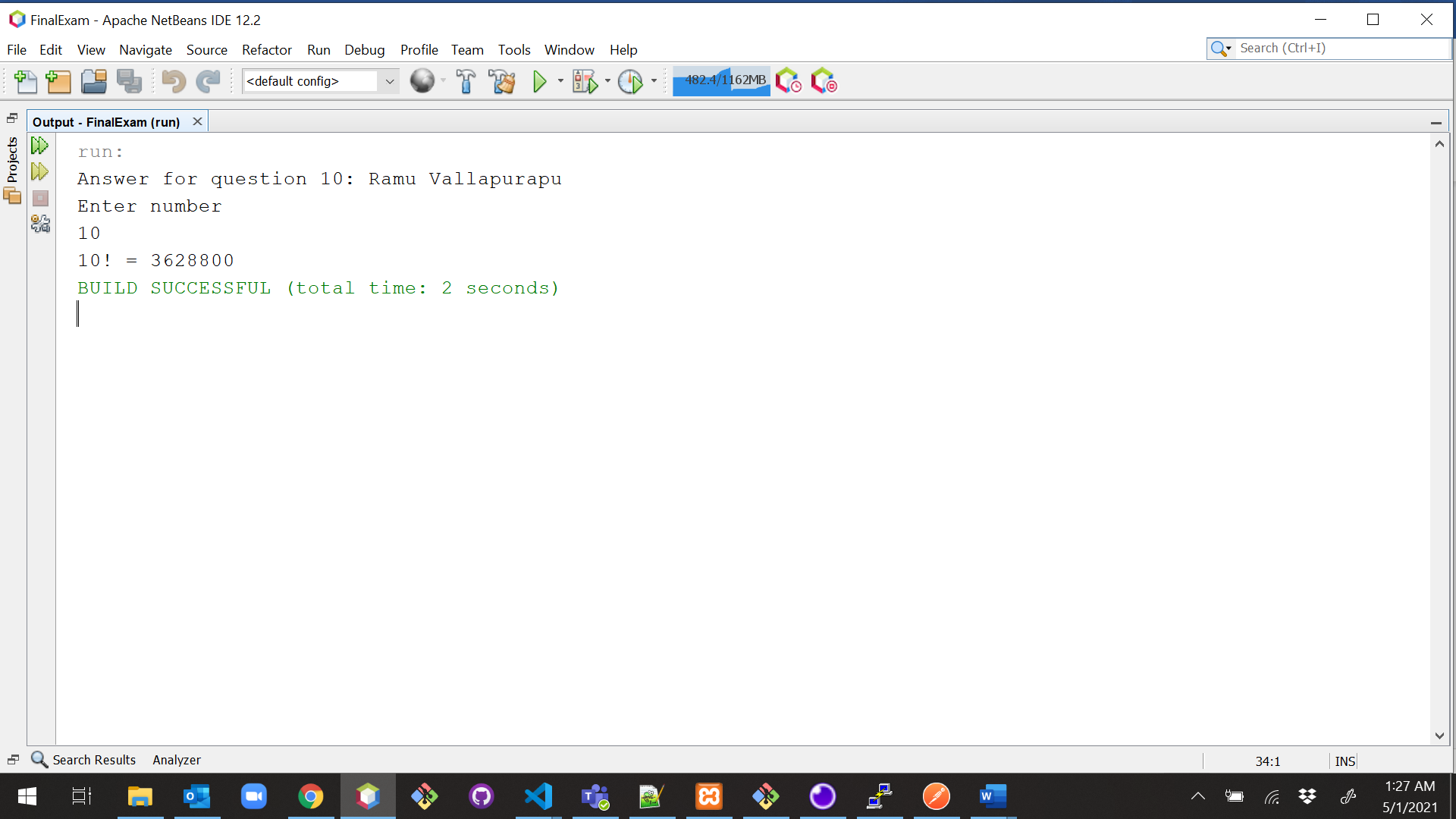
returntype method(){

//code to be executed

method();//calling same method

}

|  |
| --- |
| /\*  \* 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 Ramu Vallapurapu  \*/  public class FactorialRecursion {  public static void main(String[] args) {  System.out.println("Answer for question 10: Ramu Vallapurapu");  Scanner sc = new Scanner(System.in);  System.out.println("Enter number");  int num = sc.nextInt();  long factorial = calcFactorial(num);  System.out.println(num + "! = " + factorial);  sc.close();  }  public static long calcFactorial(int num) {  if (num >= 1) {  return num \* calcFactorial(num - 1);  } else {  return 1;  }  }  } |



**Explanation:**

Function calcFactorial(num) is called from the main() function. Here we can take any random number to calculate the factorial. Consider num 4, As 4 is greater than 0, 4 is multiplied to the result of calcFactorial() i.e 3 (num(4) -1)and it goes on. Asit is called from the same function, it is a recursive call. In each recursive call, the value of the argument number is decreased by 1for every step. Here in the example, the function is calling itself multiple numbers of times and comes out of the loop when the condition is completely satisfied i.e then the number becomes 0. There is no recursive call when the number becomes 0. Answer 4! = 4\*3\*2\*1 = 24, Displays 24.

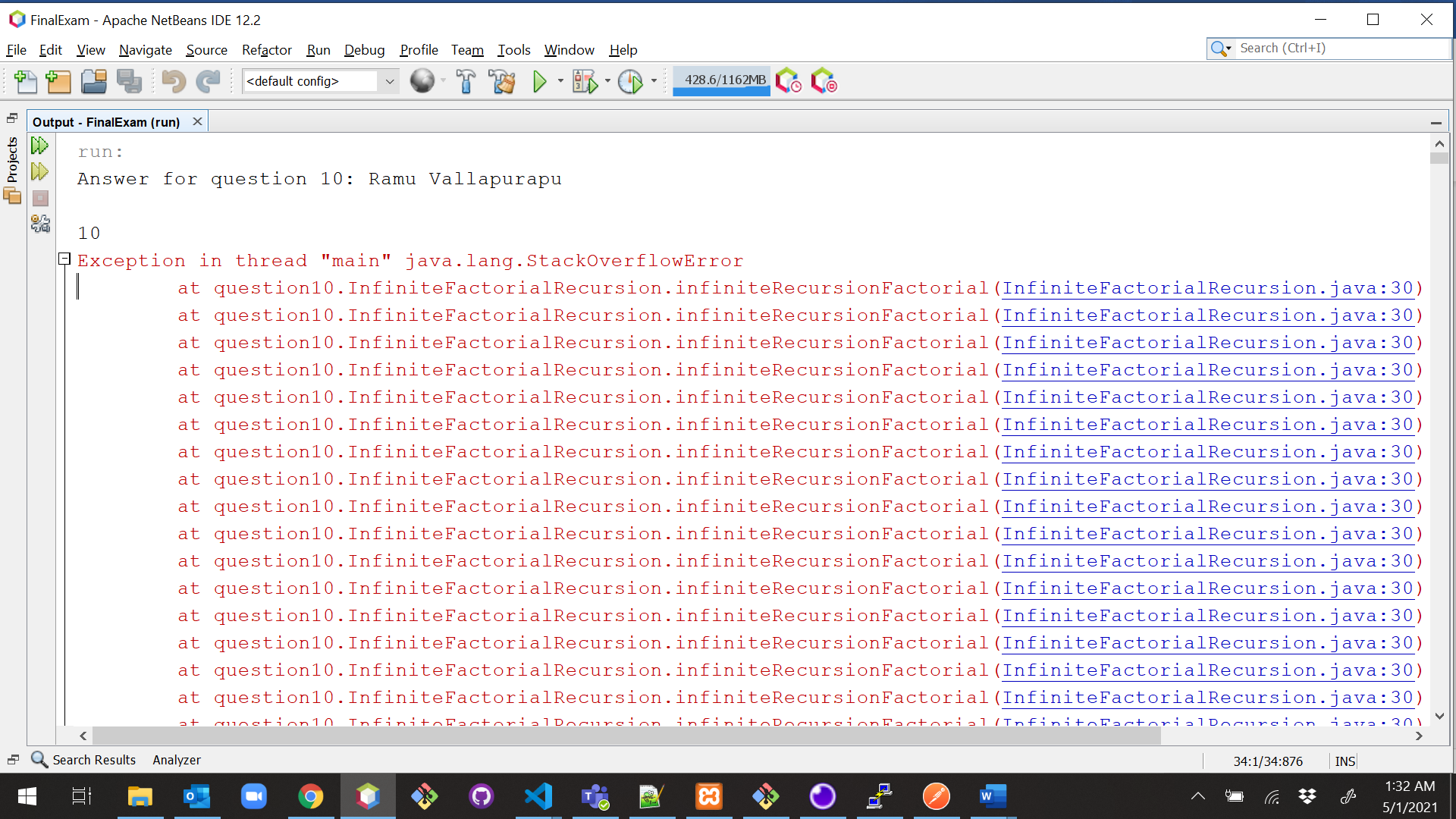
Here we observe, after each time the method is called the bigger problem is split into smaller problems.

**Infinite Recursion:**

* Any recursion which continues without is called infinite recursion.
* In this type of recursion, there is no base condition.
* It keeps on calling itself as there no condition which is getting satisfied.

To say more precisely, It occurs if the recursion step does not reduce the problem.

|  |
| --- |
| /\*  \* 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 Ramu Vallapurapu  \*/  public class InfiniteFactorialRecursion {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question 10: Ramu Vallapurapu");  Scanner sc = new Scanner(System.in);  int num = sc.nextInt();  long factorial = infiniteRecursionFactorial(num);  System.out.println(num + "! = " + factorial);  sc.close();  }  // example for infinite recursion  public static long infiniteRecursionFactorial(int num) {  return num \* infiniteRecursionFactorial(num - 1);  }  } |

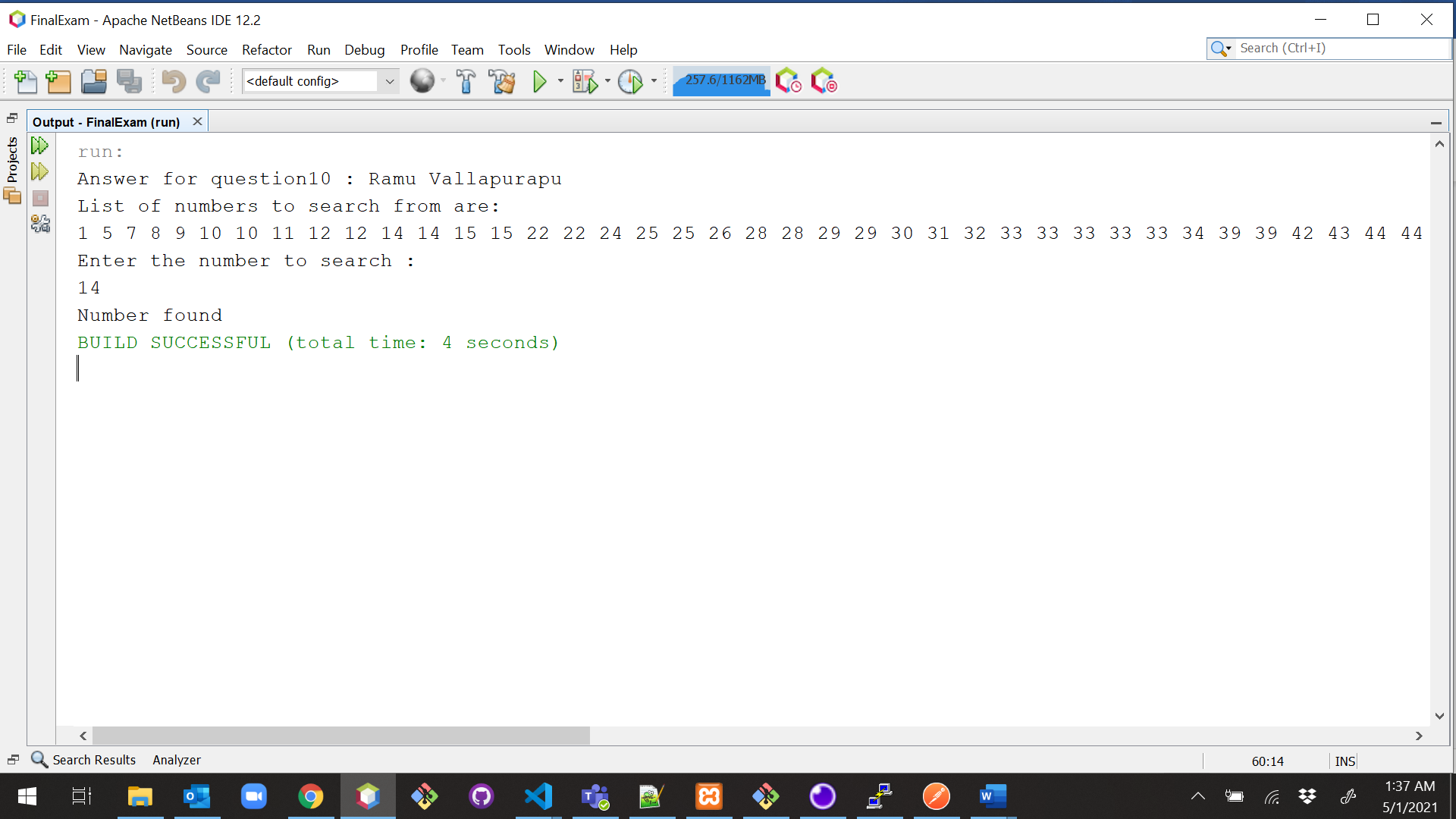


**Explanation:**

We have taken factorial example to demonstrate infinite recursion. Firstly, Function infiniteRecursionFactorial(num) is called from the main () function. We have taken the num to be 4, it starts following the method but as there is no condition to stop, it keeps on running. Here from the above example, we understand that the method is never-ending. This is happening because there is no base condition in the method (infiniteRecursionFactorial(int num)). In this case, the condition is not satisfied at any point. There are infinite times of recursive calls as there is no endpoint.

**BINARYSEARCH RECURSION**

|  |
| --- |
| /\*  \* 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.Collections;  import java.util.Scanner;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class BinarySearchRecursion {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question10 : Ramu Vallapurapu");  ArrayList<Integer> list = new ArrayList<>();  while (list.size() < 100) {  list.add((int) ((Math.random() \* 100) + 1));  }  // binary Search works only on sorted elements  Collections.sort(list);  System.out.println("List of numbers to search from are:");  for (int j = 0; j < list.size(); j++) {  System.out.print(list.get(j) + " ");  }  Scanner sc = new Scanner(System.in);  System.out.println("\nEnter the number to search :");  int x = sc.nextInt();  sc.close();  boolean result = binarySearch(list, 0, list.size() - 1, x);  if (result) {  System.out.println("Number found");  } else {  System.out.println("Number not found");  }  }  // method for binarySearch given the arraylist using recursion, left, right and  // search number  public static boolean binarySearch(ArrayList<Integer> list, int l, int r, int x) {  if (r >= l) {  int center = l + (r - l) / 2;  // If the number is present at the center  if (list.get(center) == x) {  return true;  }  // If number is smaller than center  if (x < list.get(center)) {  return binarySearch(list, l, center - 1, x);  } // If number is greater than center  else {  return binarySearch(list, center + 1, r, x);  }  }  // We reach here when the number is not present  return false;  }  } |

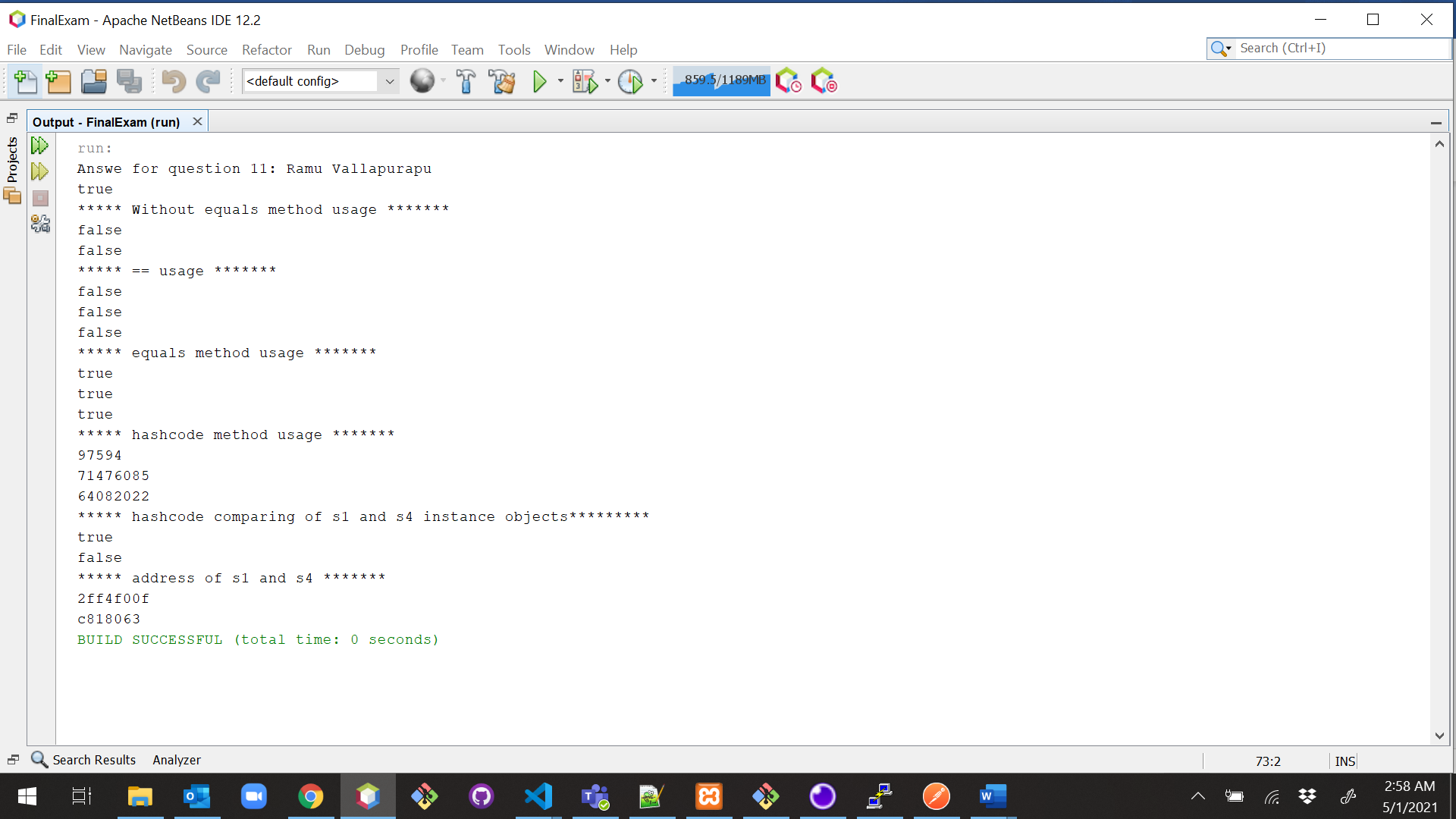


1. (10-Points) Write a java program that illustrates how equals() and hashCode() methods work? Explain your code in comments.

|  |
| --- |
| /\*  \* 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 Ramu Vallapurapu  \*/  public class StudentDriver {  public static void main(String[] args) {  System.out.println("Answe for question 11: Ramu Vallapurapu");  StudentWithoutEquals swoe = new StudentWithoutEquals(1, "WaterBottle");  StudentWithoutEquals swoe1 = new StudentWithoutEquals(1, "WaterBottle");  Student s1 = new Student(101, "Tom");  Student s2 = new Student(104, "Jerry");  Student s3 = new Student(105, "Ben10");  Student s4 = new Student(101, "Tom");  Student s5 = new Student(104, "Jerry");  Student s6 = new Student(105, "Ben10");  int a = 10;  int b = 10;  //we are just comapring integer values and if both values are same it will return true but it was differnt with objects  System.out.println(a == b);  System.out.println("\*\*\*\*\* Without equals method usage \*\*\*\*\*\*\*");  System.out.println(swoe == swoe1);  System.out.println(swoe.equals(swoe1));  System.out.println("\*\*\*\*\* == usage \*\*\*\*\*\*\*");  //here we are comapring different objects baed upon the reference location  // == for object generally compares there reference location.  // so if any two object variable refers to same location it return otherwise false.  System.out.println(s1 == s4);  System.out.println(s2 == s5);  System.out.println(s3 == s6);  System.out.println("\*\*\*\*\* equals method usage \*\*\*\*\*\*\*");  //here compare the different Student class objects based upon their instance values values.  //The output may be true false based upon the instance varibale values.  //equals method all the instance variable of one object with corresponding instance of another class and if both are equals returns true  System.out.println(s1.equals(s4));  System.out.println(s2.equals(s5));  System.out.println(s3.equals(s6));  System.out.println("\*\*\*\*\* hashcode method usage \*\*\*\*\*\*\*");  //Here are the hashcode values of all the variable  //we should get s1,s2,s3 object hash code values as same because hash code of a object on instance variable value where s1,s2,s3 as same instance variable values  System.out.println(s1.hashCode());  System.out.println(s2.hashCode());  System.out.println(s3.hashCode());  System.out.println("\*\*\*\*\* hashcode comparing of s1 and s4 instance objects\*\*\*\*\*\*\*\*\*");  //This should return true because s1,s4 has same instance variable values and hash code is generated based upon the them. so object hashcode should be same  System.out.println(s1.hashCode() == s4.hashCode());  //This should return false because s1,s3 has same instance variable values and hash code is generated based upon the them. so object hashcode should be not same  System.out.println(s1.hashCode() == s3.hashCode());  System.out.println("\*\*\*\*\* address of s1 and s4 \*\*\*\*\*\*\*");  //we are also checked the address of s1,s4 to check whether there refer to same location  System.out.println(Integer.toHexString(System.identityHashCode(s1)));  System.out.println(Integer.toHexString(System.identityHashCode(s4)));  }  } |

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package question11;  import java.util.Objects;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class Student {  private int id;  private String name;  public Student(int id, String name) {  this.id = id;  this.name = name;  }  public int getId() {  return id;  }  public String getName() {  return name;  }  @Override  public String toString() {  return "Student{" + "id=" + id + ", name=" + name + '}';  }  @Override  public int hashCode() {  /\*\*  \* this method return hascode value of a class based upon the instance  \* variables value. this method is also used to compare different object  \* of same class type based upon their instance variable value the  \* hascode value of two class will only be equal if all instance  \* variable values of one class is equal to all th instance variable  \* values of comparing object.  \*/  int hash = 7;  //Because model attribyte is string it generates a hashcode to it and add to hash value.  hash = 37 \* hash + this.id;  hash = 37 \* hash + Objects.hashCode(this.name);  //after hash value of complete class.  return hash;  }  @Override  /\*\*  \* Here I have declared equals method which is used to compares parameter  \* passed object with calling object and compares based upon all the  \* instance variable s1,s4 both the classes. since Object obj is super class  \* for all the classes we can declare it in parameter where passed instance  \* implicity converted Object type This method return boolean value true if  \* every instance attribute value are equal in both the instances.  \*/  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 Student by comparing the  \* classes of the passed argument and this object.  \*/  if (obj == null) {  return false;  }  //comparing class cast if both the class class cast is different there we can also say both the classes are different so it return false  if (getClass() != obj.getClass()) {  return false;  }  //converting obj variable to Student class which is example os down casting.  final Student other = (Student) obj;  //comparing first instance attribute id of both the objects if not equal it directly return false.  if (this.id != other.id) {  return false;  }  //comparing second instance attribute name of both the objects if not equal it directly return false.  if (!Objects.equals(this.name, other.name)) {  return false;  }  //this stage reacher only after all the instance variable are equals .so it returns true .  return true;  }  } |

|  |
| --- |
| /\*  \* 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 Ramu Vallapurapu  \*/  public class StudentWithoutEquals {  private int id;  private String name;  public StudentWithoutEquals(int id, String name) {  this.id = id;  this.name = name;  }  public int getId() {  return id;  }  public String getName() {  return name;  }  @Override  public String toString() {  return "StudentWithoutEquals{" + "id=" + id + ", name=" + name + '}';  }  } |



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

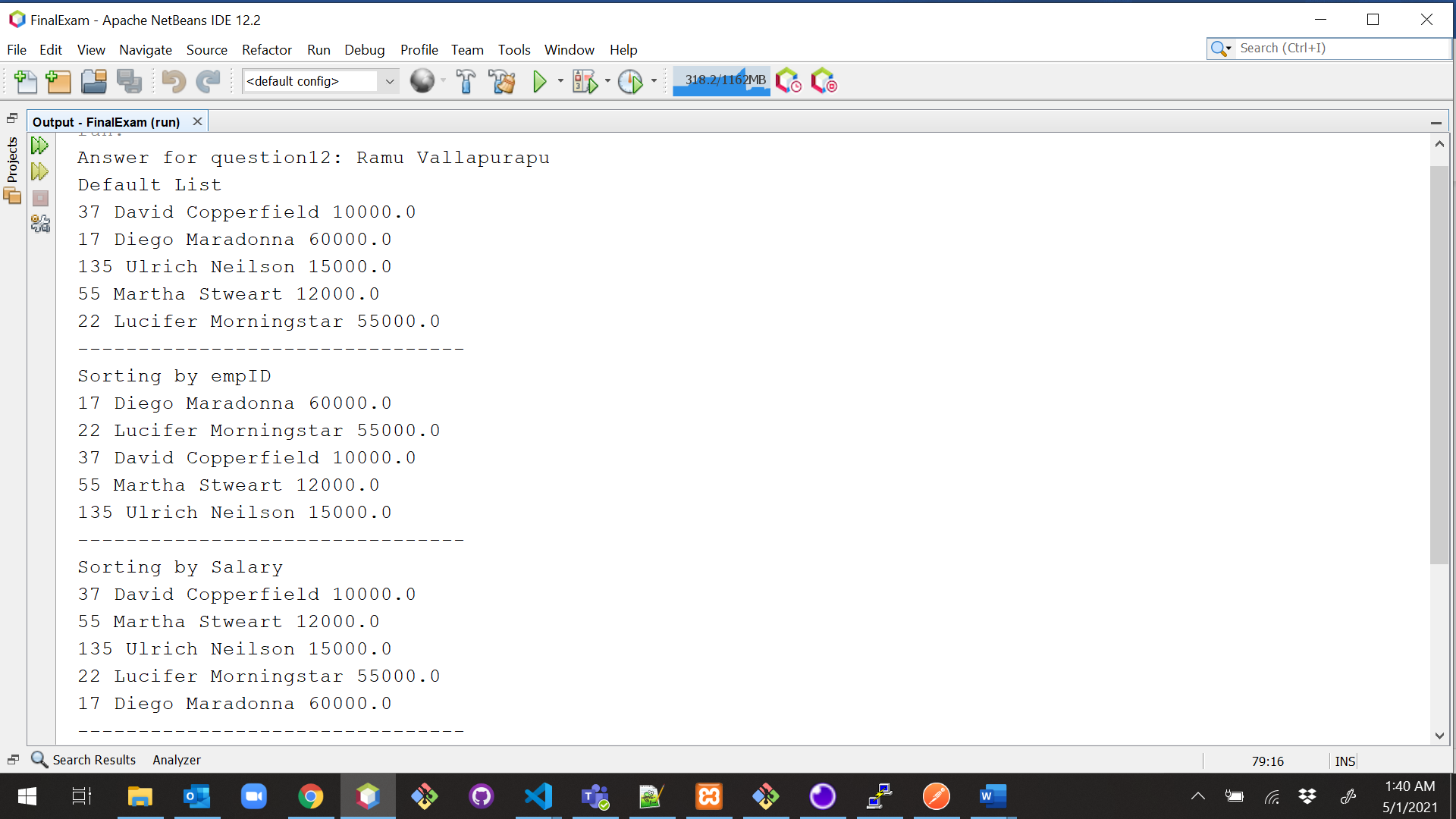
* Data fields named empId, empName and empSalary
* A constructor with parameters, listed in the same order as above.
* Create getter methods for all the parameters.
* A toString method that prints the empId, empName and empSalary. There should be one space between each value output.
* Because Employee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

1. **EmployeeDriver Class**

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

|  |
| --- |
| /\*  \* 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 Ramu Vallapurapu  \*/  public class Employee implements Comparable<Employee> {  public int empId;  public String empName;  public 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;  }  public void setEmpId(int empId) {  this.empId = empId;  }  public void setEmpName(String empName) {  this.empName = empName;  }  public void setEmpSalary(double empSalary) {  this.empSalary = empSalary;  }  @Override  public String toString() {  return empId + " " + empName + " " + empSalary;  }  @Override  public int compareTo(Employee e) {  return Integer.compare(empId, e.empId);  }  } |

|  |
| --- |
| /\*  \* 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 static java.util.Collections.sort;  import java.util.Comparator;  /\*\*  \*  \* @author Ramu Vallapurapu  \*/  public class EmployeeDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  System.out.println("Answer for question12: Ramu Vallapurapu");  ArrayList<Employee> employee = new ArrayList<Employee>();  Employee e1 = new Employee(37, "David Copperfield", 10000);  employee.add(e1);  Employee e2 = new Employee(17, "Diego Maradonna", 60000);  employee.add(e2);  Employee e3 = new Employee(135, "Ulrich Neilson", 15000);  employee.add(e3);  Employee e4 = new Employee(55, "Martha Stweart", 12000);  employee.add(e4);  Employee e5 = new Employee(22, "Lucifer Morningstar", 55000);  employee.add(e5);  System.out.println("Default List");  for (Employee item : employee) {  System.out.println(item.toString());  }  System.out.println("--------------------------------");  sort(employee);  System.out.println("Sorting by empID");  for (Employee item : employee) {  System.out.println(item.toString());  }  System.out.println("--------------------------------");  System.out.println("Sorting by Salary");  sort(employee, new Comparator<Employee>() {  @Override  public int compare(Employee e1, Employee e2) {  return Double.compare(e1.empSalary, e2.empSalary);  }  });  for (Employee item : employee) {  System.out.println(item.toString());  }  System.out.println("--------------------------------");  Collections.sort(employee, new Comparator<Employee>() {  @Override  public int compare(Employee e1, Employee e2) {  if (e1.getEmpName().compareTo(e2.getEmpName()) == 0) {  return e1.getEmpName().compareTo(e2.getEmpName());  } else {  return e1.getEmpName().compareTo(e2.getEmpName());  }  }  });  System.out.println("Sorting by Employee Name:");  for (Employee item : employee) {  System.out.println(item.toString());  }  System.out.println("--------------------------------");  }  } |



**Github Link :** [**https://github.com/s541910/Vallapurapu\_Spring2021FinalExam.docx.git**](https://github.com/s541910/Vallapurapu_Spring2021FinalExam.docx.git)