Day 9 - 11th June 2025

|  |
| --- |
| Inheritance, Polymorphism, Encapsulation, Abstraction, Interfaces, Exception Handling, |
| Collections Framework intro, Streams, File I/O, Multithreading overview |

Exceptions:

Types Exceptions:

Task 1:

What do you understand by exceptions?

Exceptions are basically errors that happen when the program is running, like when we try to open a file that doesn't exist or divide by zero. Java gives us a way to catch these errors and handle them properly instead of letting our program crash.  
  
Task 2:

What are the categories of Exceptions do we have in Java? What are they?

Categories of Exceptions in Java:

1. Checked Exceptions
2. Unchecked Exceptions (Runtime Exceptions)
3. Errors

Task 3:

Can you try the below code snippet and let me know which kind of exception is this

// Java program to demonstrates handling

// the exception using try-catch block

import java.io.\*;

class Geeks {

    public static void main(String[] args)

    {

        int n = 10;

        int m = 0;

        try {

            // Code that may throw an exception

            int ans = n / m;

            System.out.println("Answer: " + ans);

        }

        catch (ArithmeticException e) {

            // Handling the exception

            System.out.println(

                "Error: Division by zero is not allowed!");

        }

        finally {

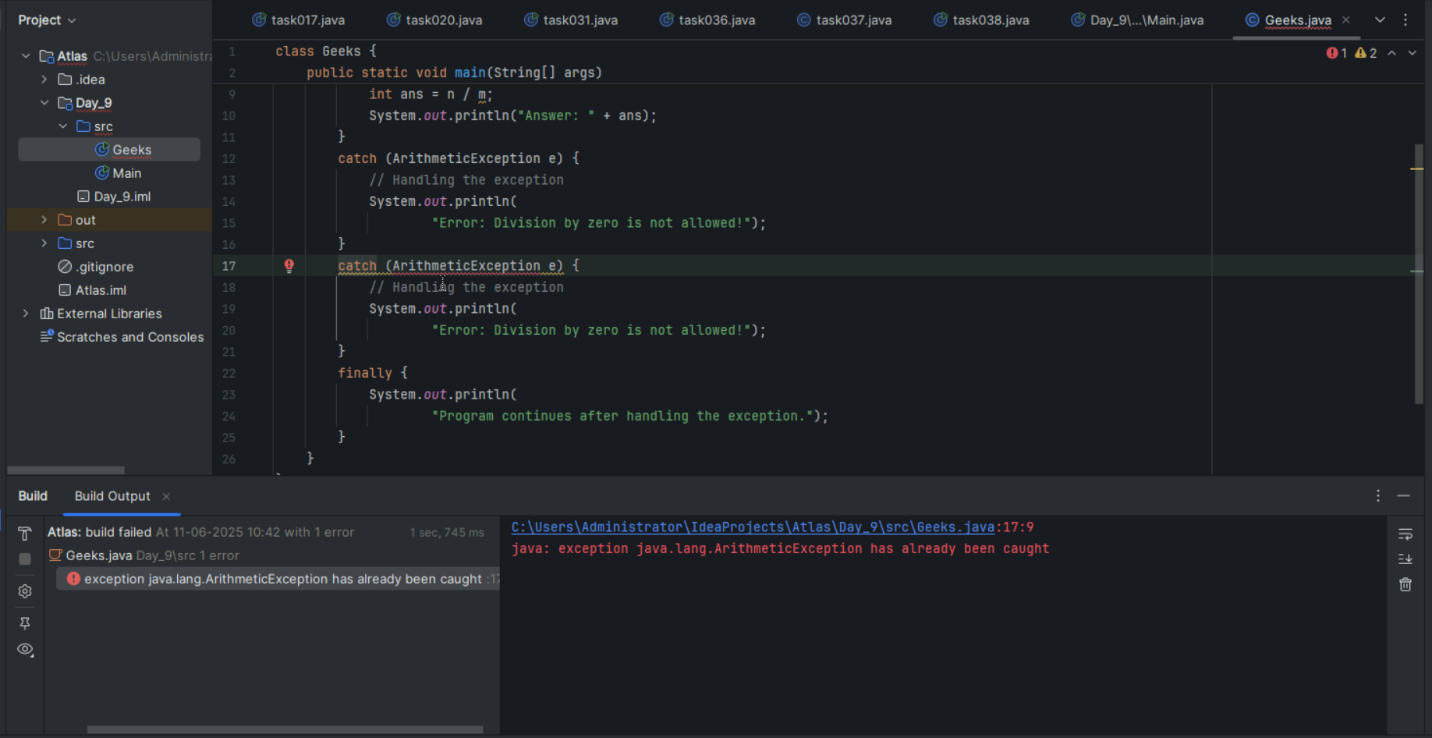
            System.out.println(

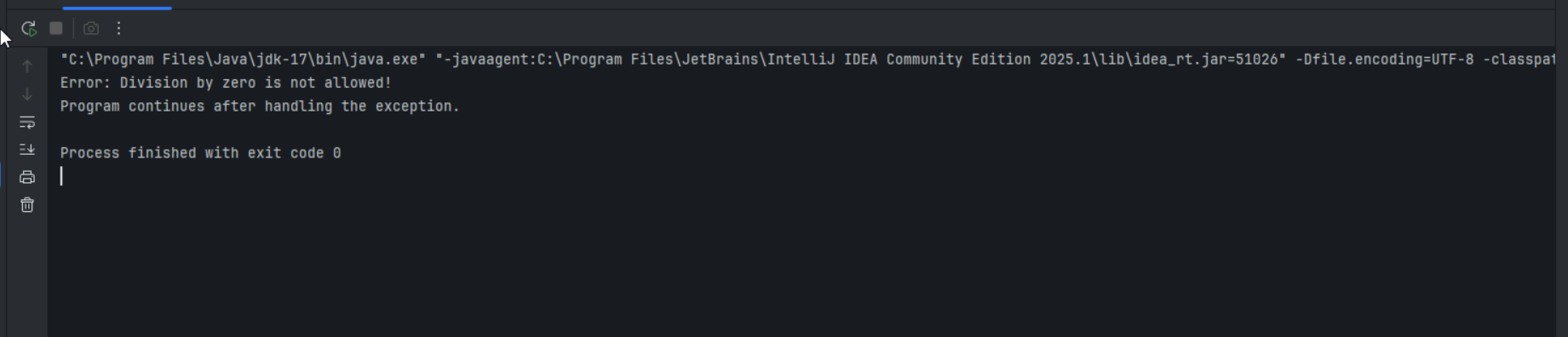
                "Program continues after handling the exception.");

        }

    }

}





Task 4:

List of checked and unchecked exceptions.

Checked Exceptions:

* IOException
* FileNotFoundException
* SQLException
* ClassNotFoundException
* InterruptedException
* ParseException
* MalformedURLException
* UnknownHostException
* SocketException
* CloneNotSupportedException

Unchecked Exceptions:

* NullPointerException
* ArrayIndexOutOfBoundsException
* IllegalArgumentException
* NumberFormatException
* ArithmeticException
* ClassCastException
* IllegalStateException
* UnsupportedOperationException
* StringIndexOutOfBoundsException
* ConcurrentModificationException

Task 5:

Try with Multiple catch blocks  …. Execute the below code snippet n display the out .. along with reason..

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 0;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("ArrayIndexOutOfBoundsException thrown  :" + e);

      }catch (Exception e) {

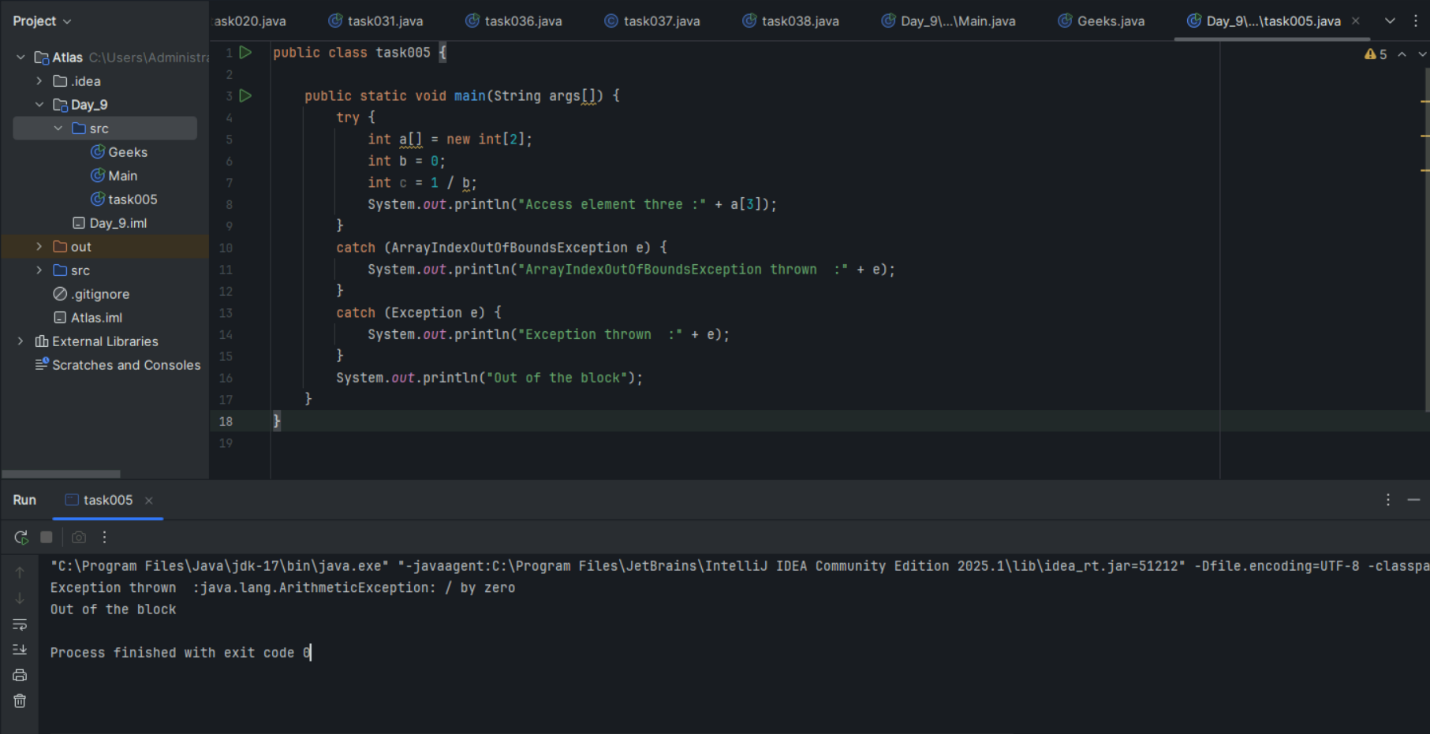
          System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}



**Reason:** The line int c = 1/b; throws ArithmeticException (division by zero). This exception is caught by the second catch block since ArithmeticException extends Exception. The array access line never executes because the exception occurred first.

Task 6:

What is the output of the below code… give your  reason for the output

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 0;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArithmeticException e) {

         System.out.println("ArithmeticException thrown  :" + e);

      }

      catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("ArrayIndexOutOfBoundsException thrown  :" + e);

      }catch (Exception e) {

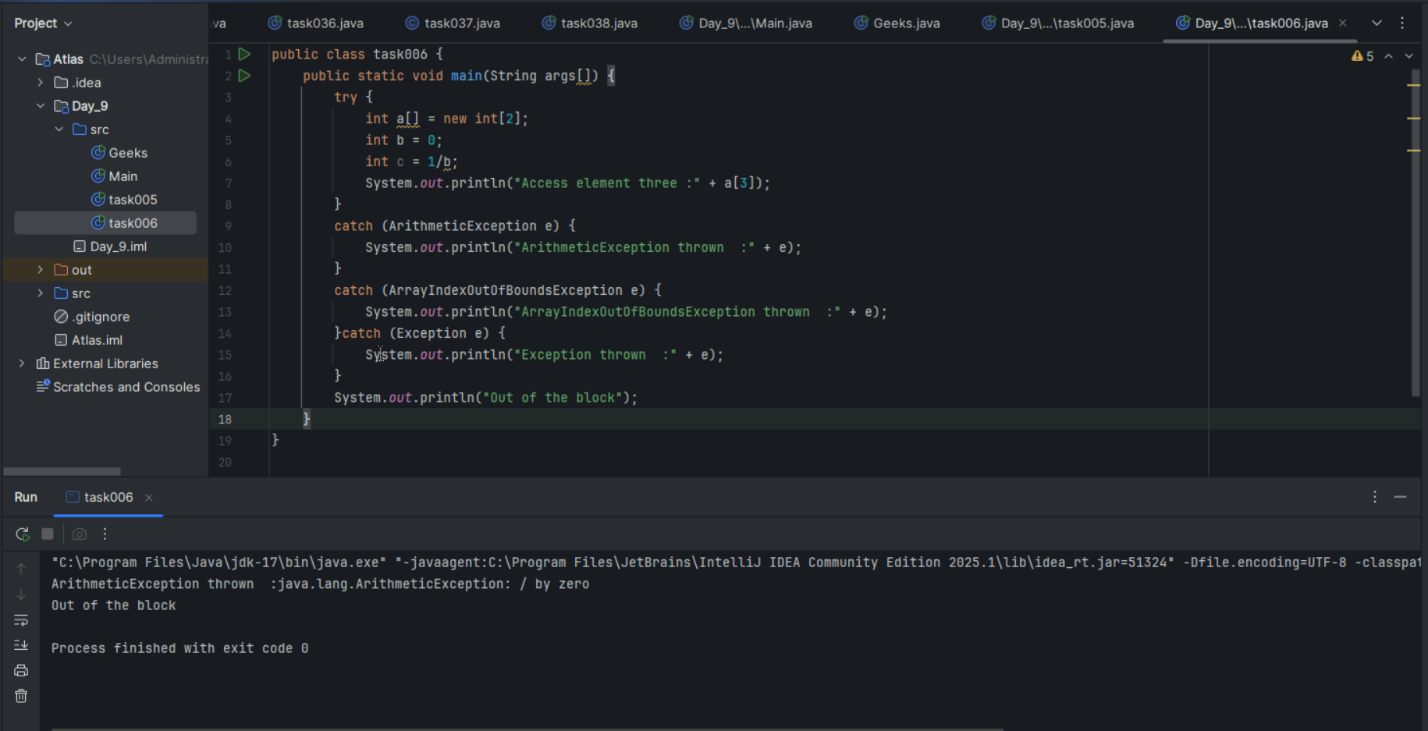
          System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}



Task 7:

In the below code we are having use multiple catch in a single statement: find the output and try to understand the code..

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         int b = 0;

         int c = 1/b;

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {

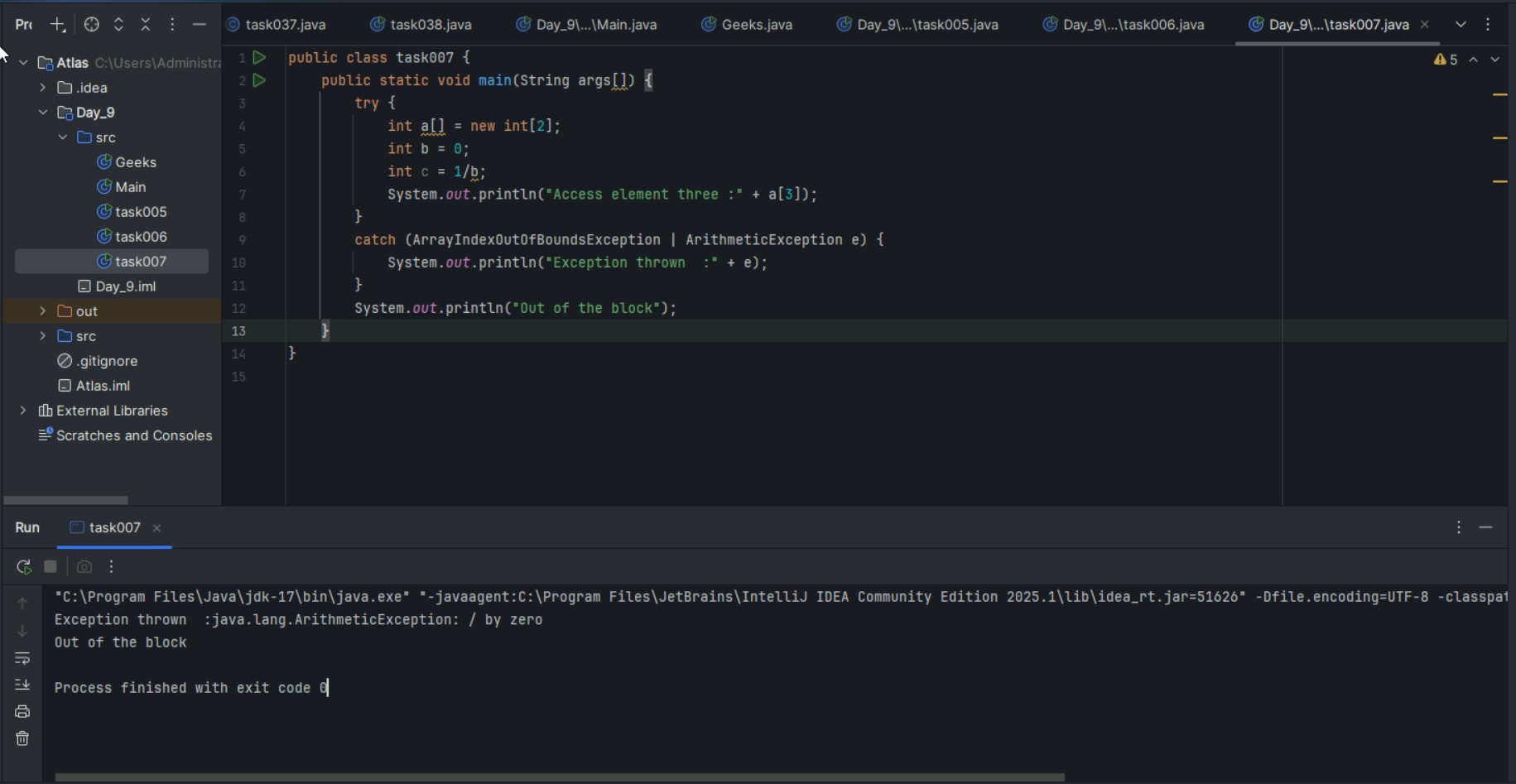
         System.out.println("Exception thrown  :" + e);

      }

      System.out.println("Out of the block");

   }

}



Naested try blocks

Task 008:

public class ExcepTest {

   public static void main(String args[]) {

      try {

         int a[] = new int[2];

         try {

            int b = 0;

            int c = 1/b;

         }catch(Exception e) {

            System.out.println("Exception thrown: " + e);

         }

         System.out.println("Access element three :" + a[3]);

      }

      catch (ArrayIndexOutOfBoundsException e) {

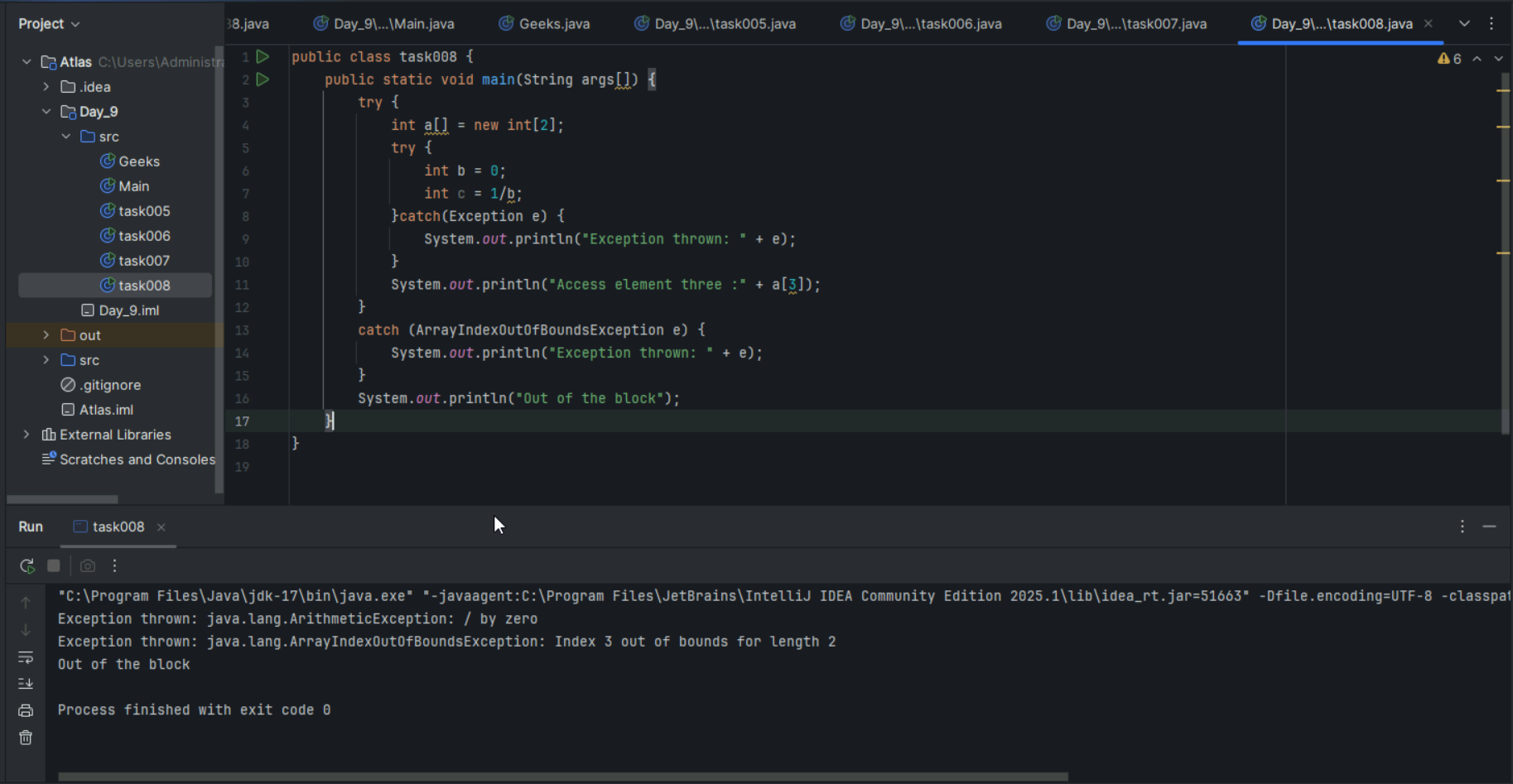
         System.out.println("Exception thrown: " + e);

      }

      System.out.println("Out of the block");

   }

}



Throw and Throws:

Task 009

// Demonstrating how to throw an exception

class MyClass {

    static void fun() throws IllegalAccessException

    {

        System.out.println("Inside fun(). ");

        throw new IllegalAccessException("demo");

    }

    public static void main(String args[])

    {

        try {

            fun();

method2();   → arrayindex…

Method3()  —> file not found….

        }

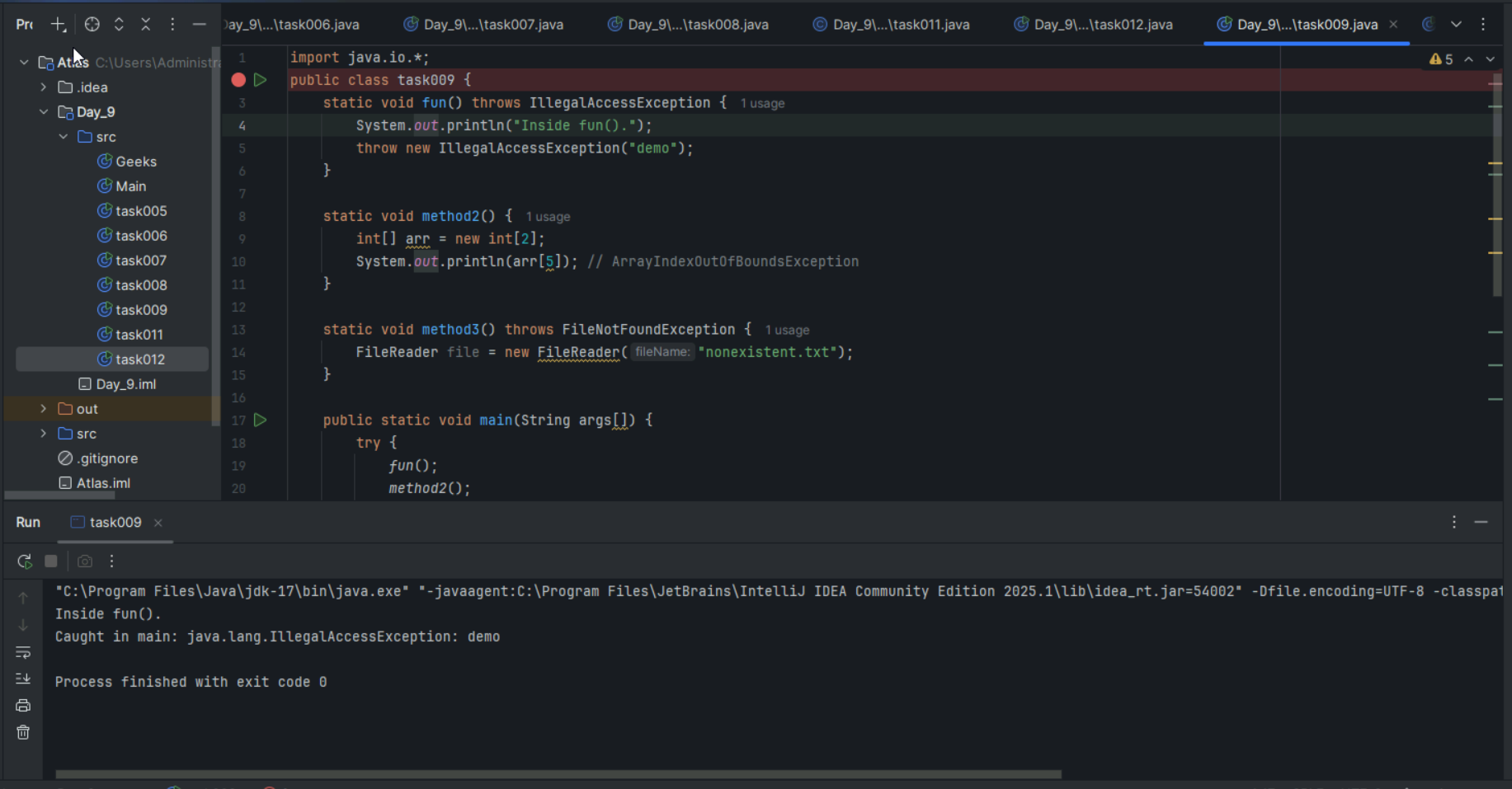
        catch (IllegalAccessException e) {

            System.out.println("Caught in main.");

        }

    }

}



Task 10:

Observe

import java.util.ArrayList;

class Main {

    public static void main (String[] args) {

       // Creating an ArrayList

       ArrayList<Integer> a = new ArrayList<Integer>();

       // Adding Element in ArrayList

       a.add(1);

       a.add(2);

       a.add(3);

       // Printing ArrayList

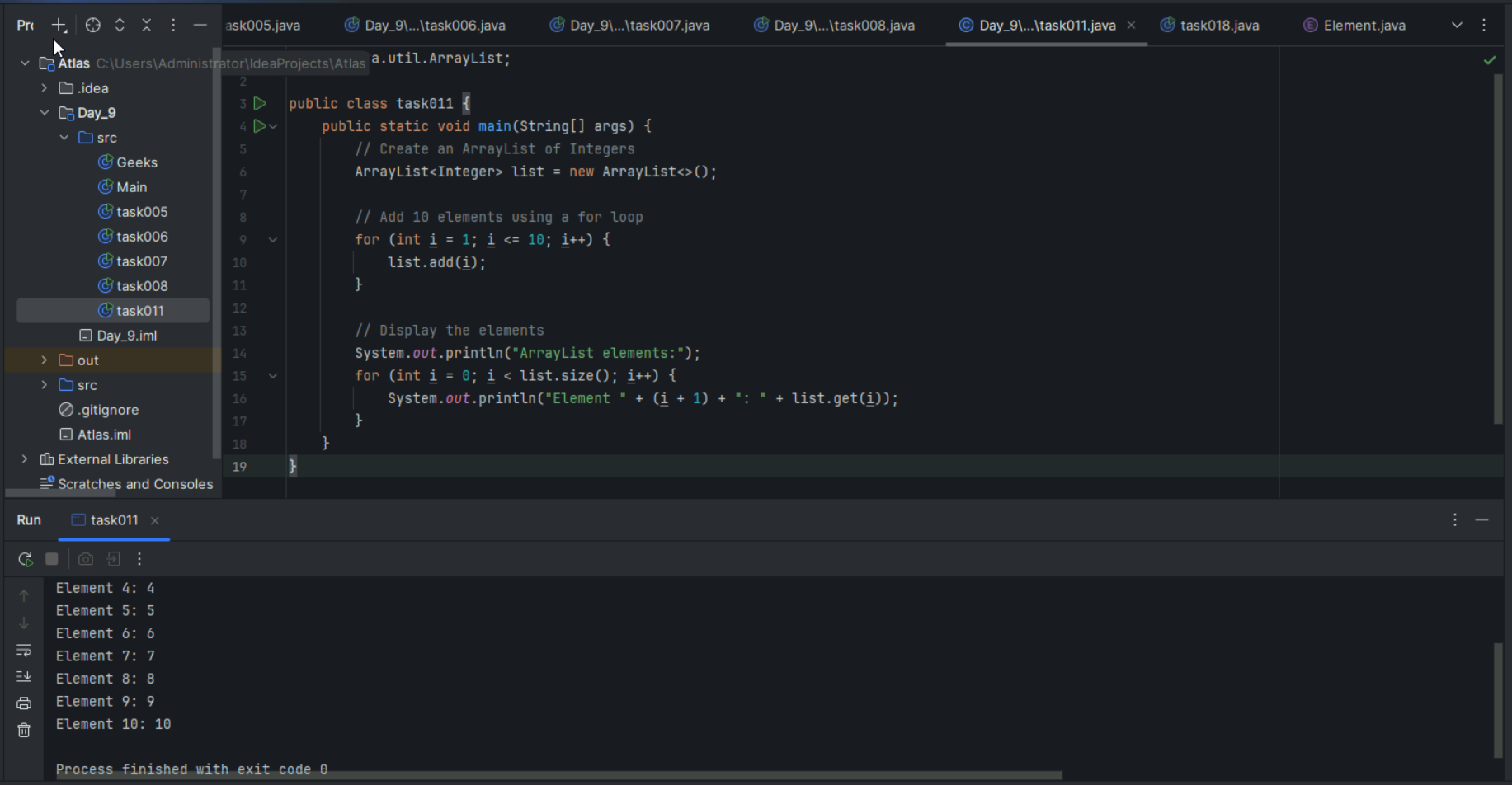
       System.out.println(a);

    }

}

Task 011:

Wap to create an array list to display 10 elements using for loop.



Task 012

Find the output of the be code snippet..

// Addition, Deletion and Updation of Element

import java.util.\*;

class Main {

    public static void main(String args[]){

        ArrayList<String> al = new ArrayList<>();

        al.add("Prasunamba");

        al.add("Meher");

       System.out.println("Orignal List : "+al);

        al.add(1, "Hello");

       System.out.println("After Adding element at index 1 : "+ al);

       al.remove(0);

       System.out.println("Element removed from index 0 : "+ al);

       al.remove("Prasunamba");

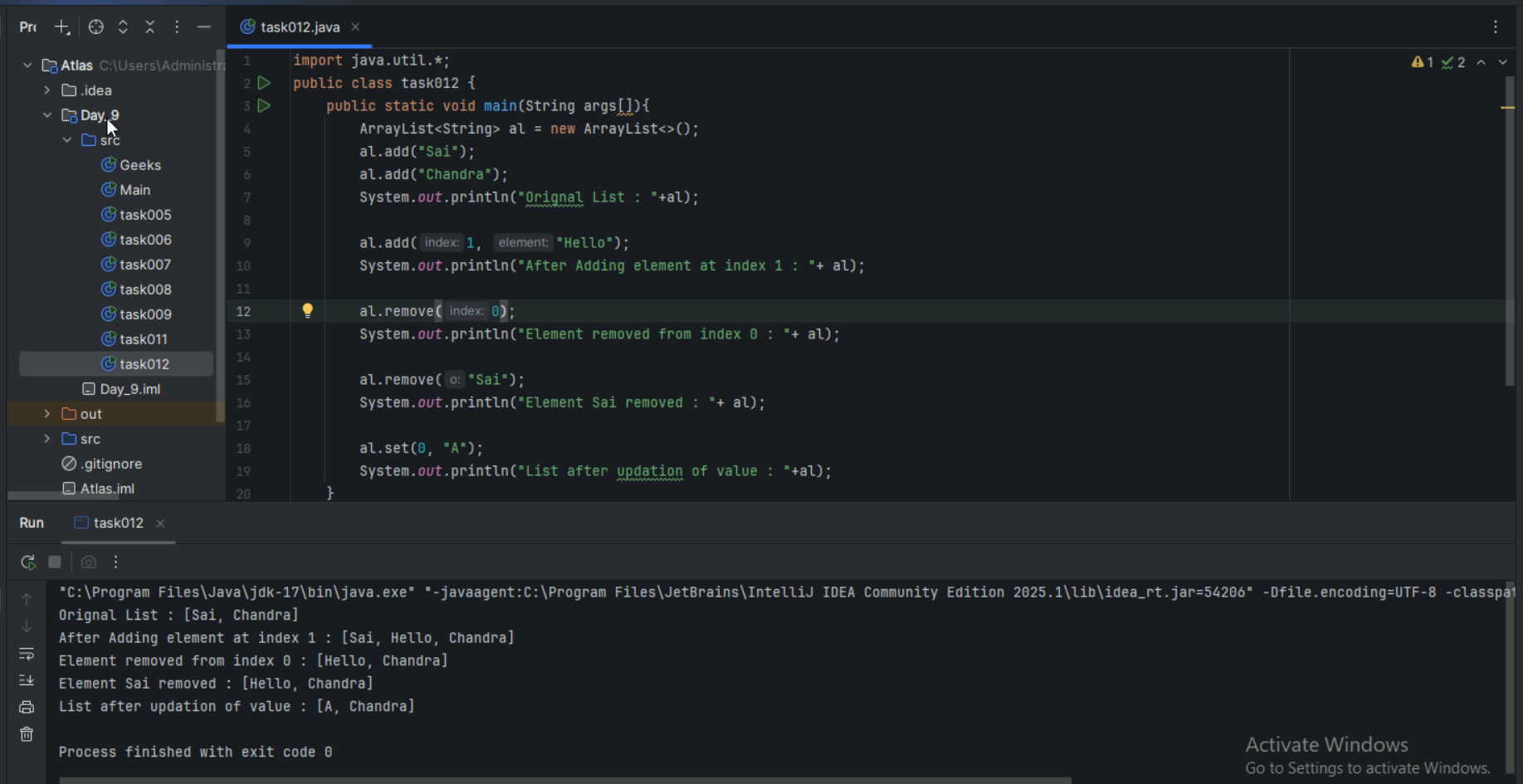
       System.out.println("Element Prasunamba removed : "+ al);

        al.set(0, "K");

        System.out.println("List after updation of value : "+al);

    }

}



Task 013:

Run the code and see hope the user defined exception works..

User defined Exception:

// A Class that represents user-defined exception

class Customer extends Exception {// predefined class Exception

    public Customer(String m) { // constructor with parameters

        super(m); // parent class constructor

    }

}

// A Class that uses the above MyException

public class setText {

    public static void main(String args[]) {

        try {

            // Throw an object of user-defined exception

            throw new MyException("This is a custom exception");

        }

        catch (MyException ex) {

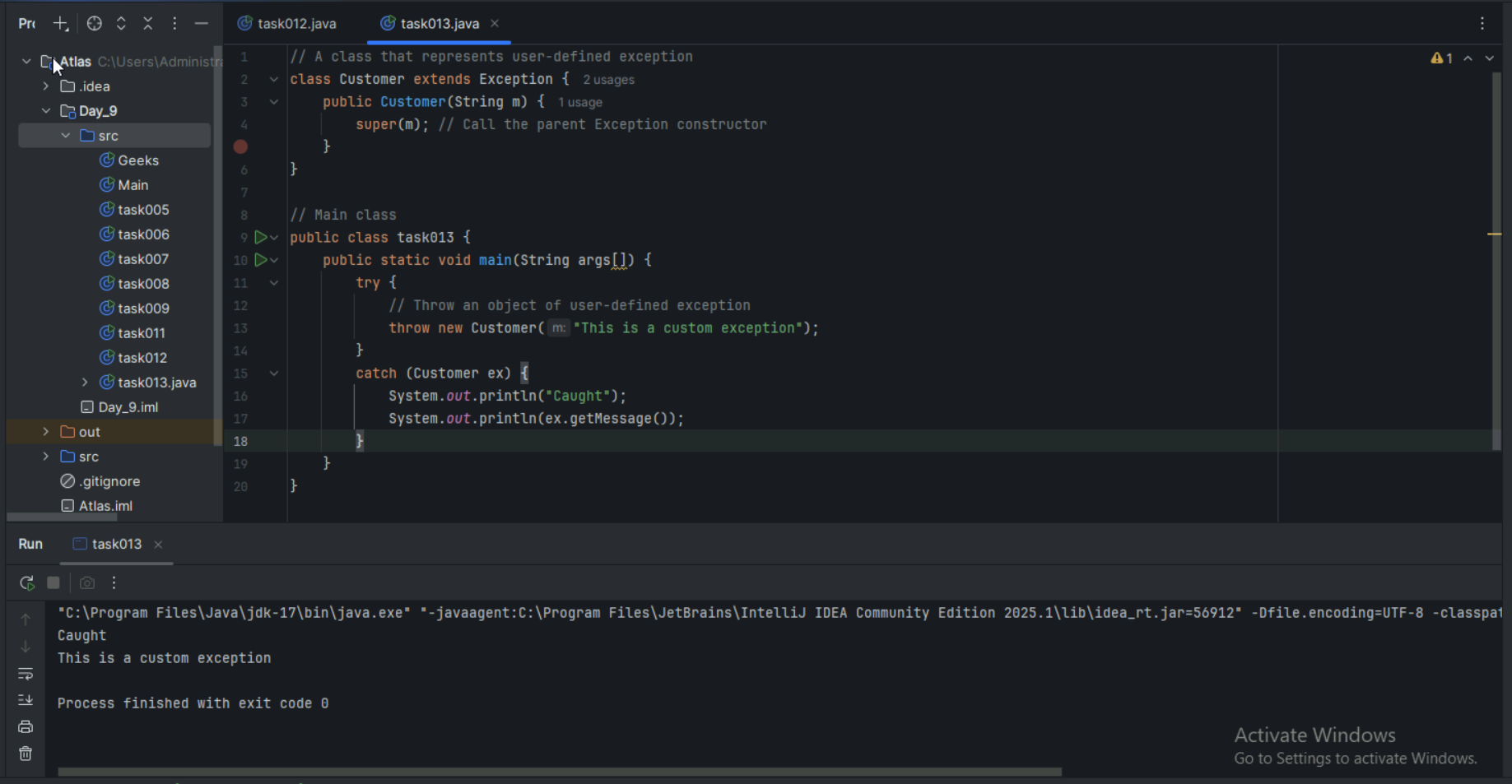
            System.out.println("Caught");

            System.out.println(ex.getMessage());

        }

    }

}



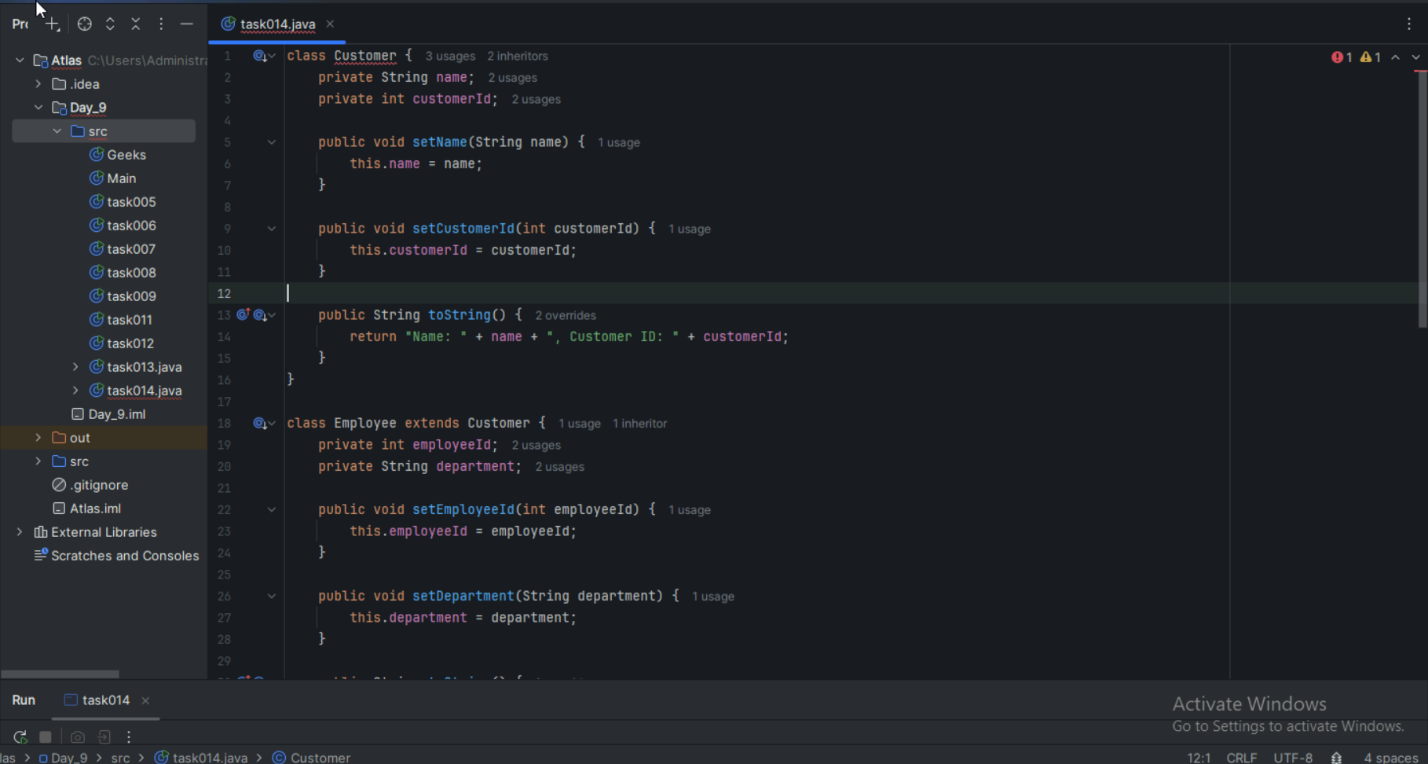
Task 014:

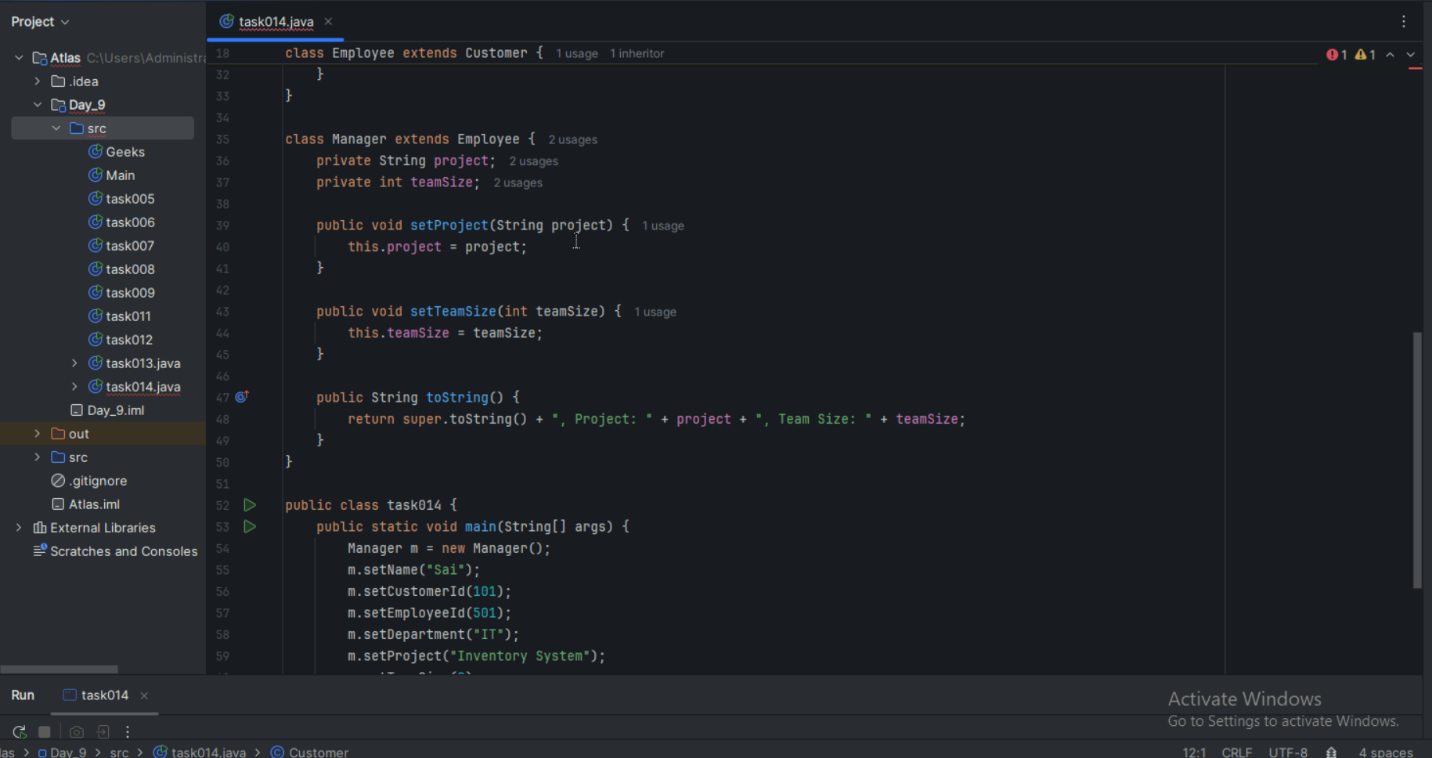
Inheritance

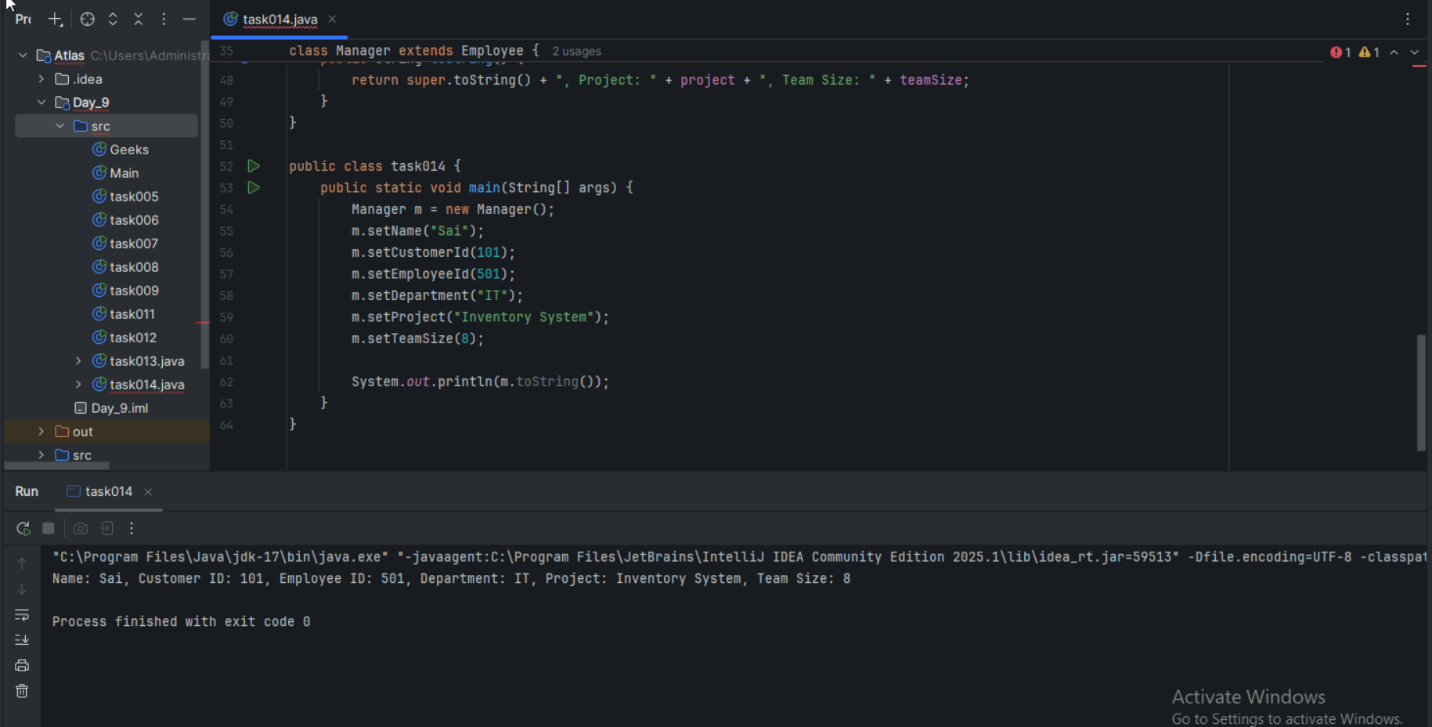
Classes customer, employee, Manager  … 2 variables in each class

Driver class – display all the variables… toString();

Hint : use getter and setters..







Task 015:

What is the output of the below code snippet..  Explain ..

class OuterClass {

  int x = 10;

  class InnerClass {

    int y = 5;

  }

}

public class Main {

  public static void main(String[] args) {

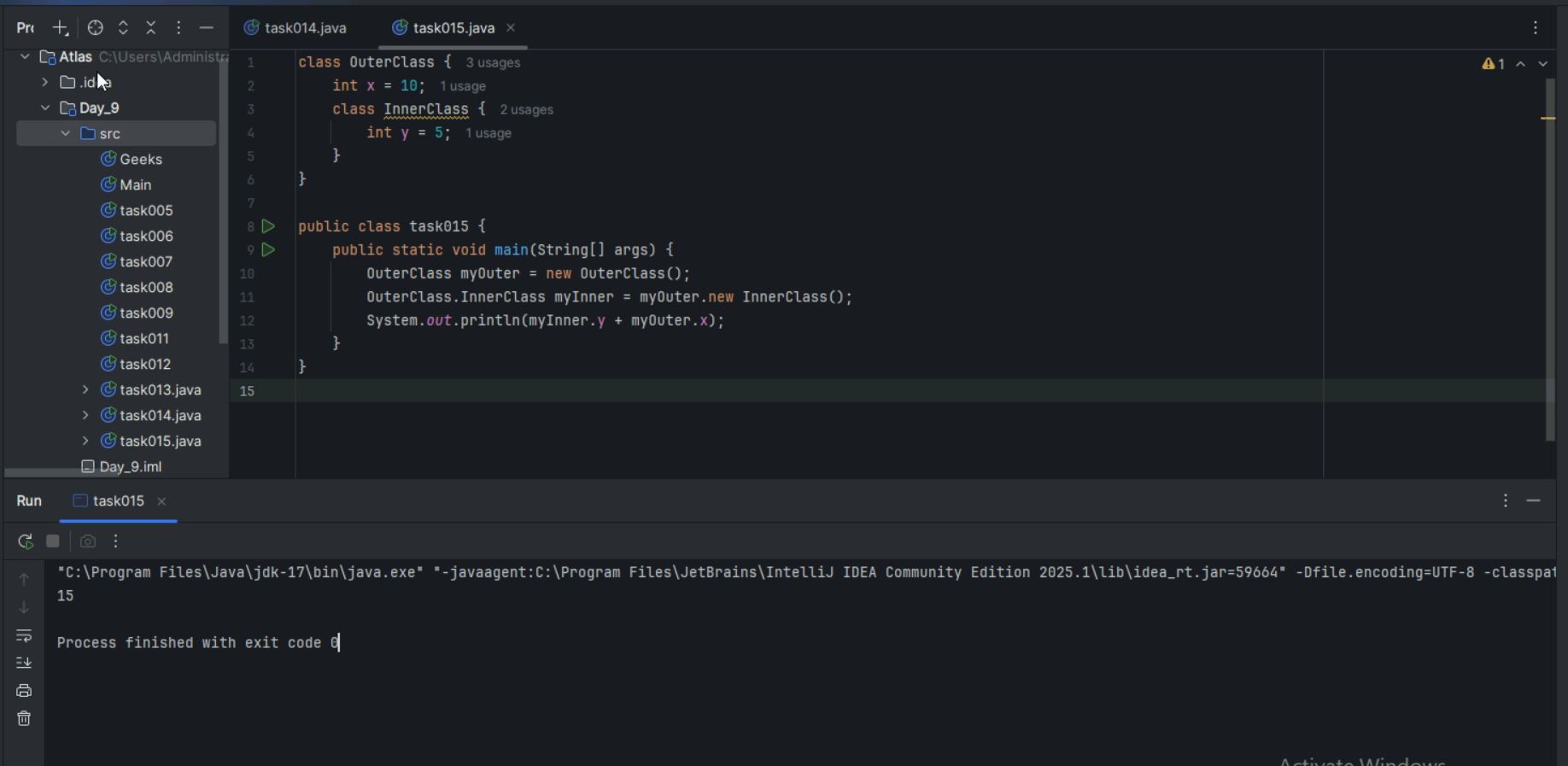
    OuterClass myOuter = new OuterClass();

    OuterClass.InnerClass myInner = myOuter.new InnerClass();

    System.out.println(myInner.y + myOuter.x);

  }

}



Task 016:

Use the above code and make the inner class as private and see the output..

Ex: Private class InnerClass {

Task 017:

Use the above code Task 015 and make the inner class static … see the output and explain..

Ex: static class InnerClass {

Task 018:

Use the above code Task 015 and create a method in innerclass and return the outer class variable

class OuterClass{

Int x = 50;

Class InnerClass {

Public int innerMethod() {

Return x;

}

}

}

Public class DriverClass {

psvm(){

OuterClass myOuter = new OuterClass();

OuterClass.InnerClass myInner = myOuter.new InnerClass();

     System.out.println(myInner.innerMethod());

}

}

Task 019  — query by vivek

class OuterClass {

  int x = 10;

  static class InnerClass {

    static int y = 5;

  }

}

public class Main {

  public static void main(String[] args) {

     OuterClass.InnerClass myInner = new OuterClass.InnerClass();

    System.out.println(myInner.y);

  }

}