Java practicals

FS19CO042

Omkar Phansopkar

FS19CO042 SECOND YEAR, FIRST SHIFT

Practical no. 1

FS19CO042

Aim:

Getting started with Java Application Development using IDE

1.1 Check whether latest version of java (at least JDK 1.8)is installed or not. If not then download and install it.

1.2 Download and install the IntelliJ IDEA Community Edition/ NetBeans IDE 8.1/ Eclipse Neon or later version of IDE

1.3 Create a Java Project/ Application in the IDE

1.4 Create a Java class Person containing two variables name and yearOfBirth of appropriate data types, take inputs from the command line argument, a method to display the name and age of the person.

1.5 Save the project and run it.

1.6 Explore all the features (the menu and shortcuts) of the IDE. Learn about their use

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called *OpenJDK*.

Java virtual machine (JVM)

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.
The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.

Java programs are compiled by the Java compiler into *bytecode*. The Java virtual machine interprets this *bytecode* and executes the Java program

Java Development kit (JDK)

The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries

Java program workflow:

javac hello.java -> Compiles java source file into a class file and new hello.class file is generated
java hello -> Executes class file generated in java virtual machine.

Base structure of Java program:

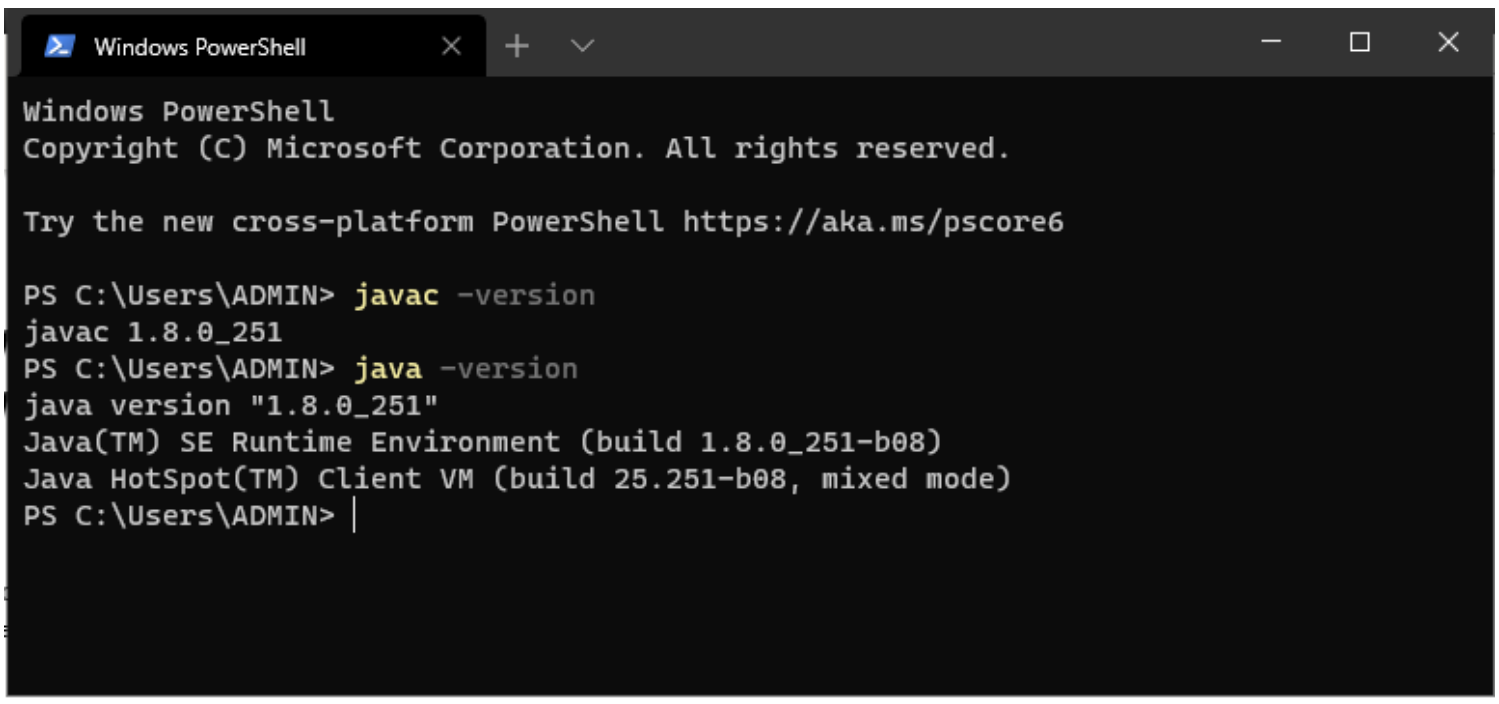
```
package test;
class Hello{
    public static void main(String[] args){
        // Code goes here
```

```
    }
}
```

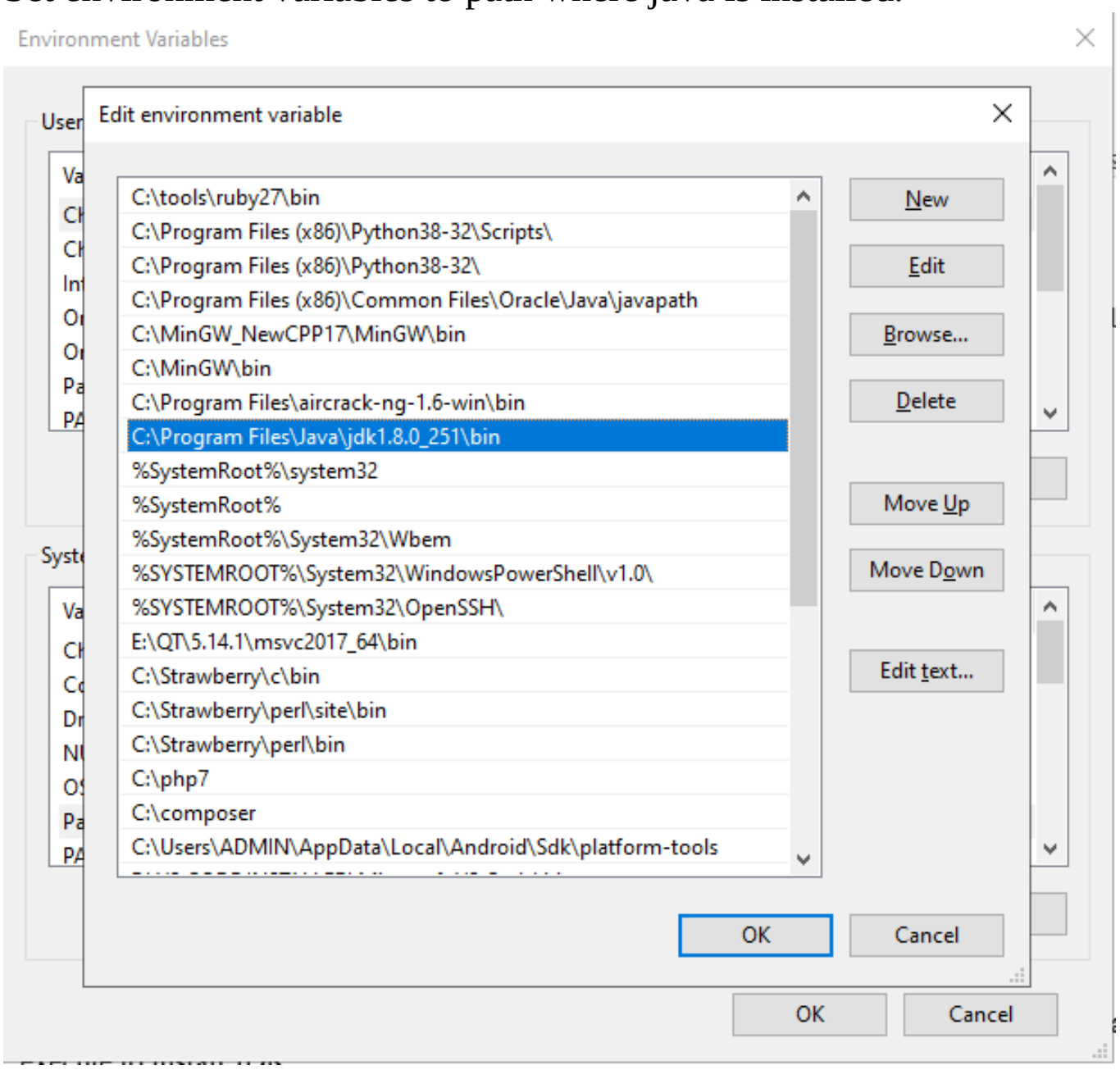
Steps:

1.1 Check whether latest version of Java (at least JDK 1.8) is installed or not. If not then download and install it.

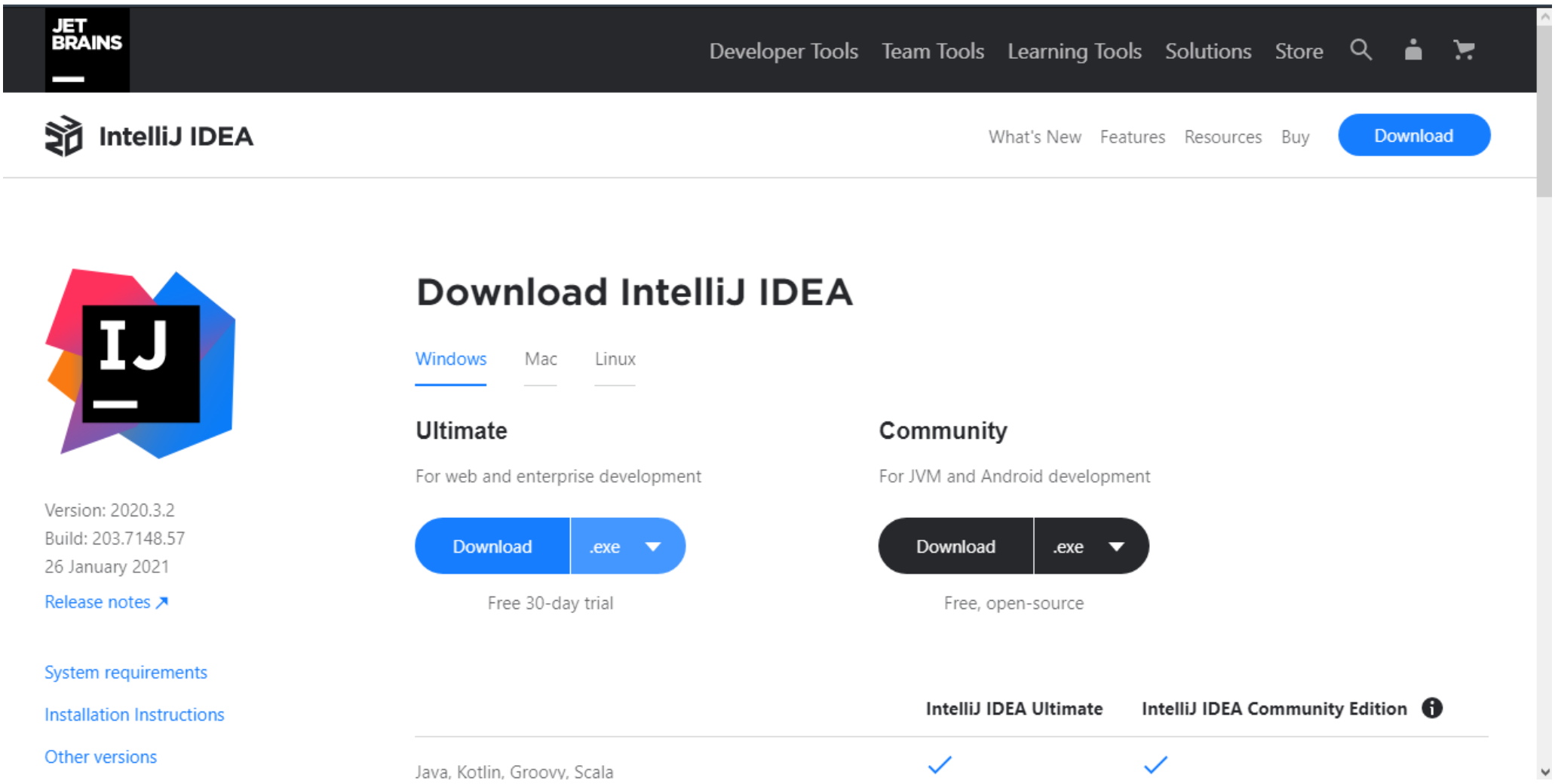
- Check if java compiler(javac) and jvm (java) are installed properly on system.
  Type these commands:
  javac -version
  java -version

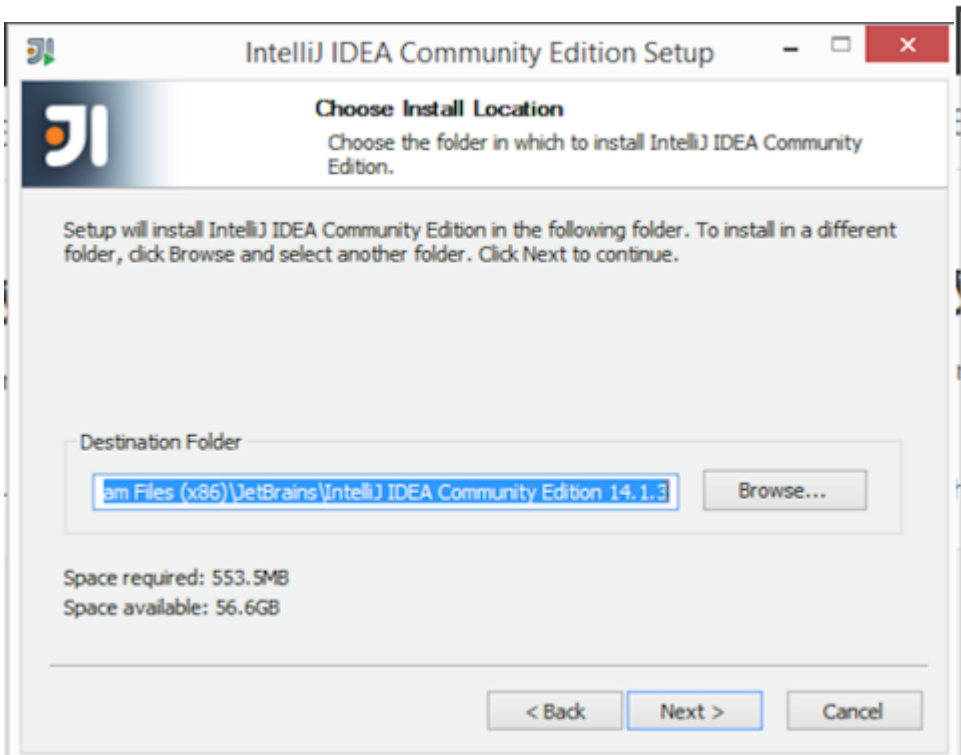


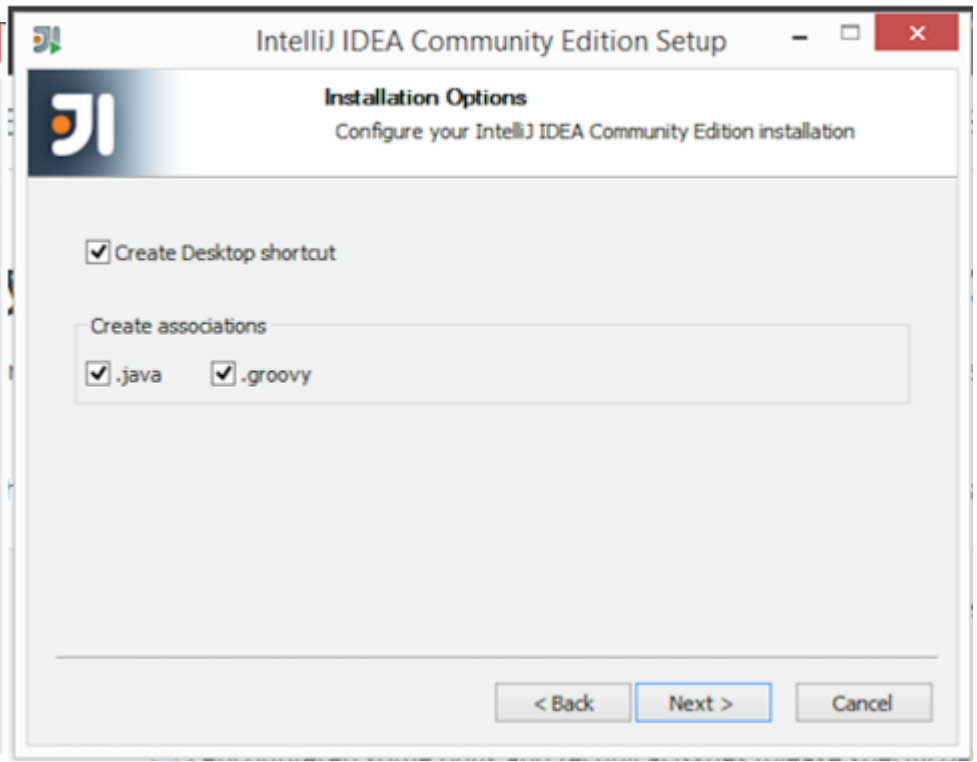
- If java compiler or jvm is not installed on system, download executibles from Oracle.com and execute to install JDK.
- Set environment variables to path where java is installed.

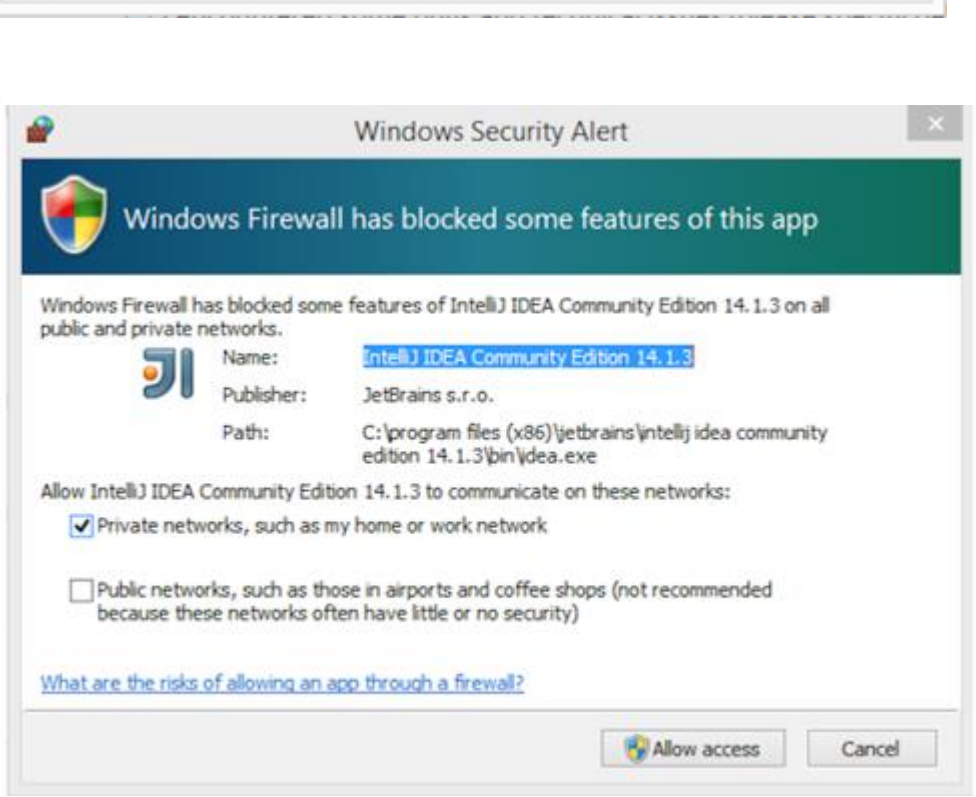


1.2 Download and install the IntelliJ IDEA Community Edition/ NetBeans IDE 8.1/ Eclipse Neon or later version of IDE

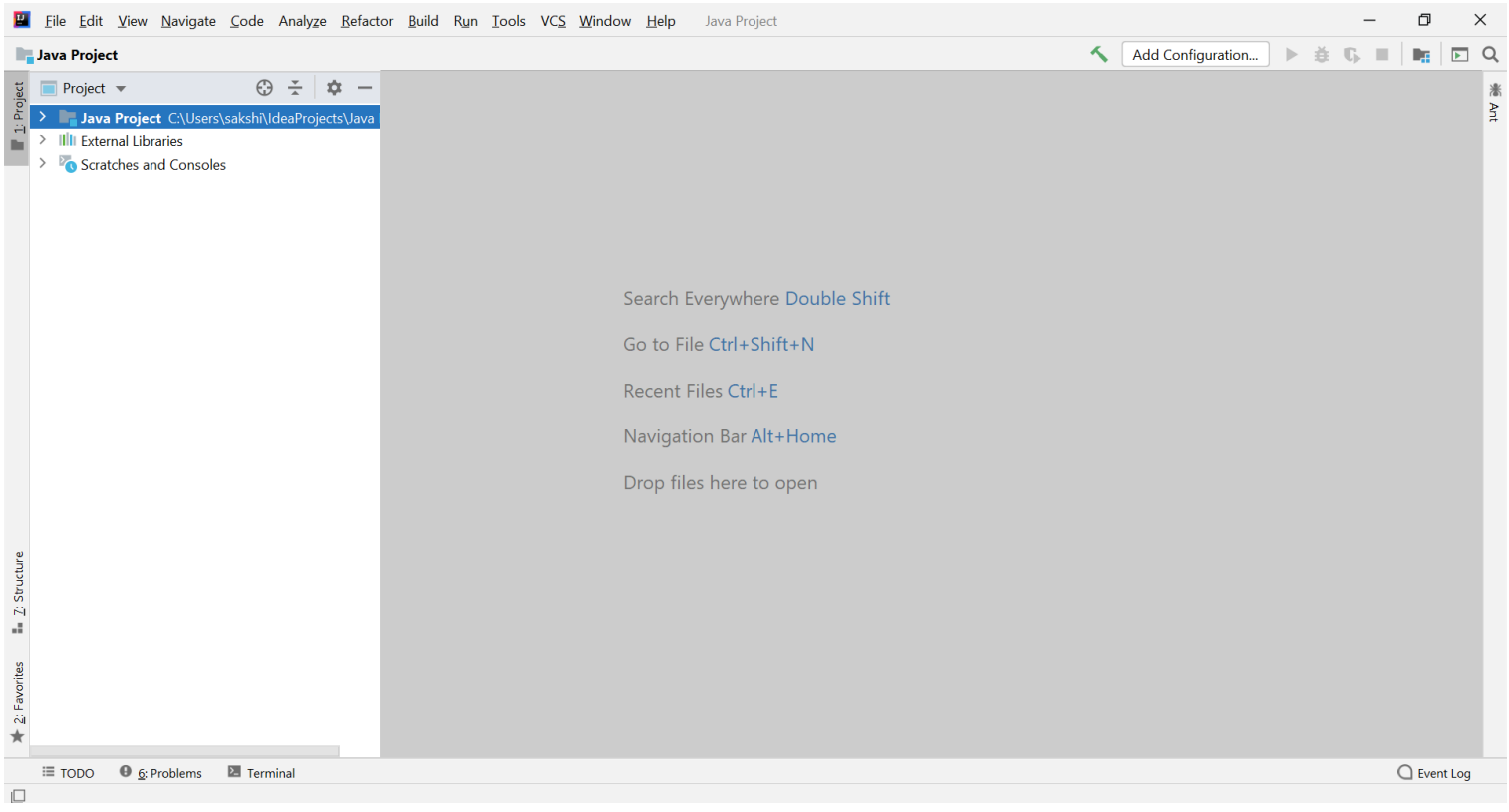








1.3 Create a Java Project/ Application in the IDE

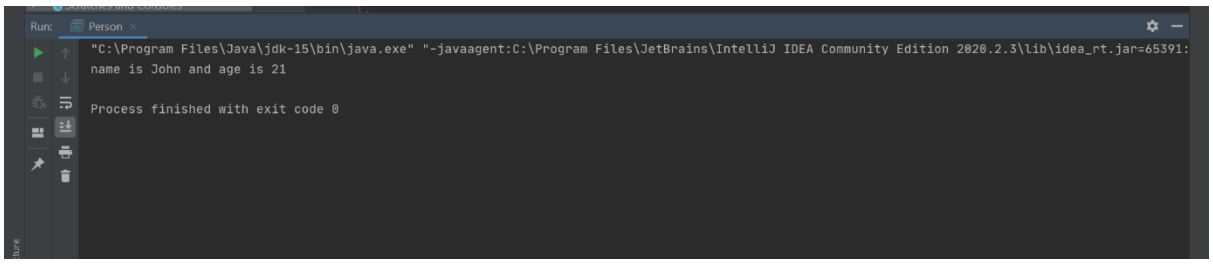


1.4 Create a Java class Person containing two variables name and yearOfBirth of appropriate data types, take inputs from the command line argument, a method to display the name and age of the person.

```
public class Person
{
   String name;
   int yearOfBirth;

   public static void disp_details(String a,int b)
   {
     int current_year=2021;
     int age=current_year-b;
     System.out.println("name is "+a+" and age is "+age);

   }


   public static void main(String [] args )
   {
      disp_details("sakshi",2004);
   }
}
```

1.5 Save the project and run it.

output :



1.6 Explore all the features (the menu and shortcuts) of the IDE. Learn about their use

Shortcut	Action
Double Shift	Search Everywhere

	Find anything related to IntelliJ IDEA or your project and open it, execute it, or jump to it.
Ctrl+Shift+A	Find Action Find a command and execute it, open a tool window or search for a setting.
Alt+Enter	Show intention actions and quick-fixes Fix highlighted error or warning, improve or optimize a code construct.
F2 Shift+F2	Navigate between code issues Jump to the next or previous highlighted error.
Ctrl+E	View recent files Select a recently opened file from the list.
Ctrl+Shift+Enter	Complete current statement Insert any necessary trailing symbols and put the caret where you can start typing the next statement.
Ctrl+Alt+L	Reformat code Reformat the whole file or the selected fragment according to the current code style settings.
Ctrl+Alt+Shift+T	Invoke refactoring Refactor the element under the caret, for example, safe delete, copy, move, rename, and so on.
Ctrl+W Ctrl+Shift+W	Extend or shrink selection Increase or decrease the scope of selection according to specific code constructs.
Ctrl+/ Ctrl+Shift+/	Add/remove line or block comment Comment out a line or block of code.
Ctrl+B	Go to declaration Navigate to the initial declaration of the instantiated class, called method, or field.
Alt+F7	Find usages Show all places where a code element is used across your project.
Alt+1	Focus the Project tool window
Escape	Focus the editor

Conclusion: Thus, we verified/installed java and IntelliJ IDE and ran first program on it. We also explored various features of the IDE.

Practical no. 2

FS19CO042

Aim: Perform following programs.

2.1 Write a program to print “Hello World”.

2.2 Write a program to print addition of two integers.

2.3 Write a program to convert a numeric string into int.

2.4 Write a program to print addition of two integers input from command line arguments.

2.5 Write a program to take two integers from command line, subtract the smaller number from the greater and print the result.

2.6 Write a program to take n integers from command line and print their sum of product (product of first number and last number added to product of second number and second last number and so on).

2.7 Consider any two integers. Write a program to print sum of their squares.

2.8 Write a program to find square root of a given positive integer using Heron’s method to find square root.

2.9 Write a program to sort and print the names of students taken from command line in alphabetical order.

2.10 Write a program to print total numbers of vowels and consonants in a given string.

2.11 Given two English words, write a program to check if the first word is anagram of the second word. (An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. (Example: Anagram of TOM MARVOLO RIDDLE is I AM LORD VOLDEMORT.)

2.12 Write a program to print a missing number in a sorted integer array.

2.13 Write a program to find all the pairs of numbers on an integer array whose sum is equal to a given number.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

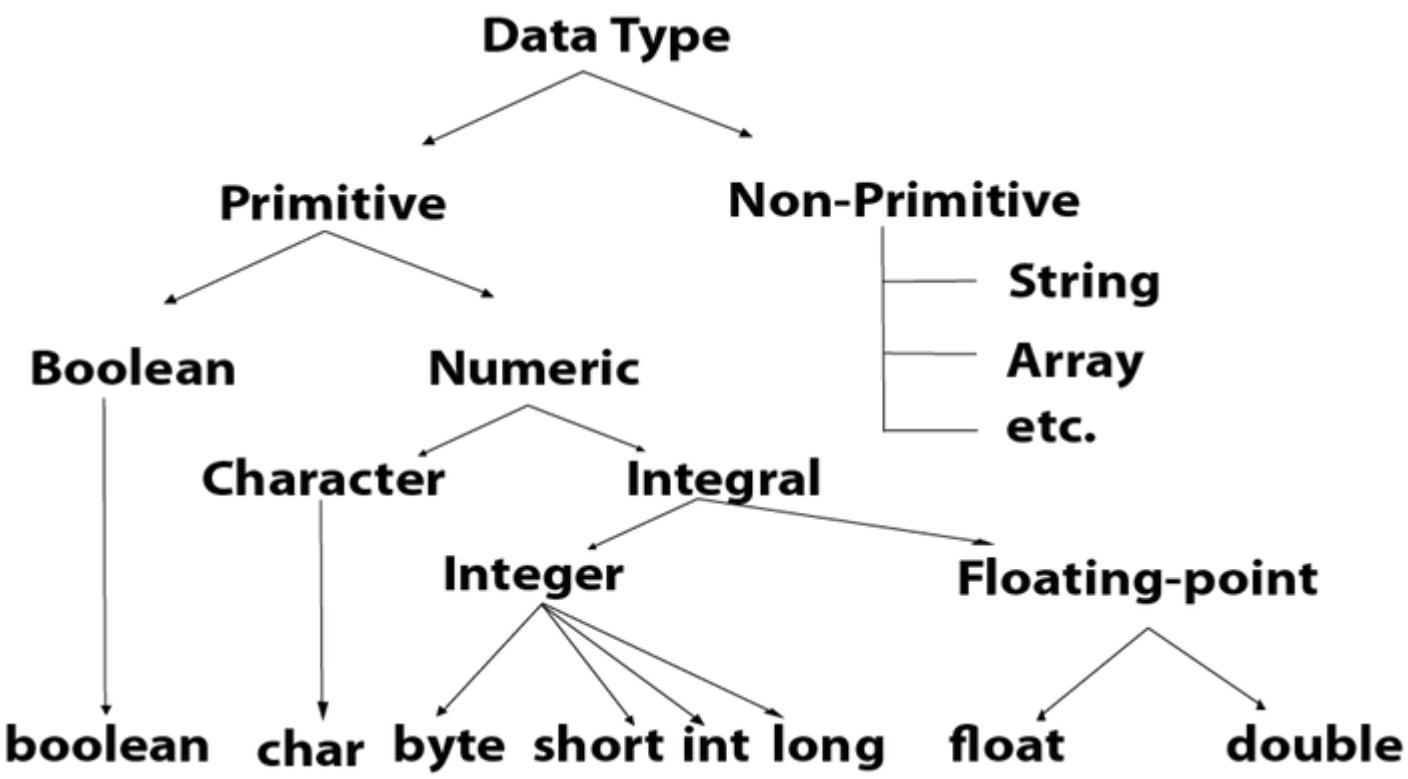
Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type

- short data type
- int data type
- long data type
- float data type
- double data type



Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Java Operators

In this tutorial, you'll learn about different types of operators in Java, their syntax and how to use them with the help of examples.

Operators are symbols that perform operations on variables and values. For example, `+` is an operator used for addition, while `*` is also an operator used for multiplication.

Operators in Java can be classified into 5 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators

---

1. Java Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example, a + b;

Here, the `+` operator is used to add two variables `a` and `b`. Similarly, there are various other arithmetic operators in Java.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. Java Assignment Operators

Assignment operators are used in Java to assign values to variables. For example,

int age;

age = 5;

Here, `=` is the assignment operator. It assigns the value on its right to the variable on its left. That is, **5** is assigned to the variable `age`.

Let's see some more assignment operators available in Java.

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

3. Java Relational Operators

Relational operators are used to check the relationship between two operands. For example,

```
// check is a is less than b

a < b;
```

Here, `>` operator is the relational operator. It checks if `a` is less than `b` or not.

It returns either `true` or `false`.

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **false**

4. Java Logical Operators

Logical operators are used to check whether an expression is `true` or `false`. They are used in decision making.

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

5. Java Unary Operators

Unary operators are used with only one operand. For example, `++` is a unary operator that increases the value of a variable by **1**. That is, `++5` will return **6**.

Different types of unary operators are:

Operator	Meaning
`+`	**Unary plus**: not necessary to use since numbers are positive without using it
`-`	**Unary minus**: inverts the sign of an expression
`++`	**Increment operator**: increments value by 1
`--`	**Decrement operator**: decrements value by 1
`!`	**Logical complement operator**: inverts the value of a boolean

Increment and Decrement Operators

Java also provides increment and decrement operators: `++` and `--` respectively. `++` increases the value of the operand by **1**, while `--` decrease it by **1**. For example,

```
int num = 5;

// increase num by 1

++num;
```

Here, the value of `num` gets increased to **6** from its initial value of **5**.

Example 5: Increment and Decrement Operators

In the above program, we have used the ++ and -- operator as **prefixes (++a, --b)**. We can also use these operators as **postfix (a++, b++)**.

There is a slight difference when these operators are used as prefix versus when they are used as a postfix.

6. Java Bitwise Operators

Bitwise operators in Java are used to perform operations on individual bits. For example,

```
Bitwise complement Operation of 35
35 = 00100011 (In Binary)
~ 00100011
_________
  11011100  = 220 (In decimal)
```

Here, `~` is a bitwise operator. It inverts the value of each bit (**0** to **1** and **1** to **0**).

The various bitwise operators present in Java are:

Operator	Description
`~`	Bitwise Complement

<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

These operators are not generally used in Java. To learn more, visit Java Bitwise and Bit Shift Operators.

Other operators

Besides these operators, there are other additional operators in Java.

Java instanceof Operator

The `instanceof` operator checks whether an object is an instanceof a particular class.

Here, `str` is an instance of the `String` class. Hence, the `instanceof` operator returns `true`.

Java Ternary Operator

The ternary operator (conditional operator) is shorthand for the `if-then-else` statement. For example,

variable = Expression ? expression1 : expression2

Here's how it works.

- If the Expression is true, expression1 is assigned to the variable.
- If the Expression is false, expression2 is assigned to the variable

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

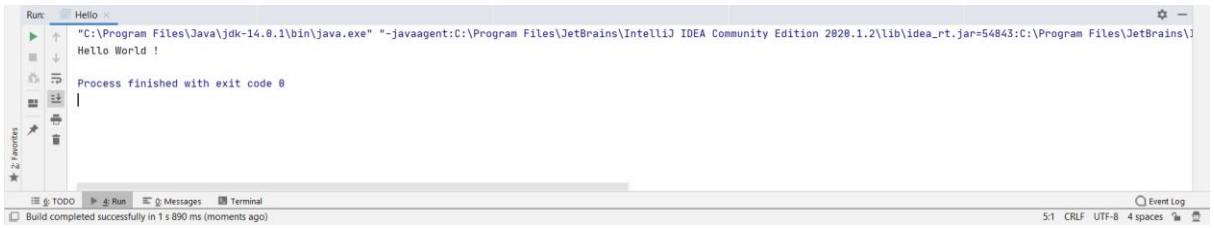
Code:

2.1 Write a program to print “Hello World”.

Code:

```
public class Hello {

   public static void main(String[] args) {
      System.out.println("Hello World !");
   }
}
```

Output:

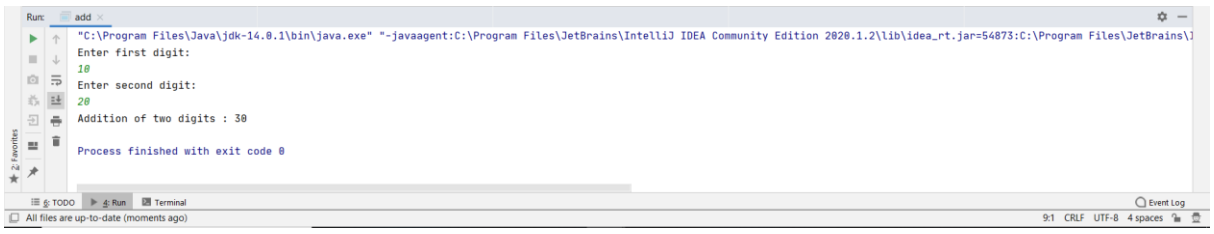


2.2 Write a program to print addition of two integers.

Code:

```
import java.util.Scanner;
public class add {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter first digit:");

      int a = scanner.nextInt();

      System.out.println("Enter second digit:");

      int b = scanner.nextInt();

      int c = a+b;

      System.out.println("Addition of two digits : " + c);

   }

}
```

Output:

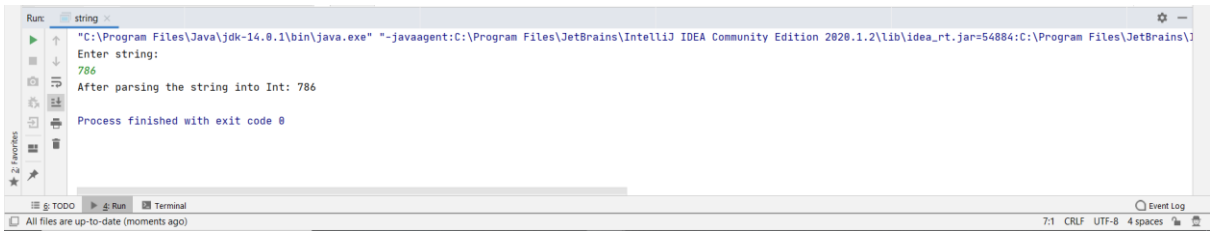


2.3 Write a program to convert a numeric string into int.

Code:

```
import java.util.Scanner;
public class string {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter string:");

      String s = scanner.nextLine();

      int number = Integer.parseInt(s);

      System.out.println("After parsing the string into Int: " + number);

   }

}
```

Output:



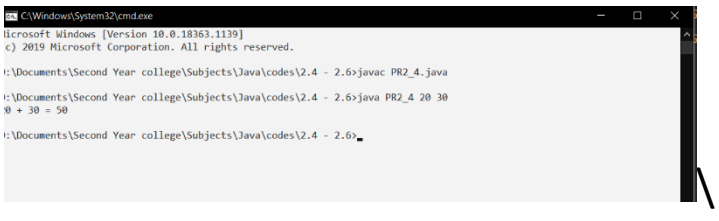
2.4 Write a program to print addition of two integers input from command line arguments.

Code:

```
public class PR2_4 {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      int num1, num2, sum;
```

```
        num1 = Integer.parseInt(args[0]);

        num2 = Integer.parseInt(args[1]);
        sum = num1+num2;
        System.out.println(num1+" + "+num2+" = "+sum);
    }
}
```

Output:

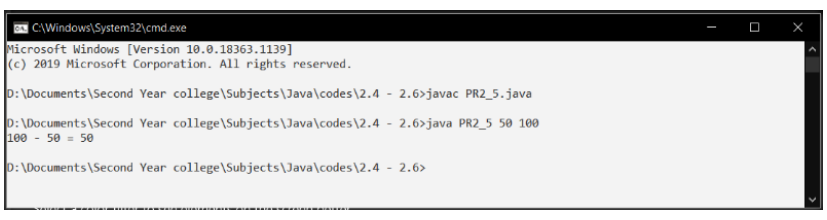


2.5 Write a program to take two integers from command line, subtract the smaller number from the greater and print the result.

Code:

```
public class PR2_5 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num1, num2;
         num1 = Integer.parseInt(args[0]);
         num2 = Integer.parseInt(args[1]);
        System.out.println(num1>num2 ? num1 +" - "+num2+" = "+(num1-num2) : num2 +" - "+num1+" =
"+(num2-num1));
    }
}
```

Output :



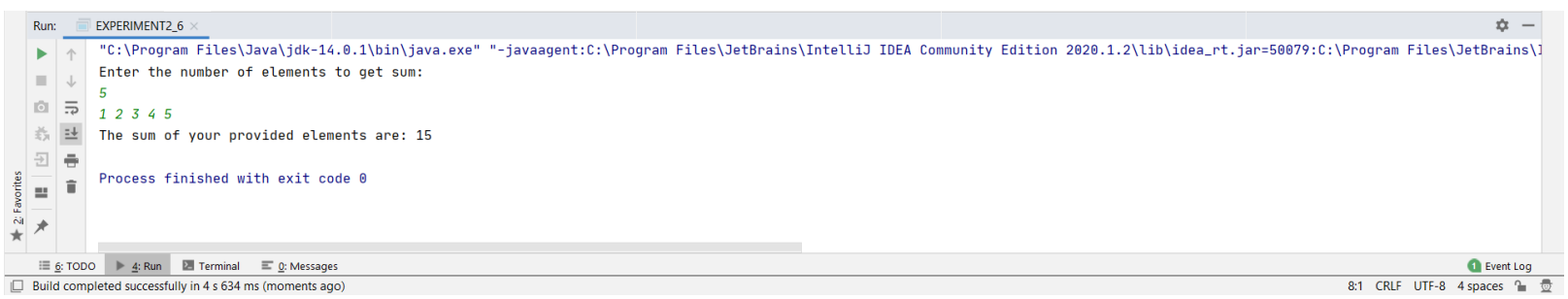
2.6 Write a program to take n integers from command line and print their sum of product (product of first number and last number added to product of second number and second last number and so on).

Code:

```
public class EXPERIMENT2_6 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter the number of elements to get sum:");

int n = sc.nextInt();
int [] arr = new int[n];
int sum = 0;

for(int k=0; k<arr.length; k++)
   arr[k] = sc.nextInt();

for(int i=0; i<arr.length/2; i++){

    int result = arr[i] + arr[arr.length-1-i];
    sum += result;
}

if(n%2 != 0){
        int middleIndex = ((n-1)/2);
        sum += arr[middleIndex];
    }
    System.out.println("The sum of your provided elements are: "+sum);

  }
}
```

Output:



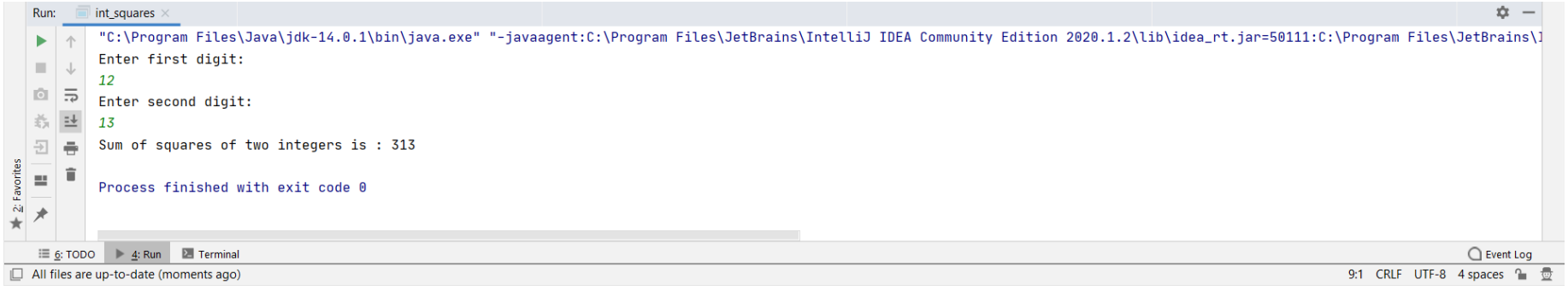
2.7 Consider any two integers. Write a program to print sum of their squares.

Code:

```
import java.util.Scanner;
public class int_squares {

   public static void main(String[] args) {

       Scanner sc = new Scanner(System.in);
       int a,b,c,d,e;
       System.out.println("Enter first digit:");

       a=sc.nextInt();
       b = a*a;
       System.out.println("Enter second digit:");

       c = sc.nextInt();
```

```
        d = c*c;
        e = b+d;
        System.out.println("Sum of squares of two integers is : " + e);
    }
}
```

Output:



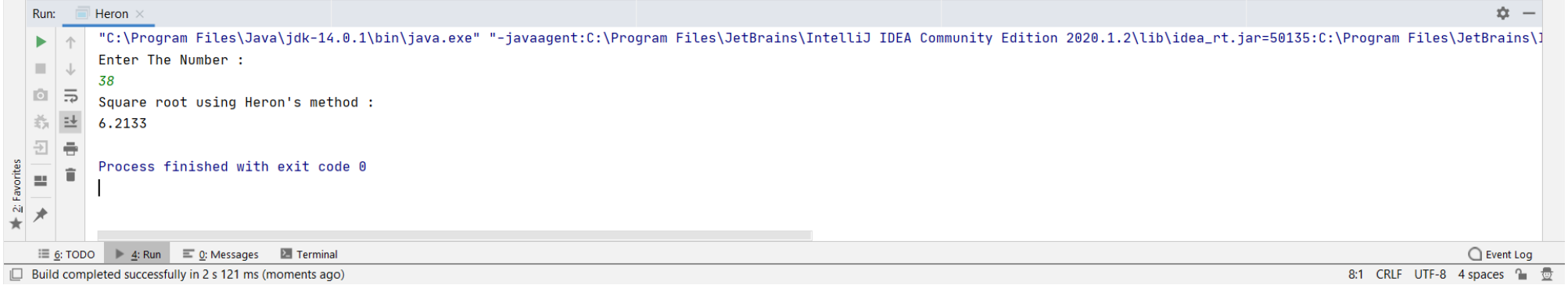
2.8 Write a program to find square root of a given positive integer using Heron's method to find square root.

Code:

```
import java.util.Scanner;
public class Heron {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter The Number : ");

        int a = scanner.nextInt();
        System.out.println((double)Math.round(heron(a) * 10000d) / 10000d);
    }

    public static int ClosetNumber(int a)   {
        int i;
    a = a - 1;
    while (a != 0) {

        for (i = 1; i * i <= a; i++)
        {

            if (i * i == a)
                return a;
        }
        a = a - 1;
    }
    return 0;
}
public static double heron(int x)
{
```

```
        double a, i;
        a = ClosetNumber(x);
        for (i = 0; i < 4; i++)
            a = 0.5 * (a + x / a);
        return a;
    }
}
```

Output :



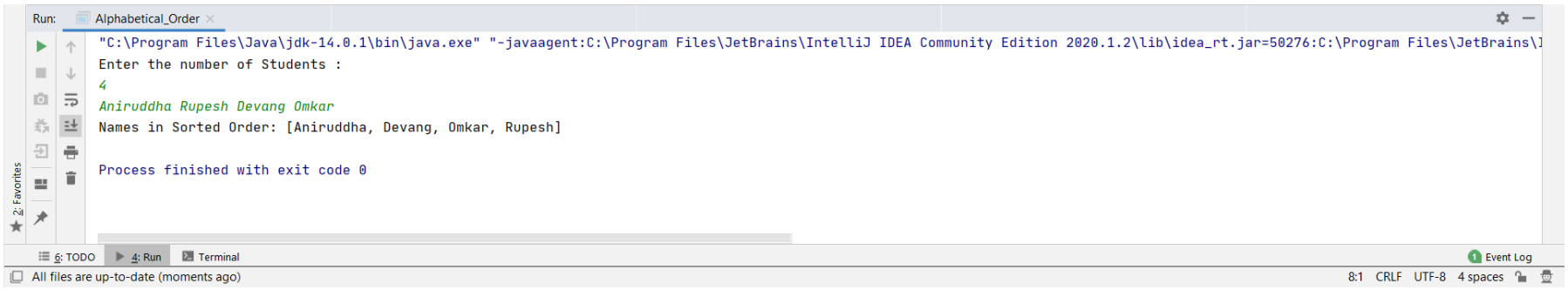
2.9 Write a program to sort and print the names of students taken from command line in alphabetical order.

Code :

```
import java.util.Scanner;
public class Alphabetical_Order
{
   public static void main(String[] args)
   {
      int n;
      String temp;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter number of names you want to enter:");
      n = s.nextInt();
      String names[] = new String[n];
      Scanner s1 = new Scanner(System.in);
      System.out.println("Enter all the names:");
      for(int i = 0; i < n; i++)
         names[i] = s1.nextLine();

      for (int i = 0; i < n; i++)     {
         for (int j = i + 1; j < n; j++)        {
            if (names[i].compareTo(names[j])>0)      {
               temp = names[i];
               names[i] = names[j];
               names[j] = temp;
            }
         }
      }
```

```
        System.out.print("Names in Sorted Order:");
        for (int i = 0; i < n - 1; i++)
        {
            System.out.print(names[i] + ",");
        }
        System.out.print(names[n - 1]);
    }
}
```

Output:



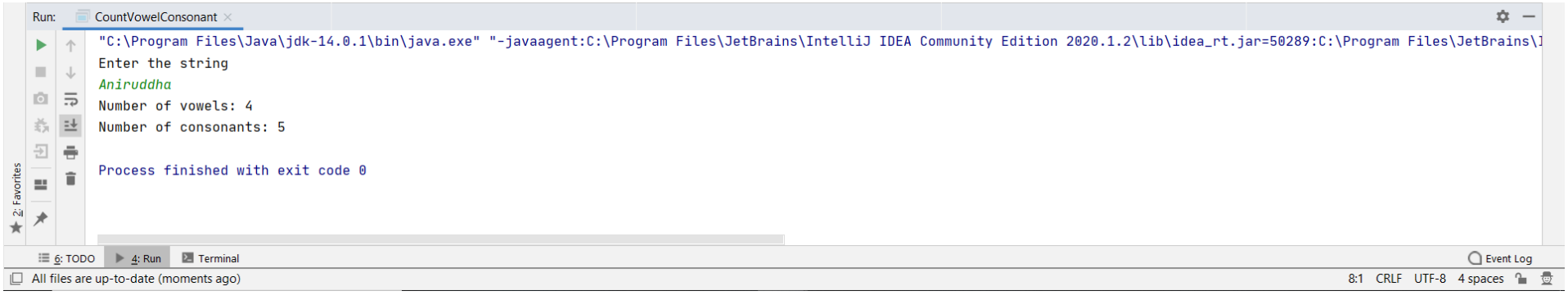
2.10 Write a program to print total numbers of vowels and consonants in a given string.

Code:

```
import java.util.Scanner;



public class CountVowelConsonant {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int vCount = 0, cCount = 0;

        System.out.println("Enter the string ");
        String str = sc.nextLine();

        str = str.toLowerCase();
        for(int i = 0; i < str.length(); i++) {

            if(str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i' || str.charAt(i) == 'o' ||
str.charAt(i) == 'u')
                vCount++;

            else if(str.charAt(i) >= 'a' && str.charAt(i)<='z')
                cCount++;

        }
```

```
        System.out.println("Number of vowels: " + vCount);
        System.out.println("Number of consonants: " + cCount);
    }
}
```

Output:



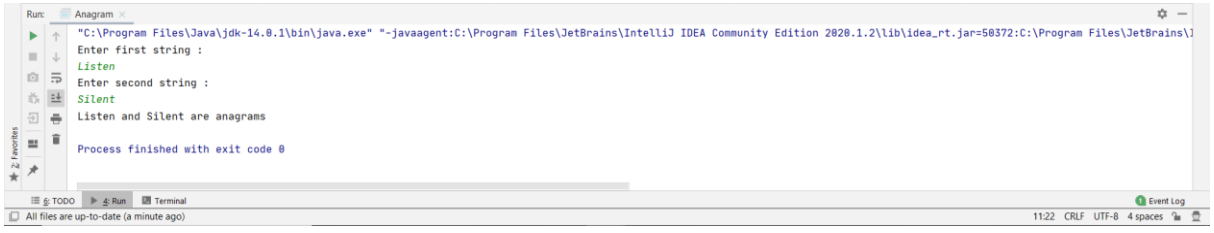
2.11 Given two English words, write a program to check if the first word is anagram of the second word. (An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. (Example: Anagram of TOM MARVOLO RIDDLE is I AM LORD VOLDEMORT.)

Code:

```
import java.util.Arrays;
import java.util.Scanner;

public class Anagram {

    static void areAnagram(String str1, String str2) {
        String s1 = str1.replaceAll("\\s", "");
        String s2 = str2.replaceAll("\\s", "");

        boolean status = true;
        int n1 = s1.length();
        int n2 = s2.length();

        if (n1 != n2)
            status = false;

        char[] ArrayS1 = s1.toLowerCase().toCharArray();
        char[] ArrayS2 = s2.toLowerCase().toCharArray();

        Arrays.sort(ArrayS1);
        Arrays.sort(ArrayS2);

        status = Arrays.equals(ArrayS1, ArrayS2);
        if (status)
            System.out.println(str1 + " and " + str2 + " are anagrams");
        else
```

```
        System.out.println(str1 + " and " + str2 + " are not anagrams");
    }

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter first string :");
        String str1 = in.nextLine();
        System.out.println("Enter second string :");
        String str2 = in.nextLine();
            areAnagram(str1, str2);
        }
    }
```

Output:



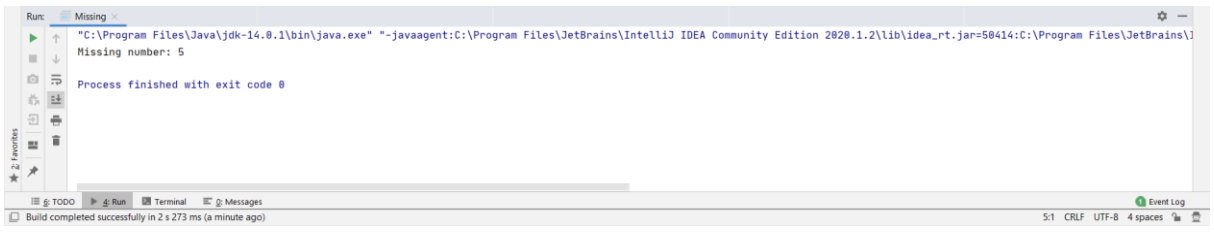
2.12 Write a program to print a missing number in a sorted integer array.

Code:

```
public class Missing {
    static int search(int arr1[], int size)
    {
        int a = 0, b = size - 1;
        int mid = 0;
        while ((b - a) > 1)
```

```
{
    mid = (a + b) / 2;
    if ((arr1[a] - a) != (arr1[mid] - mid))
        b = mid;
    else if ((arr1[b] - b) != (arr1[mid] - mid))
        a = mid;
}
return (arr1[mid] + 1);
    }
    public static void main (String[] args)
    {
        int array[] = { 1, 2, 3, 4, 6, 7, 8, 9, 10 };
        int size = array.length;
        System.out.println("Missing number: " + search(array, size));
    }


}
```

Output:

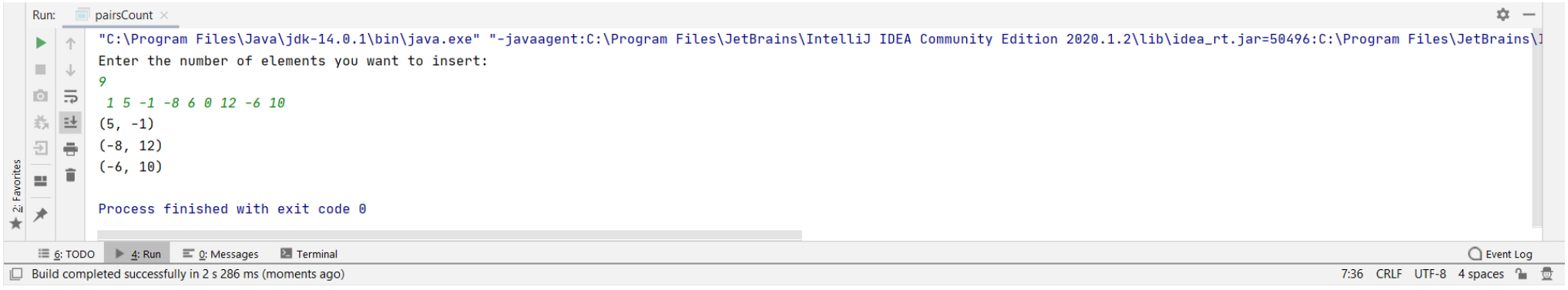


2.13 Write a program to find all the pairs of numbers on an integer array whose sum is equal to a given number.

Code:

```
import java.util.Scanner;


public class pairsCount {

   static void showPairs(int arr[], int n, int k) {

      for (int i = 0; i < n; i++)

         for (int j = i + 1; j < n; j++)

            if (arr[i] + arr[j] == k)

               System.out.println("(" + arr[i] + ", " + arr[j] + ")");
}


public static void main(String[] arg) {

   Scanner sc = new Scanner(System.in);

   System.out.println("Enter the number of elements you want to insert: ");

   int n = sc.nextInt();

   int arr[] = new int[n];

   for(int i=0; i<arr.length; i++){

      arr[i]=sc.nextInt();

      }

      int k = 4;

      showPairs(arr, n, k);

   }
}
```

Output:



Conclusion: We understood and performed various programs using Java.

Practical no. 3

FS19CO042

Aim:

3.1 Define the following classes/ interfaces with the help of above shortcuts:

1. Person(id, name, dateOfBirth, age, street, city, pin : default and parameterized constructors and setters and getters)

2. Department(id, name, dateOfEstablishment, headOfficeLocation, headId, numberOfEmployees : default and parameterized constructors and setters and getters)

3. Point(x, y, z : default and parameterized constructors and setters and getters)

4. Vehicle(registrationNumber, rcBookNumber, manufacturer, numberOfWheels, vehicleType, model, numberOfSeats : default and parameterized constructors and setters and getters)

5. Laptop(imeiNumber, processorName, processorSpeed, primaryMemoryType, primaryMemoryCapacity, secondaryStorageType, secondaryStorageCapaciry, screenResolution, screenType, isLED, listOfPorts, osInstalled : default and parameterized constructors and setters and getters)

6. interface Taxable(public int cost(), public intpercentGST())

3.2 Check whether feature of Encapsulation has been followed in 3.1. If not make necessary changes.

3.3 Define classes Car, Train and Truck with necessary member fields, constructors and methods. Make them extend class Vehicle.

3.4 Define a class Gadget with necessary member fields, constrictors and methods. Modify the class Laptop to extend the class Gadget.

3.5 In main method, declare a reference variable vehicle of class Vehicle and create an object of class Car which will be referenced by vehicle. Call getName() method on the object. (Hint: Reference Variable Casting)

3.6 Modify the classes Vehicle and Gadget implement the interface Taxable. Hence override respective methods.

3.7 Modify the classes Car and Laptop to override the implemented methods in 3.6.

3.8 Modify the class Gadget to add a data member gadgetCount such that its value will incremented as soon as a new object is initialized. Create 5 objects of the class Print its value after initializing each object.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects

- Instance
- Method
- Message Passing

Object − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Example

```
public class Dog {
   String breed;
   int age;
   String color;
   void barking() {
   }
   void hungry() {
   }
   void sleeping() {
   }
}
```

A class can contain any of the following variable types.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

Constructors

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor −

Example

```
public class Puppy {
   public Puppy() {
   }

   public Puppy(String name) {
      // This constructor has one parameter, name.
   }
}
```

Java also supports Singleton Classes where you would be able to create only one instance of a class.

Note − We have two different types of constructors. We are going to discuss constructors in detail in the subsequent chapters.

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface −

Example

Following is an example of an interface −

```
/* File name : NameOfInterface.java */

import java.lang.*;

// Any number of import statements

public interface NameOfInterface {

   // Any number of final, static fields

   // Any number of abstract method declarations\

}Interfaces have the following properties −
```

- An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example

```
/* File name : Animal.java */

interface Animal {

   public void eat();

   public void travel();

}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
/* File name : MammalInt.java */

public class MammalInt implements Animal {

   public void eat() {

      System.out.println("Mammal eats");

   }

   public void travel() {

      System.out.println("Mammal travels");

   }

   public int noOfLegs() {

      return 0;

   }

   public static void main(String args[]) {

      MammalInt m = new MammalInt();

      m.eat();

      m.travel();

   }

}
```

This will produce the following result −

Output

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces, there are several rules to be followed −

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules −

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.

- An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```
// Filename: Sports.java

public interface Sports {

   public void setHomeTeam(String name);

   public void setVisitingTeam(String name);

}



// Filename: Football.java

public interface Football extends Sports {

   public void homeTeamScored(int points);

   public void visitingTeamScored(int points);

   public void endOfQuarter(int quarter);

}

// Filename: Hockey.java

public interface Hockey extends Sports {

   public void homeGoalScored();

   public void visitingGoalScored();

   public void endOfPeriod(int period);

   public void overtimePeriod(int ot);

}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as −

Example

```
public interface Hockey extends Sports, Event
```

Code:

3.1 Define the following classes/ interfaces with the help of above shortcuts

1. Person(id, name, dateOfBirth, age, street, city, pin : default and parameterized constructors and setters and getters)

Code :

```
class Person {

    int dateOfBirth, age, id, pin;
    String name, street, city;

    Person() {

    }

    Person(int a, int b, int c, int d, String s, String s2, String s3) {
        this.dateOfBirth = c;
        this.age = b;
        this.id = a;
        this.pin = d;
        this.name = s;
        this.street = s2;
        this.city = s3;
    }

    public int getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(int dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public int getPin() {
        return pin;
    }

    public void setPin(int pin) {
        this.pin = pin;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    void setAge(int a) {
        age = a;
    }

}
```

2. Department(id, name, dateOfEstablishment, headOfficeLocation, headId, numberOfEmployees : default and parameterized constructors and setters and getters)

```
class Department {
    int id, headId, numberOfEmployees, dateOfEstablishment;
    String headOfficeLocation;
    String name;

    Department() {

    }

    public Department(int id, int headId, int numberOfEmployees, int dateOfEstablishment, String headOfficeLocation, String
name) {
        this.id = id;
        this.headId = headId;
        this.numberOfEmployees = numberOfEmployees;
```

```
        this.dateOfEstablishment = dateOfEstablishment;
        this.headOfficeLocation = headOfficeLocation;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getHeadId() {
        return headId;
    }

    public void setHeadId(int headId) {
        this.headId = headId;
    }

    public int getNumberOfEmployees() {
        return numberOfEmployees;
    }

    public void setNumberOfEmployees(int numberOfEmployees) {
        this.numberOfEmployees = numberOfEmployees;
    }

    public int getDateOfEstablishment() {
        return dateOfEstablishment;
    }

    public void setDateOfEstablishment(int dateOfEstablishment) {
        this.dateOfEstablishment = dateOfEstablishment;
    }

    public String getHeadOfficeLocation() {
        return headOfficeLocation;
    }

    public void setHeadOfficeLocation(String headOfficeLocation) {
        this.headOfficeLocation = headOfficeLocation;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


}
```

Point(x, y, z : default and parameterized constructors and setters and getters)

Code :

```
class Point {
    int x, y, z;

    Point() {
    }

    public Point(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getZ() {
        return z;
    }

    public void setZ(int z) {
        this.z = z;
    }
```

```
}
```

4. Vehicle(registrationNumber, rcBookNumber, manufacturer, numberOfWheels, vehicleType, model, numberOfSeats : default and parameterized constructors and setters and getters)

Code :

```
class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }
public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getRegistrationNumber() {
    return registrationNumber;
}

public void setRegistrationNumber(int registrationNumber) {
    this.registrationNumber = registrationNumber;
}

public int getRcBookNumber() {
    return rcBookNumber;
}

public void setRcBookNumber(int rcBookNumber) {
    this.rcBookNumber = rcBookNumber;
}

public int getManufacturer() {
    return manufacturer;
}
public void setManufacturer(int manufacturer) {
    this.manufacturer = manufacturer;
}

public int getNumberOfWheels() {
    return numberOfWheels;
}

public void setNumberOfWheels(int numberOfWheels) {
    this.numberOfWheels = numberOfWheels;
}

public int getNumberOfSeats() {
    return numberOfSeats;
}

public void setNumberOfSeats(int numberOfSeats) {
    this.numberOfSeats = numberOfSeats;
}

public String getVehicleType() {
    return vehicleType;
}

public void setVehicleType(String vehicleType) {
    this.vehicleType = vehicleType;
}

public String getModel() {
    return model;
```

```
    }

    public void setModel(String model) {
        this.model = model;
    }


    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

5. Laptop(imeiNumber, processorName, processorSpeed, primaryMemoryType, primaryMemoryCapacity, secondaryStorageType, secondaryStorageCapaciry, screenResolution, screenType, isLED, listOfPorts, osInstalled : default and parameterized constructors and setters and getters)

Code :

```
class Laptop extends Gadget {
    int imeiNumber;
    String processorName, primaryMemoryType, secondaryStorageType, screenType;
    boolean isLED, osInstalled;
    float processorSpeed, primaryMemoryCapacity, secondaryStorageCapaciry, screenResolution;
    String listOfPorts;
    int cost;

    Laptop() {

    }

    public Laptop(int imeiNumber, String processorName, String primaryMemoryType, String secondaryStorageType, String
screenType, boolean isLED, boolean osInstalled, float processorSpeed, float primaryMemoryCapacity, float
secondaryStorageCapaciry, float screenResolution, String listOfPorts) {
        this.imeiNumber = imeiNumber;
        this.processorName = processorName;
        this.primaryMemoryType = primaryMemoryType;
        this.secondaryStorageType = secondaryStorageType;
        this.screenType = screenType;
        this.isLED = isLED;
        this.osInstalled = osInstalled;
        this.processorSpeed = processorSpeed;
        this.primaryMemoryCapacity = primaryMemoryCapacity;
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
        this.screenResolution = screenResolution;
        this.listOfPorts = listOfPorts;

    }

    public boolean isLED() {
        return isLED;
    }

    public void setLED(boolean LED) {
        isLED = LED;
    }

    public boolean isOsInstalled() {
        return osInstalled;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public int getImeiNumber() {
        return imeiNumber;
    }

    public void setImeiNumber(int imeiNumber) {
        this.imeiNumber = imeiNumber;
    }

    public String getProcessorName() {
        return processorName;
    }

    public void setProcessorName(String processorName) {
        this.processorName = processorName;
    }
```

```
public String getPrimaryMemoryType() {
    return primaryMemoryType;
}

public void setPrimaryMemoryType(String primaryMemoryType) {
    this.primaryMemoryType = primaryMemoryType;
}

public String getSecondaryStorageType() {
    return secondaryStorageType;
}

public void setSecondaryStorageType(String secondaryStorageType) {
    this.secondaryStorageType = secondaryStorageType;
}

public String getScreenType() {
    return screenType;
}

public void setScreenType(String screenType) {
    this.screenType = screenType;
}

public boolean getIsLED() {
    return isLED;
}

public void setIsLED(Boolean isLED) {
    this.isLED = isLED;
}
```

```
public boolean getOsInstalled() {
    return osInstalled;
}

public void setOsInstalled(boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public void setOsInstalled(Boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public float getProcessorSpeed() {
    return processorSpeed;
}

public void setProcessorSpeed(float processorSpeed) {
    this.processorSpeed = processorSpeed;
}

public float getPrimaryMemoryCapacity() {
    return primaryMemoryCapacity;
}

public void setPrimaryMemoryCapacity(float primaryMemoryCapacity) {
    this.primaryMemoryCapacity = primaryMemoryCapacity;
}

public float getSecondaryStorageCapaciry() {
    return secondaryStorageCapaciry;
}

public void setSecondaryStorageCapaciry(float secondaryStorageCapaciry) {
    this.secondaryStorageCapaciry = secondaryStorageCapaciry;
}
```

```
    public float getScreenResolution() {
        return screenResolution;
    }

    public void setScreenResolution(float screenResolution) {
        this.screenResolution = screenResolution;
    }

    public String getListOfPorts() {
        return listOfPorts;
    }

    public void setListOfPorts(String listOfPorts) {
        this.listOfPorts = listOfPorts;
    }

    void print() {
        System.out.println("emi no is " + getImeiNumber() + "\n Processor name is: " + getProcessorName() + "\n led :" +
getIsLED() + "\n Ports are :" + getListOfPorts() + "\n OS :" + getOsInstalled() + "\n Meomory Capacity :" +
getPrimaryMemoryCapacity());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
```

```
        return percentGST;
    }

}
```

6. interface Taxable(public int cost(), public intpercentGST())

Code :

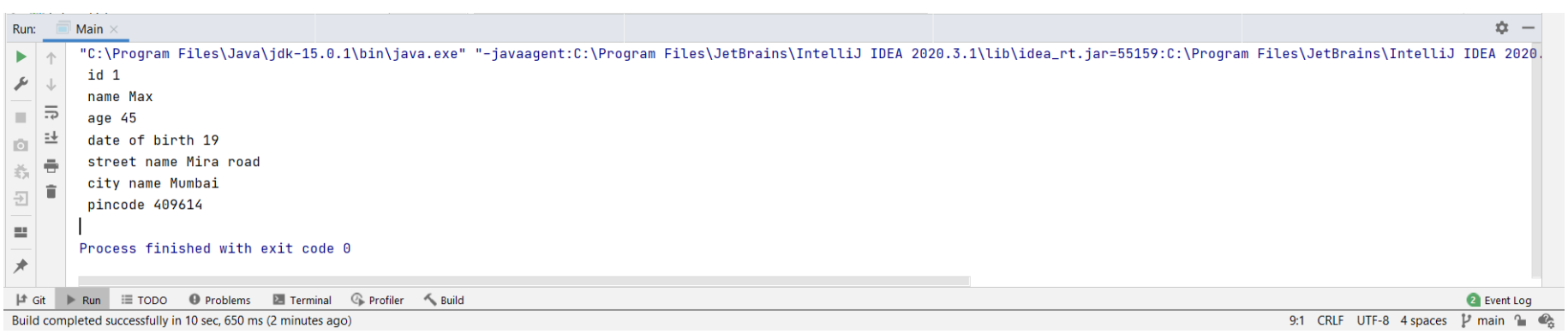
```
interface Taxable {
    int cost();

    int percentGST();
}
```

3.2 Check whether feature of Encapsulation has been followed in 3.1. If not make necessary changes.

Code :

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setAge(45);
        person.setId(01);
        person.setName("Max");
        person.setDateOfBirth(19);
        person.setStreet("Mira road");
        person.setCity("Mumbai");
        person.setPin(409614);
        System.out.println(" id " + person.getId() + "\n name " + person.getName() + "\n age " + person.getAge() + "\n date of
birth " + person.getDateOfBirth() + "\n street name " + person.getStreet() + "\n city name " + person.getCity() + "\n pincode "
+ person.getPin());
    }
}
```

Output :



3.3 Define classes Car, Train and Truck with necessary member fields, constructors and methods. Make them extend class Vehicle.

Code :

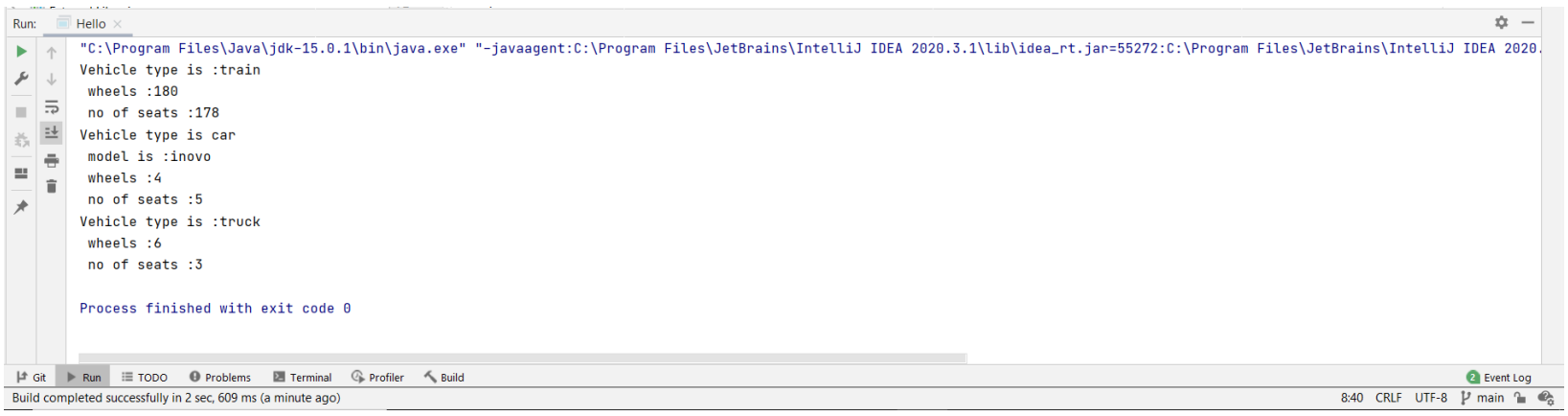
```
class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }
```

```
    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}



class Train extends Vehicle {
    void show() {
        System.out.println("vehicle type is :" + getVehicleType() + "\n wheels :" + getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }


}



class Truck extends Vehicle {


    void show() {
        System.out.println("vehicle type is :" + getVehicleType() + "\n wheels :" + getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }
}
```

Main method :

```
public static void main(String[] args) {

    Train h = new Train();
    h.setVehicleType("train");
    h.setNumberOfSeats(178);
    h.setNumberOfWheels(180);
    h.setRegistrationNumber(90764);
    h.show();

    Car c = new Car();
    c.setVehicleType("car");
    c.setModel("inovo");
    c.setNumberOfSeats(5);
    c.setNumberOfWheels(4);
    c.setRegistrationNumber(90671);
    c.show();

    Truck t = new Truck();
    t.setVehicleType("truck");
    t.setNumberOfSeats(3);
    t.setNumberOfWheels(6);
    t.show();
}
```

Output :



3.4 Define a class Gadget with necessary member fields, constrictors and methods. Modify the class Laptop to extend the class Gadget.

Code :

```
public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
```

```
    this.gadgetName = gadgetName;
}

void disp() {
    System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
}

public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }

}
```

Main method :

```
public static void main(String args[]) {
    Laptop l = new Laptop();
    l.setGadgetName("Laptop");
    l.setImeiNumber(2345);
    l.setIsLED(true);
    l.setListOfPorts("2 USB Port,1 Charging port,1 Pendrive Port");
    l.setOsInstalled(true);
    l.setPrimaryMemoryCapacity(1500);
    l.setProcessorName("intel core i5");
    l.Show();
    l.print();
}
```

Output :

3.5 In main method, declare a reference variable vehicle of class Vehicle and create an object of class Car which will be referenced by vehicle. Call getName() method on the object. (Hint: Reference Variable Casting)

Code :

```
class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
```

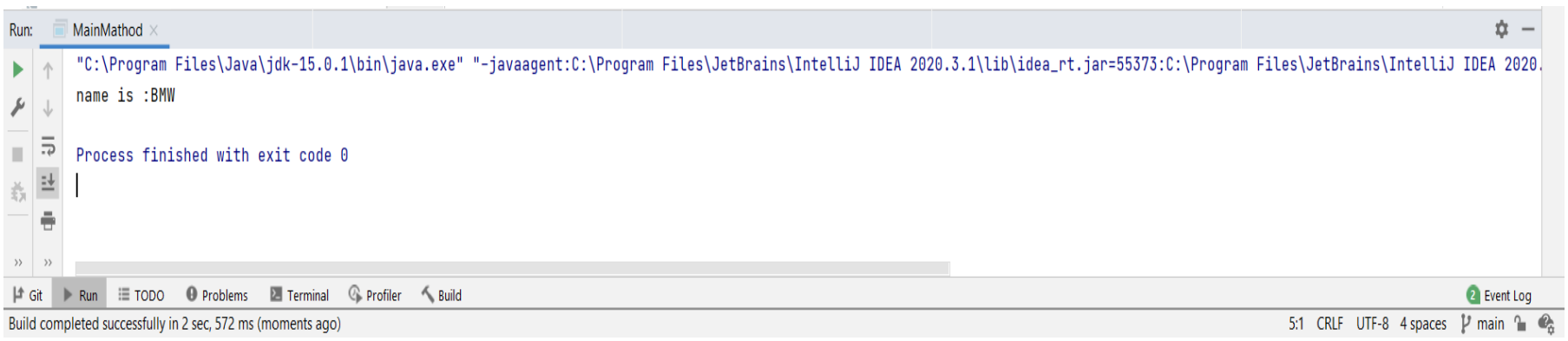
```
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getRegistrationNumber() {
        return registrationNumber;
    }

    public void setRegistrationNumber(int registrationNumber) {
        this.registrationNumber = registrationNumber;
    }

    public int getRcBookNumber() {
        return rcBookNumber;
    }

    public void setRcBookNumber(int rcBookNumber) {
        this.rcBookNumber = rcBookNumber;
    }

    public int getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(int manufacturer) {
        this.manufacturer = manufacturer;
    }

    public int getNumberOfWheels() {
        return numberOfWheels;
    }

    public void setNumberOfWheels(int numberOfWheels) {
        this.numberOfWheels = numberOfWheels;
    }

    public int getNumberOfSeats() {
        return numberOfSeats;
    }

    public void setNumberOfSeats(int numberOfSeats) {
        this.numberOfSeats = numberOfSeats;
    }

    public String getVehicleType() {
        return vehicleType;
    }

    public void setVehicleType(String vehicleType) {
        this.vehicleType = vehicleType;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }


    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

```
class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("Vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}
```

Main method :

```
public static void main(String [] args)
{
    Vehicle vehicle;
    Car c1=new Car();
    c1.setName("BMW");
    c1.disp();

}
```

Output :



3.6 Modify the classes Vehicle and Gadget implement the interface Taxable. Hence override respective methods.

Code :

```
interface Taxable {
    int cost();

    int percentGST();
}

public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
        this.gadgetName = gadgetName;
    }

    void disp() {
        System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
    }
```

```
public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
}

public int percentGST() {
    float a = 0.18f;
    int percentGST = (int) (getCost() + (getCost() * a));
    return percentGST;
    }

}


class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public String getName() {
        return name;
}

public void setName(String name) {
    this.name = name;
}

public int getRegistrationNumber() {
    return registrationNumber;
}

public void setRegistrationNumber(int registrationNumber) {
    this.registrationNumber = registrationNumber;
}

public int getRcBookNumber() {
    return rcBookNumber;
}

public void setRcBookNumber(int rcBookNumber) {
    this.rcBookNumber = rcBookNumber;
}

public int getManufacturer() {
    return manufacturer;
}

public void setManufacturer(int manufacturer) {
    this.manufacturer = manufacturer;
}

public int getNumberOfWheels() {
```

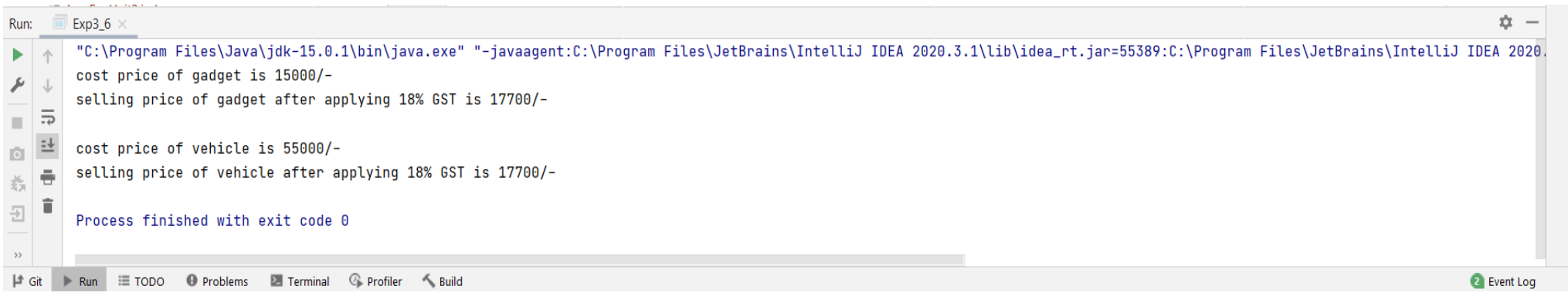
```
    return numberOfWheels;
}

public void setNumberOfWheels(int numberOfWheels) {
    this.numberOfWheels = numberOfWheels;
}

public int getNumberOfSeats() {
    return numberOfSeats;
}

public void setNumberOfSeats(int numberOfSeats) {
    this.numberOfSeats = numberOfSeats;
}

public String getVehicleType() {
    return vehicleType;
}

public void setVehicleType(String vehicleType) {
    this.vehicleType = vehicleType;
}

public String getModel() {
    return model;
}

public void setModel(String model) {
    this.model = model;
}
```

```
    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

Main method :

```
public static void main(String args[]) {
    Gadget g = new Gadget();
    g.setCost(15000);
    Vehicle v = new Vehicle();
    v.setCost(55000);
    System.out.println("cost price of gadget is " + g.cost() + "/-");
    System.out.println("selling price of gadget after applying 18% GST is " + g.percentGST() + "/-");
    System.out.println();
    System.out.println("cost price of vehicle is " + v.cost() + "/-");
    System.out.println("selling price of vehicle after applying 18% GST is " + g.percentGST() + "/-");

}
```

Output :



3.7 Modify the classes Car and Laptop to override the implemented methods in 3.6.

Code :

```
class Laptop extends Gadget {
    int imeiNumber;
    String processorName, primaryMemoryType, secondaryStorageType, screenType;
    boolean isLED, osInstalled;
    float processorSpeed, primaryMemoryCapacity, secondaryStorageCapaciry, screenResolution;
    String listOfPorts;
    int cost;

    Laptop() {

    }

    public Laptop(int imeiNumber, String processorName, String primaryMemoryType, String secondaryStorageType, String
```

```
screenType, boolean isLED, boolean osInstalled, float processorSpeed, float primaryMemoryCapacity, float
secondaryStorageCapaciry, float screenResolution, String listOfPorts) {
        this.imeiNumber = imeiNumber;
        this.processorName = processorName;
        this.primaryMemoryType = primaryMemoryType;
        this.secondaryStorageType = secondaryStorageType;
        this.screenType = screenType;
        this.isLED = isLED;
        this.osInstalled = osInstalled;
        this.processorSpeed = processorSpeed;
        this.primaryMemoryCapacity = primaryMemoryCapacity;
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
        this.screenResolution = screenResolution;
        this.listOfPorts = listOfPorts;

    }

    public boolean isLED() {
        return isLED;
    }

    public void setLED(boolean LED) {
        isLED = LED;
    }

    public boolean isOsInstalled() {
        return osInstalled;
    }

    public int getCost() {
        return cost;
    }

public void setCost(int cost) {
    this.cost = cost;
}

public int getImeiNumber() {
    return imeiNumber;
}

public void setImeiNumber(int imeiNumber) {
    this.imeiNumber = imeiNumber;
}

public String getProcessorName() {
    return processorName;
}

public void setProcessorName(String processorName) {
    this.processorName = processorName;
}

public String getPrimaryMemoryType() {
    return primaryMemoryType;
}

public void setPrimaryMemoryType(String primaryMemoryType) {
    this.primaryMemoryType = primaryMemoryType;
}

public String getSecondaryStorageType() {
    return secondaryStorageType;
}

public void setSecondaryStorageType(String secondaryStorageType) {
    this.secondaryStorageType = secondaryStorageType;
}

public String getScreenType() {
    return screenType;
}

public void setScreenType(String screenType) {
    this.screenType = screenType;
}

public boolean getIsLED() {
    return isLED;
}

public void setIsLED(Boolean isLED) {
    this.isLED = isLED;
}

public boolean getOsInstalled() {
    return osInstalled;
}

public void setOsInstalled(boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public void setOsInstalled(Boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public float getProcessorSpeed() {
```

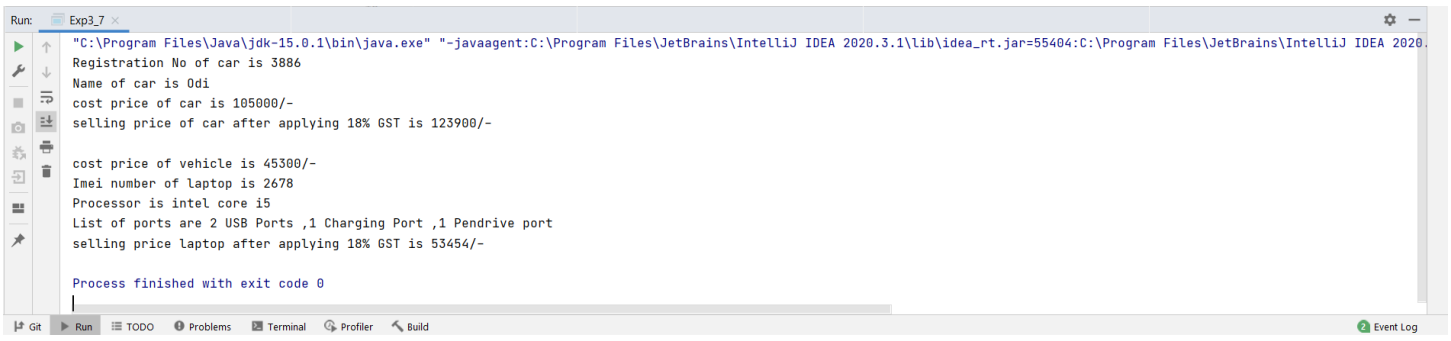
```
        return processorSpeed;
    }

    public void setProcessorSpeed(float processorSpeed) {
        this.processorSpeed = processorSpeed;
    }

    public float getPrimaryMemoryCapacity() {
        return primaryMemoryCapacity;
    }

    public void setPrimaryMemoryCapacity(float primaryMemoryCapacity) {
        this.primaryMemoryCapacity = primaryMemoryCapacity;
    }

    public float getSecondaryStorageCapaciry() {
        return secondaryStorageCapaciry;
    }

    public void setSecondaryStorageCapaciry(float secondaryStorageCapaciry) {
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
    }

    public float getScreenResolution() {
        return screenResolution;
    }

    public void setScreenResolution(float screenResolution) {
        this.screenResolution = screenResolution;
    }

    public String getListOfPorts() {
        return listOfPorts;
    }

    public void setListOfPorts(String listOfPorts) {
        this.listOfPorts = listOfPorts;
    }

    void print() {
        System.out.println("emi no is " + getImeiNumber() + "\n Processor name is: " + getProcessorName() + "\n led :" +
getIsLED() + "\n Ports are :" + getListOfPorts() + "\n OS :" + getOsInstalled() + "\n Meomory Capacity :" +
getPrimaryMemoryCapacity());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }

}


class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("Vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}
```

Main method :

```
public static void main(String args[])
{
```

```
    Car c=new Car();
    c.setName("Odi");
    c.setCost(105000);
    c.setRegistrationNumber(07456);
    Laptop l=new Laptop();
    l.setCost(45300);
    l.setProcessorName("intel core i5");
    l.setImeiNumber(2678);
    l.setListOfPorts("2 USB Ports ,1 Charging Port ,1 Pendrive port ");
    System.out.println("Registration No of car is "+c.getRegistrationNumber()+"\nName of car is "+ c.getName()+"\ncost price of
car is "+c.cost()+"/-");
    System.out.println("selling price of car after applying 18% GST is "+c.percentGST()+"/-");
    System.out.println();
    System.out.println("cost price of vehicle is "+l.cost()+"/-");
    System.out.println("Imei number of laptop is "+l.getImeiNumber()+"\nProcessor is "+l.getProcessorName()+"\nList of ports are
"+l.getListOfPorts()+"\nselling price laptop after applying 18% GST is "+l.percentGST()+"/-");


}
```

Output :



3.8 Modify the class Gadget to add a data member gadgetCount such that its value will incremented as soon as a new object is initialized. Create 5 objects of the class Print its value after initializing each object.

Code :

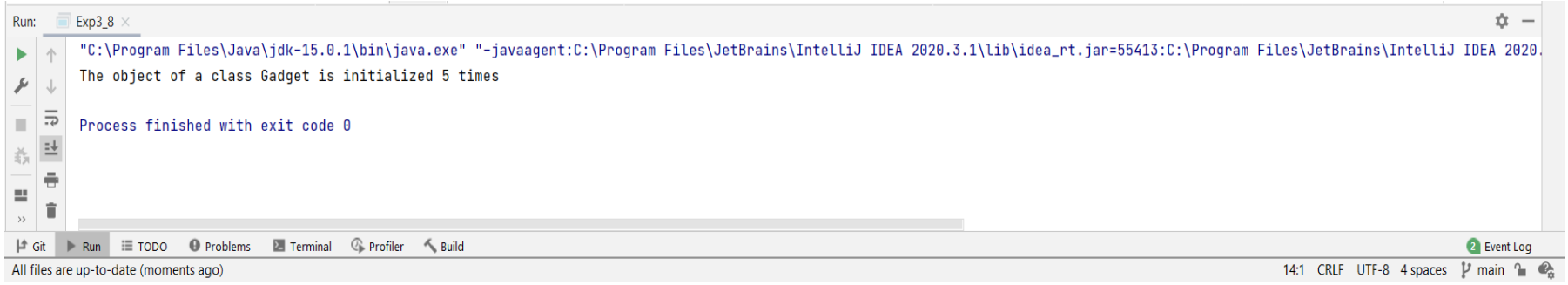
```
public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
        this.gadgetName = gadgetName;
    }

    void disp() {
        System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
    }

public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
}

public int percentGST() {
    float a = 0.18f;
    int percentGST = (int) (getCost() + (getCost() * a));
    return percentGST;
}
```

```
}
```

Main method :

```
   public static void main(String args[])
    {
        Gadget g=new Gadget();
        Gadget g1=new Gadget();
        Gadget g2=new Gadget();
        Gadget g3=new Gadget();
        Gadget g4=new Gadget();
        g.disp();
    }
}
```

Output :



Conclusion: Thus, we understood and executed various programs using classes, interfaces, etc. and explored various concepts related to these topics.

Practical no. 4

FS19CO042

Aim:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
  }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
```

```
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Code:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

Code :

```
package com.gpm.complex;

public interface Complex {
    void realPart();

    void imgPart();

    void magnitude();

    void argument();

    default float plus(float a, float b) {
        return a + b;
    }

    default float minus(float a, float b) {
        return a - b;
    }

    default float into(float a, float b) {
        return a * b;
    }

    default float divideBy(float a, float b) {
        return a / b;
    }

}
```

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

```
package com.gpm.complex;

public class CartesianComplex implements Complex {
    CartesianComplex real;
    CartesianComplex img;

    @Override
    public String toString() {
        return "CartesianComplex";
    }

    @Override
    public void realPart() {

    }

    @Override
    public void imgPart() {

    }
```

```
    @Override
    public void magnitude() {

    }

    @Override
    public void argument() {

    }

}
```

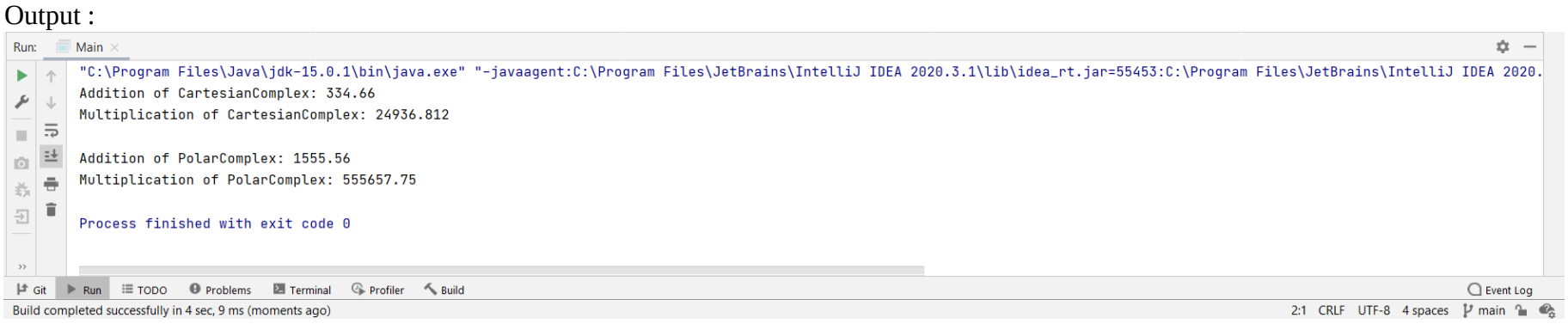
```
package com.gpm.complex;

public class PolarComplex implements Complex {
    PolarComplex r;
    PolarComplex theta;

    @Override
    public String toString() {
        return "PolarComplex";
    }

    @Override
    public void realPart() {

    }

    @Override
    public void imgPart() {

    }

    @Override
    public void magnitude() {

    }

    @Override
    public void argument() {

    }
}
```

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

Code :

```
package com.gpm.complex;

public class Main {

    public static void main(String[] args) {

        CartesianComplex cartesianComplex = new CartesianComplex();
        PolarComplex polarComplex = new PolarComplex();

        System.out.println("Addition of CartesianComplex: "+cartesianComplex.plus(111.99f, 222.67f));
        System.out.println("Multiplication of CartesianComplex: "+cartesianComplex.into(111.99f, 222.67f)+"\n");

        System.out.println("Addition of PolarComplex: "+polarComplex.plus(555.78f, 999.78f));
        System.out.println("Multiplication of PolarComplex: "+polarComplex.into(555.78f, 999.78f));

    }
}
```

Output :



4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked.

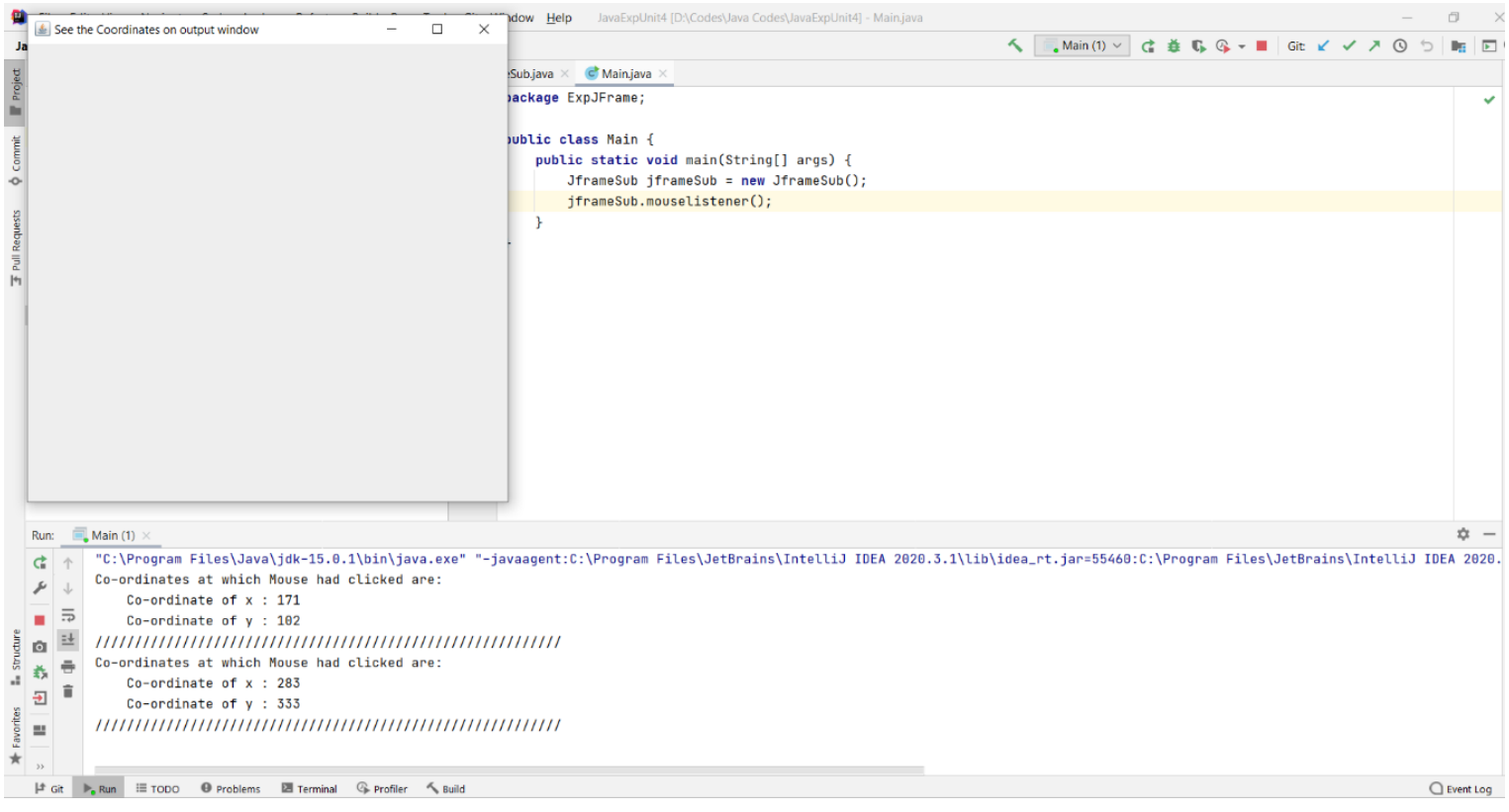
Code :

```
package ExpJFrame;

import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class JframeSub extends JFrame {

    public JframeSub() {

    }

    public void mouselistener() {

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                int x = e.getX();
                int y = e.getY();
                System.out.println("Co-ordinates at which Mouse had clicked are: \n" +
                        "\tCo-ordinate of x : " + x +
                        "\n\tCo-ordinate of y : " + y);
                System.out.println("////////////////////////////////////////////////////////////");
            }
        });

        setTitle("See the Coordinates on output window");
        setLayout(null);
        setVisible(true);
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }
}
```

Main method :

```
package ExpJFrame;
public class Main {
    public static void main(String[] args) {
        JframeSub jframeSub = new JframeSub();
        jframeSub.mouselistener();
    }
}
```

Output :



Conclusion: Thus, we understood and executed programs regarding interfaces and swing framework.

Practical no. 5

FS19CO042

Aim: Using Stream API implement following programs.

- 5.1 Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).
- 5.2 Write a method which takes a list of words as an argument, groups the words by their lengths and returns the groupings in the form of Map>. (The keys in the map are the lengths and the values are the lists of words of that length.)
- 5.3 Given a List<List> write a program to convert it into a List. (Hint: Use flatMap method in Stream interface)
- 5.4 Given: class Album{ public final String name; public final int yearOfRelease; public final List tracks; }
  class Track{ public final int rating; }
  a) Write a method which takes a list of albums as an argument and returns a list of names of all albums sorted by the year of release.
  b) Write a method which takes a list of albums as an argument and returns a list of names of all albums containing at least one track having rating more than four. The returned list should be sorted by the year of release.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Stream In Java

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

- A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Streams don't change the original data structure, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Different Operations On Streams-

Intermediate Operations:

1. **map:** The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.

```
List number = Arrays.asList(2,3,4,5);
List square = number.stream().map(x->x*x).collect(Collectors.toList());
```

2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
```

3. **sorted:** The sorted method is used to sort the stream.

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().sorted().collect(Collectors.toList());
```

Terminal Operations:

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.

```
List number = Arrays.asList(2,3,4,5,3);
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

2. **forEach:** The forEach method is used to iterate through every element of the stream.

```
List number = Arrays.asList(2,3,4,5);
number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

3. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.

   The reduce method takes a BinaryOperator as a parameter.

```
List number = Arrays.asList(2,3,4,5);
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);
```

   Here ans variable is assigned 0 as the initial value and i is added to it .

Code:

- **5.1 Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).**

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;


public class  exp5_1 {

static class Student {
   String name = "";
   public int roll = 0;
   public int marks = 0;
   public Student(String name, int roll, int marks) {
      this.name = name;
      this.roll = roll;
      this.marks = marks;
   }
}

public static <T extends  Number> long  evenNumbers(List<T> list) {
   Stream<T> stream = list.stream();
   return stream.filter(number -> number.doubleValue() % 2 != 0).count();
```

```
}

public static <T extends Student> long  numberOfPassedStudents(List<? extends Student> list) {
   Stream<T> stream = (Stream<T>) list.stream();
   return stream.filter(student -> student.marks >= 35).count();

}

public static void main(String[] args) {

   Student s1 = new Student("Roy", 43, 60);
   Student s2 = new Student("Niel", 44, 49);
   Student s3 = new Student("Leo", 30, 75);
   Student s4 = new Student("lisa", 35, 30);
      Student s5 = new Student("Russ", 40, 28);

      List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      List<Student> list1 = Arrays.asList(s1, s2, s3, s4, s5);

      long evenNumbers = evenNumbers(list) ;
      long numberOfPassedStudents = numberOfPassedStudents(list1) ;

      System.out.println("There are "+ evenNumbers +" even numbers.");
      System.out.println(numberOfPassedStudents+" students have passed the exam.");
   }
}
```

Output:

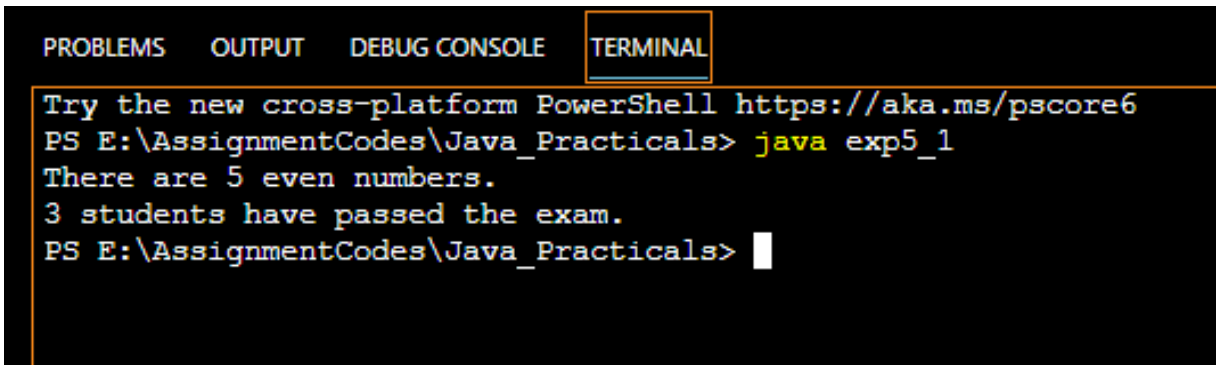
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS E:\AssignmentCodes\Java_Practicals> java exp5_1
There are 5 even numbers.
3 students have passed the exam.
PS E:\AssignmentCodes\Java_Practicals>
```

- **5.2 Write a method which takes a list of words as an argument, groups the words by their lengths and returns the groupings in the form of Map>. (The keys in the map are the lengths and the values are the lists of words of that length.)**

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

// main class and method
public class exp5_2{

   // main Driver
   private static Stream<String> stream;
```

```
    public static void main(String[] args)
    {

        // create a list

List<String> strings = Arrays.asList("this", "is", "a", "long", "list", "of",
        "strings", "to", "use", "as", "a", "trial");

stream = strings.stream();
Map<Integer, List<String>> lengthMap = stream.collect(Collectors.groupingBy(String::length));

lengthMap.forEach((k,v) -> System.out.printf("%d: %s%n", k, v));
    }
}
```

Output:



- **5.3 Given a List<List> write a program to convert it into a List. (Hint: Use flatMap method in Stream interface)**

```
import java.util.*;

import java.util.stream.*;

class exp5_3 {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList();

list1.add("Government");

list1.add("Polytechnic");

list1.add("Mumbai");


ArrayList<String> list2 = new ArrayList();

list2.add("A.Y. Jung Marg");

list2.add("Kherwadi");

list2.add("Bandra (E)");


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

listOflist.add(list1);

listOflist.add(list2);
```

```
        System.out.println("LIST1 : " + list1);
        System.out.println("LIST2 : " + list2);
        System.out.println("LIST<LIST<String>> :" + listOflist);

        ArrayList<String> result = new ArrayList();
        listOflist.forEach(result::addAll);

        System.out.println("RESULT : " + result);
    }
}
```

```
PROBLEMS 453   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: Code
PS E:\AssignmentCodes\Java_Practicals> java exp5_3
LIST1 : [Government, Polytechnic, Mumbai]
LIST2 : [A.Y. Jung Marg, Kherwadi, Bandra (E)]
LIST<LIST<String>> :[[Government, Polytechnic, Mumbai], [A.Y. Jung Marg, Kherwadi, Bandra (E)]]
RESULT : [Government, Polytechnic, Mumbai, A.Y. Jung Marg, Kherwadi, Bandra (E)]
PS E:\AssignmentCodes\Java_Practicals>
```

- **5.4 Given: class Album{ public final String name; public final int yearOfRelease; public final List tracks;**

  class Track{ public final int rating; }

  a) Write a method which takes a list of albums as an argument and returns a list of names of all albums sorted by the year of release.

  b) Write a method which takes a list of albums as an argument and returns a list of names of all albums containing at least one track having rating more than four. The returned list should be sorted by the year of release.

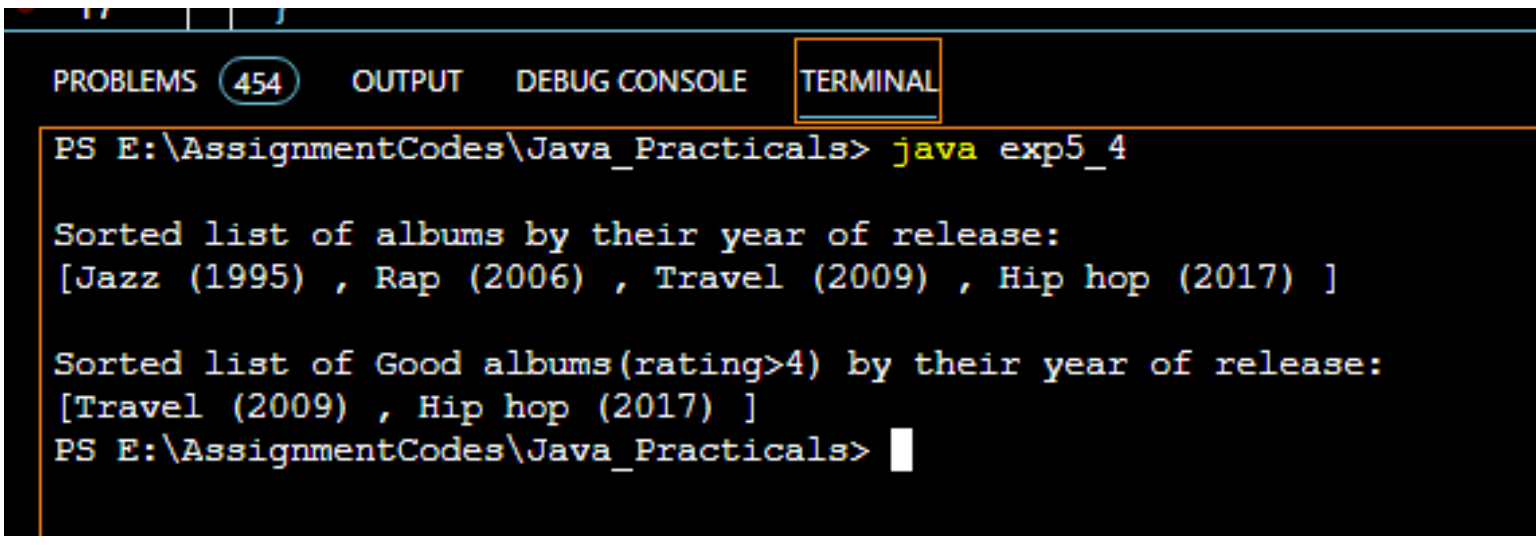
```
import java.util.*;
import java.util.stream.*;
public class exp5_4{
    static class Track{
        public final String name;
        public final int rating;
        public Track(String name, int rating){
            this.name = name;
            this.rating = rating;
        }
```

```
    public String toString(){
        return this.name + " (" + this.rating + ") ";
    }
}
static class Album{
    public final String name;
    private final List<Track> tracks;
    private int yearOfRelease;
    public int getYear(){
        return yearOfRelease;
    }
public List<Track> getTracks(){
    return tracks;
}
public int maxRating(){
    Track maxTrack = tracks.stream().reduce(new Track("temp",0), (maxTrackYet, currTrack) -> {
        if(maxTrackYet.rating < currTrack.rating)
            return currTrack;
        return maxTrackYet;
    });
    return maxTrack.rating;
}
public Album(String name, List<Track> trackList, int yearOfRelease){
    this.name = name;
        this.yearOfRelease = yearOfRelease;
        this.tracks = trackList;
    }
    public String toString(){
        return this.name+" ("+this.yearOfRelease+") ";
    }
}
static List<String>  sortAlbumsByYear(List<Album> albums){
    Stream albumStream = albums.stream();
    Object sorted[] = albumStream.sorted(Comparator.comparingInt(Album::getYear)).toArray();
```

```
    List<String> sortedList = new ArrayList<>();
    for(Object obj : sorted)
        sortedList.add(String.valueOf(obj));
    return sortedList;
}

static public List<String> filterGoodAlbums(List<Album> albums){
    List<String> goodAlbums = new ArrayList<>();
    albums.stream().forEach(album -> {
        if(album.maxRating() >4)
            goodAlbums.add(album.toString());
    });
     return goodAlbums;
}

public  static void main(String[] args) {
    System.out.println();
    // Preparing albums and tracks
    String[] travelNames = {"Ve maahi", "Duniya", "Bolna", "Kabira", "Vaaste"};
    int[] travelRatings= {5,6,4,8,5};
    List<Track> travelSongs = new  ArrayList<>();
    for(int i=0; i<travelNames.length; i++)
        travelSongs.add(new Track(travelNames[i], travelRatings[i]));
    Album travelAlbum = new Album("Travel",travelSongs, 2009);

String[] rapNames = {"Mirchi", "ChalBombay", "Kohinoor"};
int[] rapRatings = {1, 3,2};
List<Track> rapSongs = new  ArrayList<>();
for(int i=0; i<rapNames.length; i++)
        rapSongs.add(new Track(rapNames[i], rapRatings[i]));
Album rapAlbum = new Album("Rap",rapSongs, 2006);

String[] hipHopNames = {"Ve maahi", "Duniya", "Bolna", "Kabira", "Vaaste"};
int[] hiphopRatings = {5,9,9,4,7};
List<Track> hipHopSongs = new  ArrayList<>();
```

```
for(int i=0; i<hipHopNames.length; i++)
        hipHopSongs.add(new Track(hipHopNames[i], hiphopRatings[i]));
Album hipHopAlbum = new Album("Hip hop",hipHopSongs, 2017);


String[] jazzNames = {"Sham", "Masakali","Lovely"};
int[] jazzRatings = {3,1,2};
List<Track> jazzSongs = new  ArrayList<>();
for(int i=0; i<jazzNames.length; i++)
    jazzSongs.add(new Track(jazzNames[i], jazzRatings[i]));
Album jazzAlbum = new Album("Jazz",jazzSongs, 1995);
```

```
        List<String> sortedAlbums =  sortAlbumsByYear(Arrays.asList(travelAlbum, jazzAlbum,
hipHopAlbum, rapAlbum));
        System.out.println("Sorted list of albums by their year of release: \n"+sortedAlbums+"\n");


        List<String> goodAlbums =  filterGoodAlbums(Arrays.asList(travelAlbum, jazzAlbum,
hipHopAlbum, rapAlbum));
        System.out.println("Sorted list of Good albums(rating>4) by their year of release:
\n"+goodAlbums);


    }
}
```



Conclusion: In this experiment, we used various methods of Java Stream API and performed various programs.

Practical no. 6

FS19CO042

Aim: Implement programs related to File I/O

- 6.1 Create a csv file which will contain 10 integers in a spreadsheet. Read the file using class java.util.Scanner and display the sum of the numbers in the file. Handle all possible exceptions. Write a Java program to create, read and modify a file.
- 6.2 Create two objects of class Path viz., source and target. Perform the following operations a. Create a file at source b. Copy a file from source to target c. Move a file from source to target d. Delete a file from source e. Retrieve information about source and target

Tools used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java file I/O

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

The two important streams are **FileInputStream** and **FileOutputStream**

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file −

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have *InputStream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file −

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Once you have *OutputStream* object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

Directories in Java

A directory is a File which can contain a list of other files and directories. You use **File** object to create directories, to list down files available in a directory. For complete detail, check a list of all the methods which you can call on File object and what are related to directories.

Creating Directories

There are two useful **File** utility methods, which can be used to create directories −

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Code:

- **Create a csv file which will contain 10 integers in a spreadsheet. Read the file using class java.util.Scanner and display the sum of the numbers in the file. Handle all possible exceptions. Write a Java program to create, read and modify a file.**

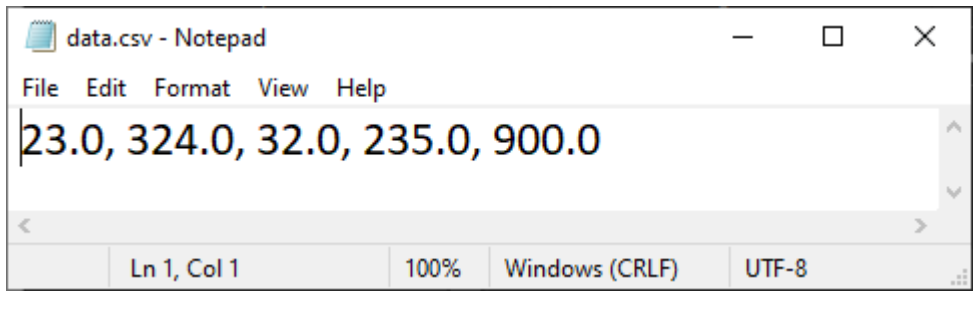
```
import java.util.*;
import java.io.*;


public class exp6_1 {
    public static void main(String[] args) {
        double sum = 0;
        List<Double> numbers = new ArrayList<>();


        String dataFilePath = "data.csv";
        String defaultData = "200,500,25,50\n100,50,25\n70,30,40,60\n40\n60,90,150\n20,40";
        Scanner inputScanner = new Scanner(System.in);
        char opted;

        File dataFile = new File(dataFilePath);


        // If file doesn't exist, creating one and writing default data to the file
        if (!dataFile.exists()) {
            System.out.println("CSV file could not be found, hence creating one !\nYou can put desired data in the file manually separated by commas");
            try {
                if (dataFile.createNewFile()) {
                    System.out.println("File created: " + dataFile.getAbsolutePath());
                    FileWriter defauFileWriter = new FileWriter(dataFile.getName());
                    defauFileWriter.write(defaultData);
```

```
            defauFileWriter.close();
        } else
            System.out.println("There was a problem creating new file, try creating one manually");

    } catch (Exception e) {
        System.out.println("An error occurred while writing default data to the file, try writing manually :)");
        e.printStackTrace();
    }
}

// Find sum of all numbers in csv file
// All numbers in CSV file:
displaySum(dataFile, sum, numbers);

// Modify the numbers in csv file

System.out.print("Do you want to write numbers to csv file? y/n: ");
opted = inputScanner.nextLine().trim().charAt(0);
if (opted == 'y' || opted == 'Y') {
    int n;
    double entity;
    System.out.println("How many decimal numbers do you wish to write to file(eg. 4 or 8): ");
    n = inputScanner.nextInt();
    System.out.println("Enter the numbers separated by spaces:");
    List<Double> newData = new ArrayList<>();
while (n-- > 0) {
    entity = inputScanner.nextDouble();
    newData.add(entity);
}
String data = newData.toString();
data = data.substring(1, data.length() - 1);
System.out.println(data);

try {
    FileWriter customWriter = new FileWriter(dataFile.getName());
    customWriter.write(data);
    customWriter.close();
```

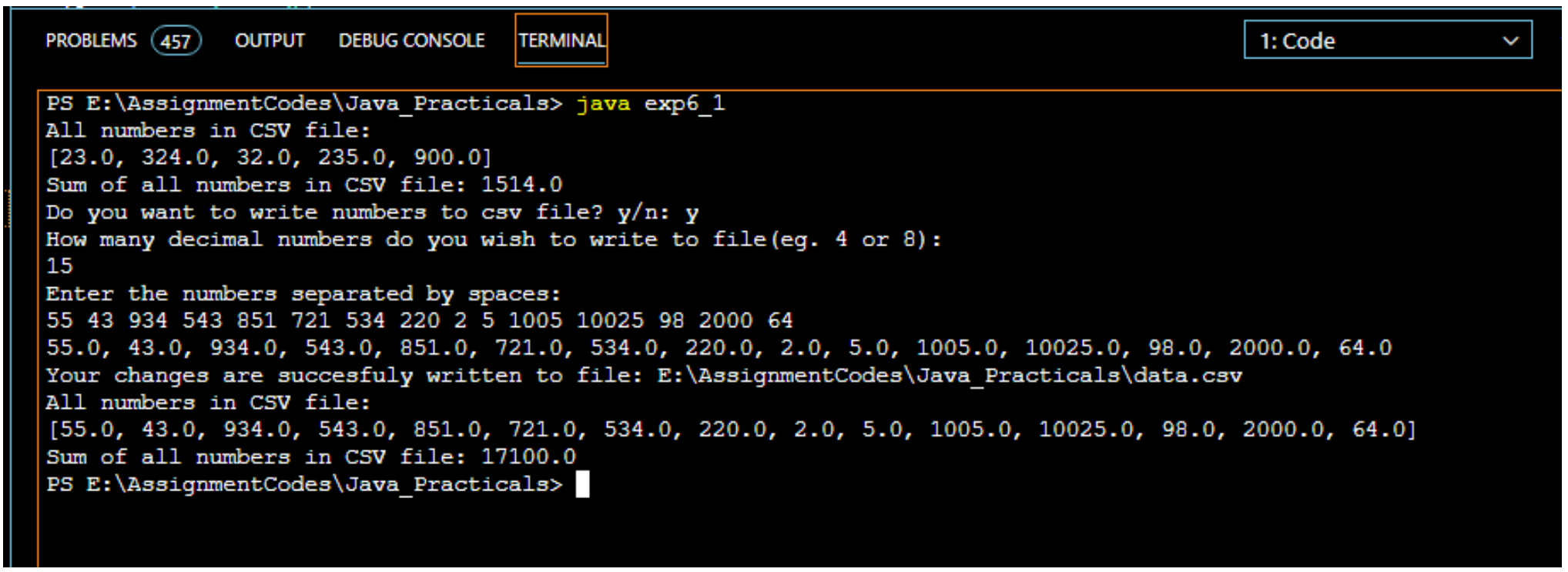
```
            System.out.println("Your changes are succesfuly written to file: " + dataFile.getAbsolutePath());
        } catch (Exception e) {
            System.out.println("xxxxxx ERROR xxxxxxxxx::: Close the CSV file And Try Again   :)");
            e.printStackTrace();
        }
    }
    numbers.clear();
    sum = 0;
    displaySum(dataFile, sum, numbers);                 // All numbers in CSV file:
    inputScanner.close();
}

private static void displaySum(File dataFile, double sum, List<Double> numbers) {
    try {
        Scanner csvScanner = new Scanner(dataFile);
        Scanner dataScanner = null;
        while (csvScanner.hasNextLine()) {
            dataScanner = new Scanner(csvScanner.nextLine());
            dataScanner.useDelimiter(",");

            while (dataScanner.hasNext()) {
                try {
                    String data = dataScanner.next().trim();
                    numbers.add(Double.parseDouble(data));
                    sum += Double.parseDouble(data);
                } catch (NumberFormatException ne) {
                    continue;
                }
            }
            dataScanner.close();
        }
        csvScanner.close();
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }

    System.out.println("All numbers in CSV file:\n" + numbers.toString());
```

```
        System.out.println("Sum of all numbers in CSV file: " + sum);
    }
}
```

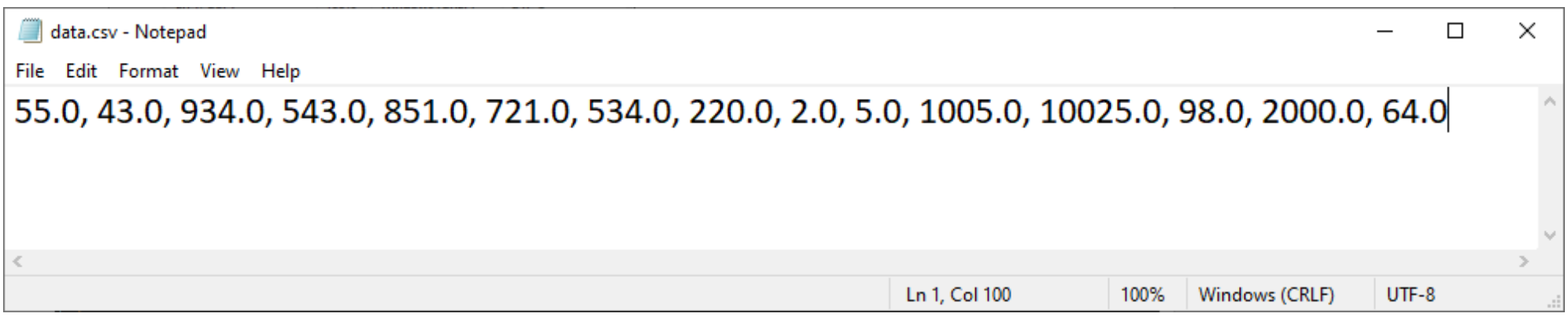
Previous CSV file:



Output:



CSV file after modifications:



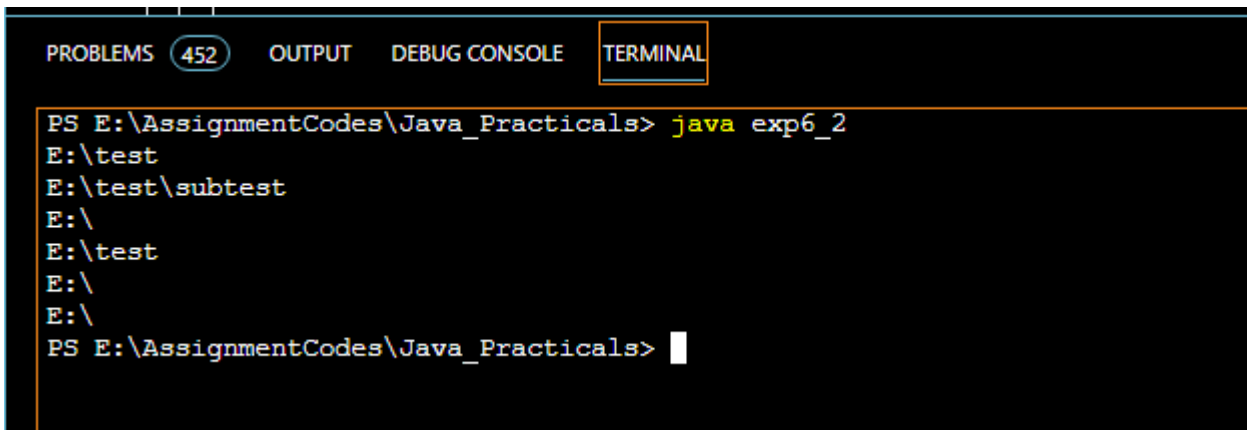
- **Create two objects of class Path viz., source and target. Perform the following operations a. Create a file at source b. Copy a file from source to target c. Move a file from source to target d. Delete a file from source e. Retrieve information about source and target**

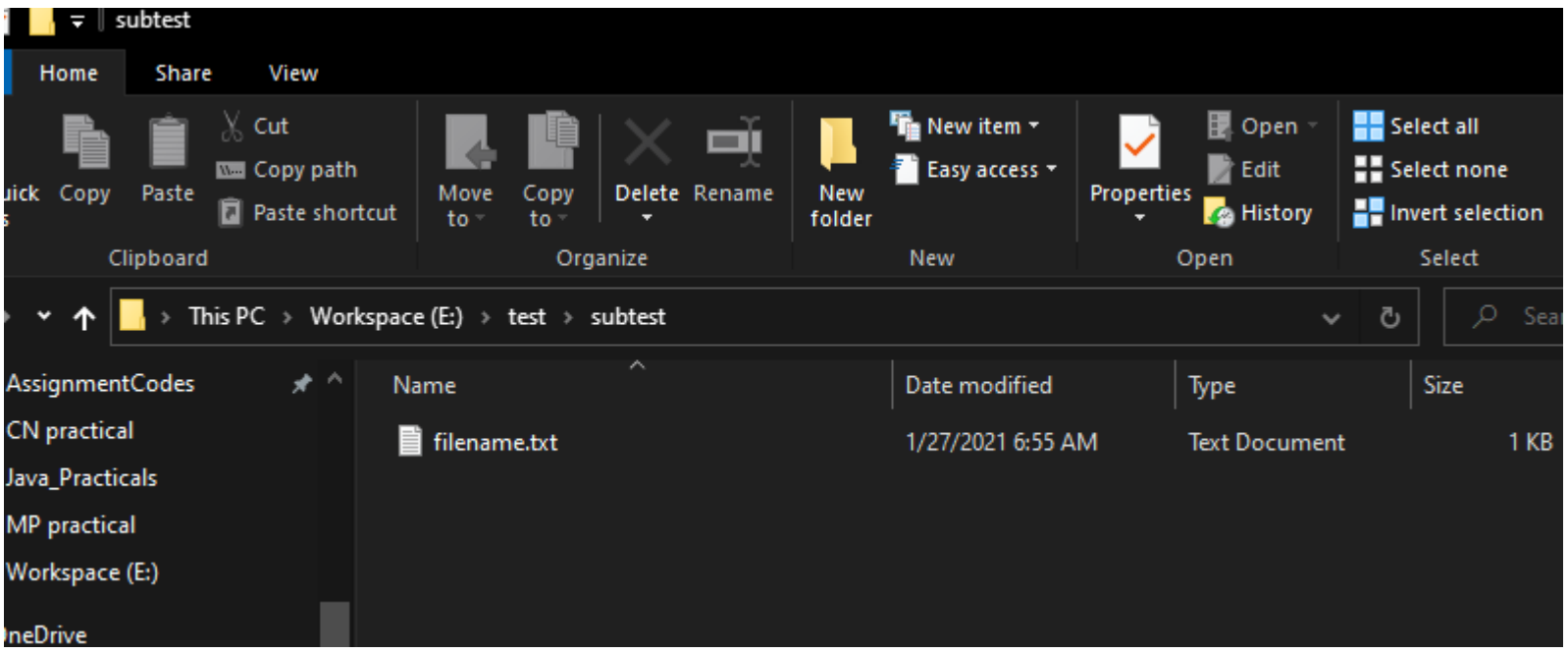
```
import java.nio.file.*;
import java.nio.file.Paths;
import java.io.*;
public class exp6_2 {

    public static void main(String[] args) {
```

```
try{
    Path source = Paths.get("E:\\test");
    Path target = Paths.get("E:\\test\\subtest");

    String fn1=source+"\\";
    String fn2=target+"\\";

    FileWriter myWriter = new FileWriter(fn1+"filename.txt");
    myWriter.write("Files in Java might be tricky, but it is fun enough!");
    myWriter.close();

    //copy
    InputStream is = null;
    OutputStream os = null;
    File s=new File(fn1+"filename.txt");
    File d=new File(fn2+"filename.txt");
    is = new FileInputStream(s);
    os = new FileOutputStream(d);
    byte[] buffer = new byte[1024];
    int length;
    while ((length = is.read(buffer)) > 0) {
       os.write(buffer, 0, length);
    }
    is.close();
    os.close();

    //move
    d.delete();
    Path temp = Files.move(Paths.get(fn1+"filename.txt"),  Paths.get(fn2+"filename.txt"));

    //delete
    s.delete();

    //retrieve
```

```
            System.out.println(source+"");
            System.out.println(target+"");
            System.out.println(source.getParent()+"");
            System.out.println(target.getParent()+"");
            System.out.println(source.getRoot()+"");
            System.out.println(target.getRoot()+"");
        }catch(Exception ex){
            System.out.println(ex+"");
        }
    }
}
```

Output:





Conclusion: In this experiment, we performed various File Read/write operations like editing csv files, copying/moving/deleting files, etc using FIleWriter, InputStream, OutputStream, etc.

Practical no. 7

FS19CO042

Aim: Generate complete Javadocs for any two of the above experiments

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Javadoc

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.

Following is a simple example where the lines inside /*….*/ are Java multi-line comments. Similarly, the line which preceeds // is Java single-line comment.

Example

```
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Omkar Phansopkar
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {
   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

You can include required HTML tags inside the description part. For instance, the following example makes use of <h1>....</h1> for heading and <p> has been used for creating paragraph break −

Example

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
*
* @author  Omkar Phansopkar
* @version 1.0
* @since   2014-03-31
```

```
*/
public class HelloWorld {


   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

The javadoc Tags

The javadoc tool recognizes the following tags −

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.	{@code text}
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@inheritDoc}	Inherits a comment from the **nearest** inheritable class or implementable interface.	Inherits a comment from the immediate surperclass.
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
{@linkplain}	Identical to {@link}, except the link's label is displayed in plain text than code font.	{@linkplain package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to	@param parameter-name description

	the "Parameters" section.	
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField component.	@serialField field-name field-type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
{@value}	When {@value} is used in the doc comment of a static field, it displays the value of that constant.	{@value package.class#field}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

Command to generate html pages from javadocs:
javadoc -d .\pathToDestination .\pathToSrc.java file.

Code:

File 1, exp7a.java

```
import java.util.Scanner;
/**
 * in this class we are adding the square root of indivisual numbers;
 */
```

```
public class exp7a{
   /**
    * in this method we are taking the numbers as input and returning the addtion to main function;
    * @param x;
    * @return to main;
    */
   public static int add(int ...x)
   {
      return x[0]*x[0]+x[1]*x[1];
   }
   /**
    * this is the main method where the calling to add function is done and printing the result;
    * @param args
    */
  public static void main(String args[])
  {   //creating Scanner class object and passing system.in ;
      Scanner sc= new Scanner(System.in);
      System.out.println("enter the first number:");
      int n1=sc.nextInt();
      System.out.println("enter the second number:");
      int n2=sc.nextInt();
      int a=exp7a.add(n1,n2);
      System.out.print("the addtion of square root of indivisual is :"+a);
  }
}
```

File 2, exp7b.java

```
import java.util.Arrays;
/**
 * this is a assignment7 class for sorting the array;
 */
public class exp7b{
   /**
    * This is the main method where the arrays sorted and printed;
```

```
 *@param args
 */
public static void main(String args[])
{
    System.out.println("sorting the array.....");
    System.out.println("sorted array:");
    Arrays.sort(args);
    for(String i:args)
    {
      System.out.println(i);
    }
  }
}
```

Generating docs:
Perform following commands to generate doc:
javadoc -d .\pathToDestination .\pathToSrc.java file.

Actual commands:

javadoc -d .\javadocs\ .\exp7a.java

javadoc -d .\javadocs\ .\exp7b.java

Console output:

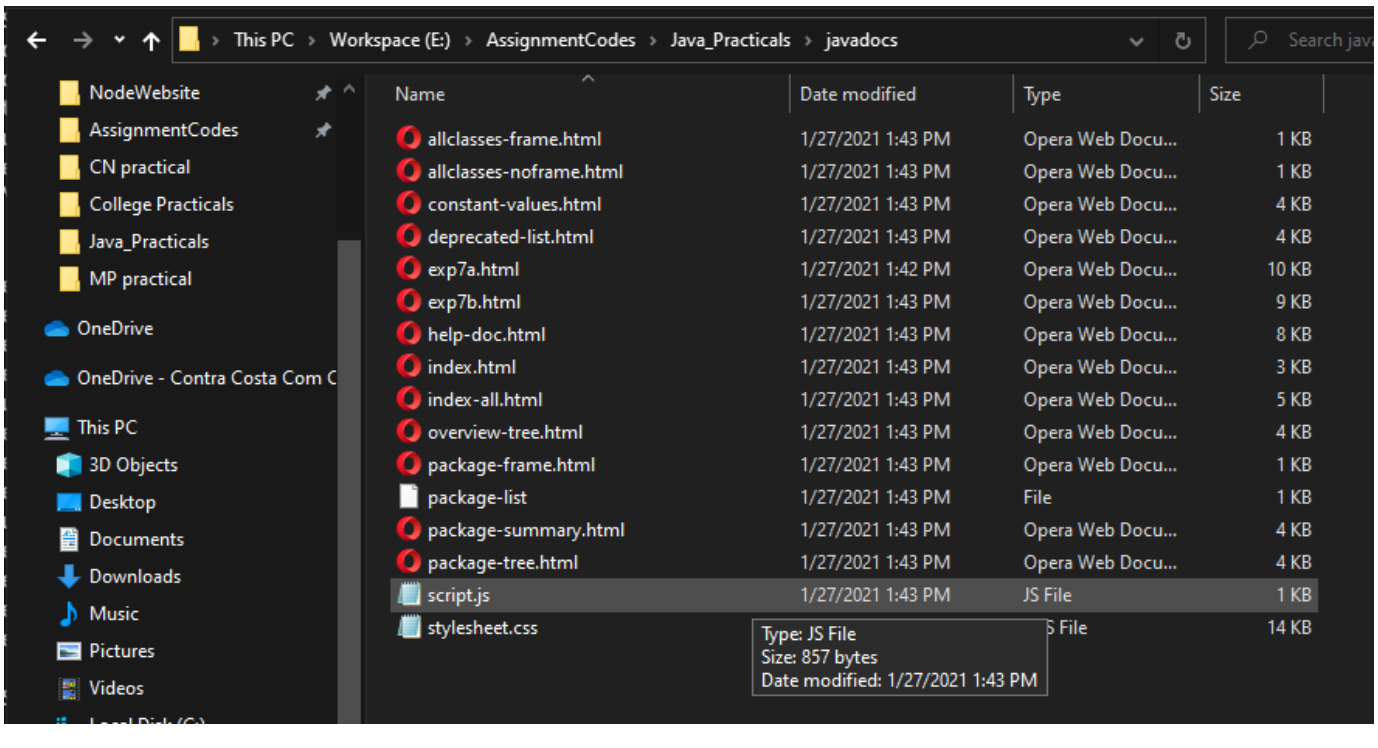
Windows PowerShell

```
PS E:\AssignmentCodes\Java_Practicals> javadoc -d .\javadocs\ .\exp7a.java
Loading source file .\exp7a.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_251
Building tree for all the packages and classes...
Generating .\javadocs\exp7a.html...
.\exp7a.java:17: warning: no description for @param
     * @param args
       ^
Generating .\javadocs\package-frame.html...
Generating .\javadocs\package-summary.html...
Generating .\javadocs\package-tree.html...
Generating .\javadocs\constant-values.html...
Building index for all the packages and classes...
Generating .\javadocs\overview-tree.html...
Generating .\javadocs\index-all.html...
Generating .\javadocs\deprecated-list.html...
Building index for all classes...
Generating .\javadocs\allclasses-frame.html...
Generating .\javadocs\allclasses-noframe.html...
Generating .\javadocs\index.html...
Generating .\javadocs\help-doc.html...
1 warning
PS E:\AssignmentCodes\Java_Practicals>
```

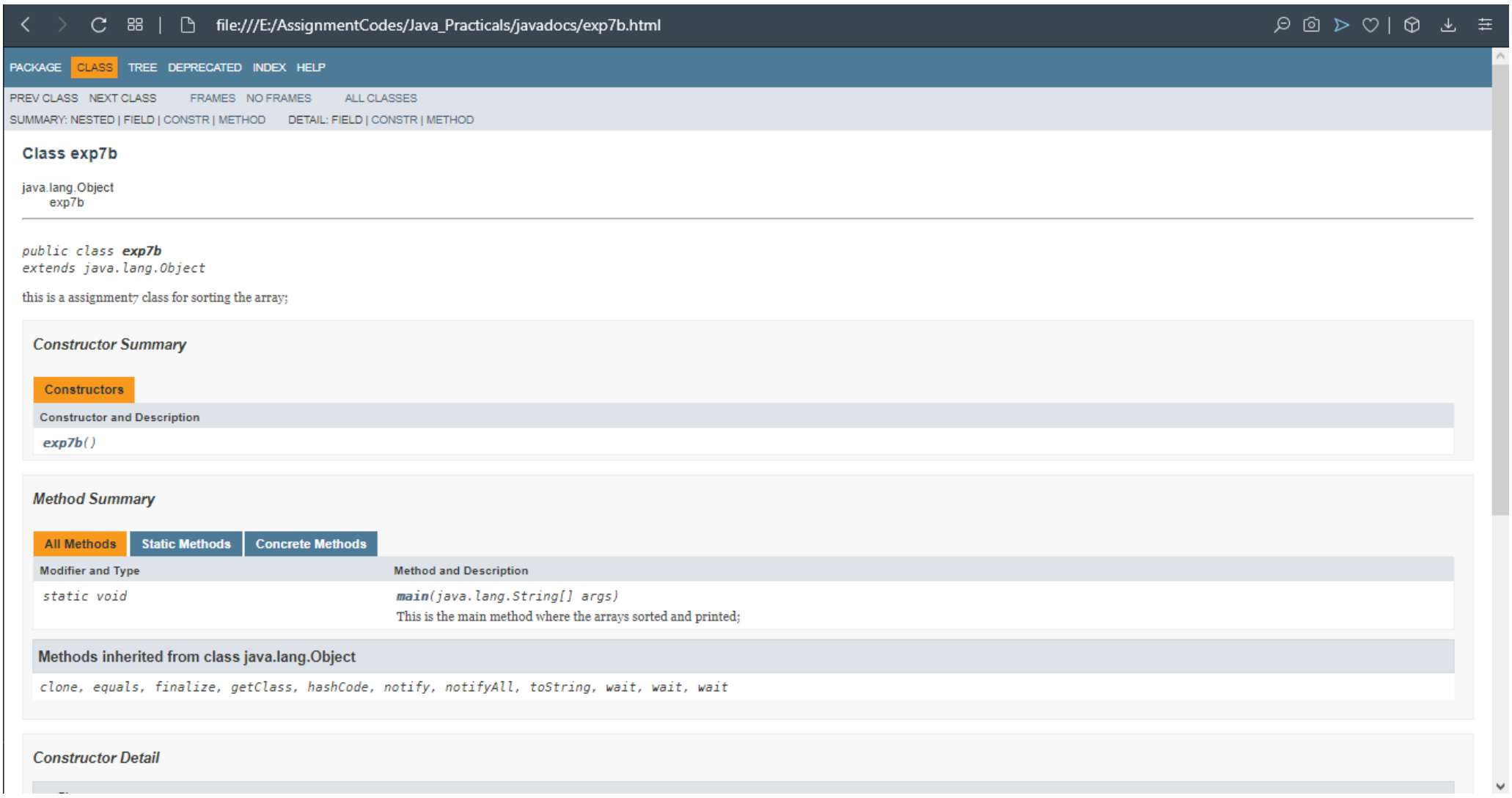
Windows PowerShell

```
PS E:\AssignmentCodes\Java_Practicals> javadoc -d .\javadocs\ .\exp7b.java
Loading source file .\exp7b.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_251
Building tree for all the packages and classes...
Generating .\javadocs\exp7b.html...
.\exp7b.java:8: warning: no description for @param
    *@param args
     ^
Generating .\javadocs\package-frame.html...
Generating .\javadocs\package-summary.html...
Generating .\javadocs\package-tree.html...
Generating .\javadocs\constant-values.html...
Building index for all the packages and classes...
Generating .\javadocs\overview-tree.html...
Generating .\javadocs\index-all.html...
Generating .\javadocs\deprecated-list.html...
Building index for all classes...
Generating .\javadocs\allclasses-frame.html...
Generating .\javadocs\allclasses-noframe.html...
Generating .\javadocs\index.html...
Generating .\javadocs\help-doc.html...
1 warning
PS E:\AssignmentCodes\Java_Practicals>
```

Docs Output:


This PC › Workspace (E:) › AssignmentCodes › Java_Practicals › javadocs
NodeWebsite
AssignmentCodes
CN practical
College Practicals
Java_Practicals
MP practical
OneDrive
OneDrive - Contra Costa Com C
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Name
Date modified
Type
Size
allclasses-frame.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
allclasses-noframe.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
constant-values.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
deprecated-list.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
exp7a.html 1/27/2021 1:42 PM Opera Web Docu... 10 KB
exp7b.html 1/27/2021 1:43 PM Opera Web Docu... 9 KB
help-doc.html 1/27/2021 1:43 PM Opera Web Docu... 8 KB
index.html 1/27/2021 1:43 PM Opera Web Docu... 3 KB
index-all.html 1/27/2021 1:43 PM Opera Web Docu... 5 KB
overview-tree.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
package-frame.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
package-list 1/27/2021 1:43 PM File 1 KB
package-summary.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
package-tree.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
script.js 1/27/2021 1:43 PM JS File 1 KB
stylesheet.css File 14 KB
Type: JS File
Size: 857 bytes
Date modified: 1/27/2021 1:43 PM

file:///E:/AssignmentCodes/Java_Practicals/javadocs/exp7a.html
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Class exp7a
java.lang.Object
exp7a
public class exp7a
extends java.lang.Object
in this class we are adding the square root of indivisual numbers;
Constructor Summary
Constructors
Constructor and Description
exp7a()
Method Summary
All Methods Static Methods Concrete Methods
Modifier and Type
Method and Description
static int
add(int... x)
in this method we are taking the numbers as input and returning the addtion to main function;
static void
main(java.lang.String[] args)
this is the main method where the calling to add function is done and printing the result;
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Conclusion: Thus we understood and successfully created javadocs for our project using various techniques used for commenting and documenting java experiments.

Java practicals

FS19CO042

Omkar Phansopkar

FS19CO042 SECOND YEAR, FIRST SHIFT

Practical no. 1

FS19CO042

Aim:

Getting started with Java Application Development using IDE

1.1 Check whether latest version of java (at least JDK 1.8)is installed or not. If not then download and install it.

1.2 Download and install the IntelliJ IDEA Community Edition/ NetBeans IDE 8.1/ Eclipse Neon or later version of IDE

1.3 Create a Java Project/ Application in the IDE

1.4 Create a Java class Person containing two variables name and yearOfBirth of appropriate data types, take inputs from the command line argument, a method to display the name and age of the person.

1.5 Save the project and run it.

1.6 Explore all the features (the menu and shortcuts) of the IDE. Learn about their use

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called *OpenJDK*.

Java virtual machine (JVM)

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.
The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.

Java programs are compiled by the Java compiler into *bytecode*. The Java virtual machine interprets this *bytecode* and executes the Java program

Java Development kit (JDK)

The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries

Java program workflow:

javac hello.java -> Compiles java source file into a class file and new hello.class file is generated
java hello -> Executes class file generated in java virtual machine.

Base structure of Java program:

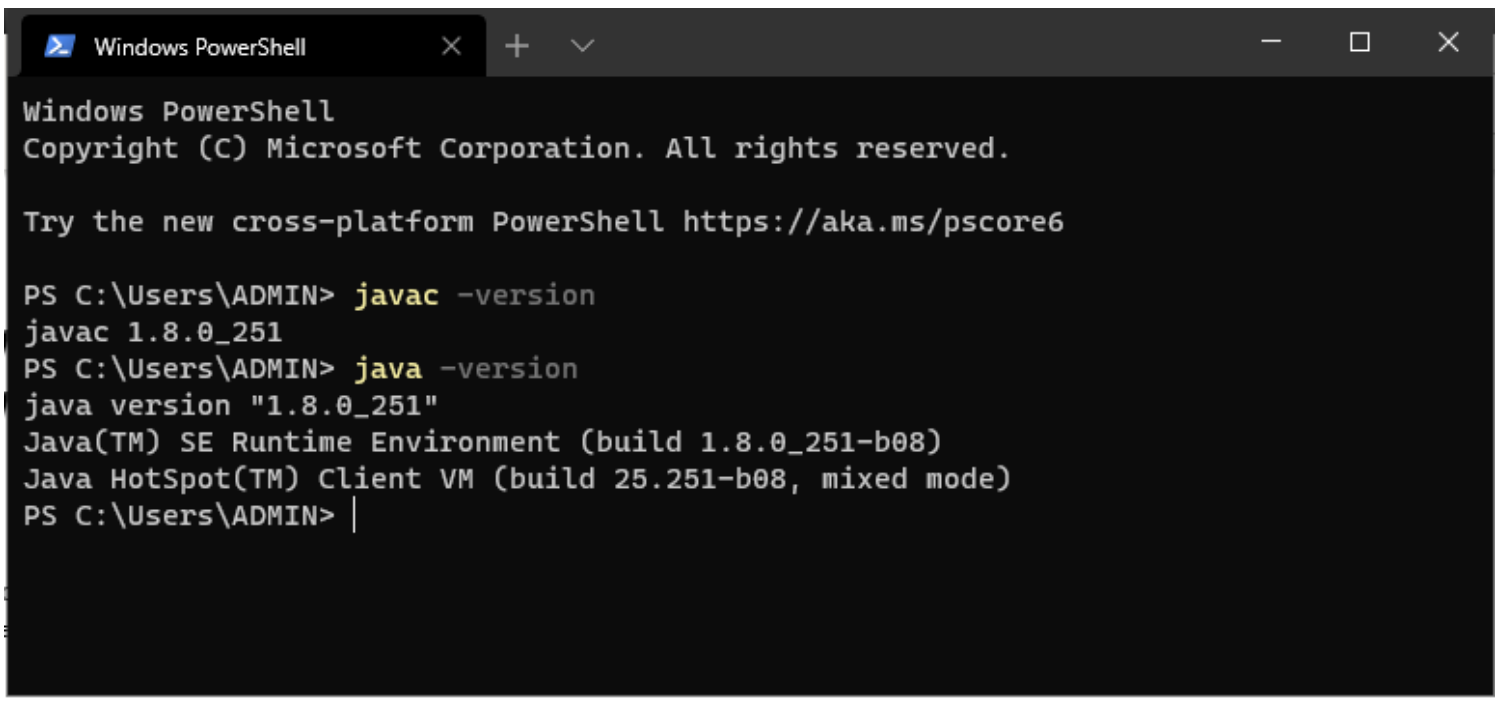
```
package test;
class Hello{
    public static void main(String[] args){
        // Code goes here
```

```
    }
}
```

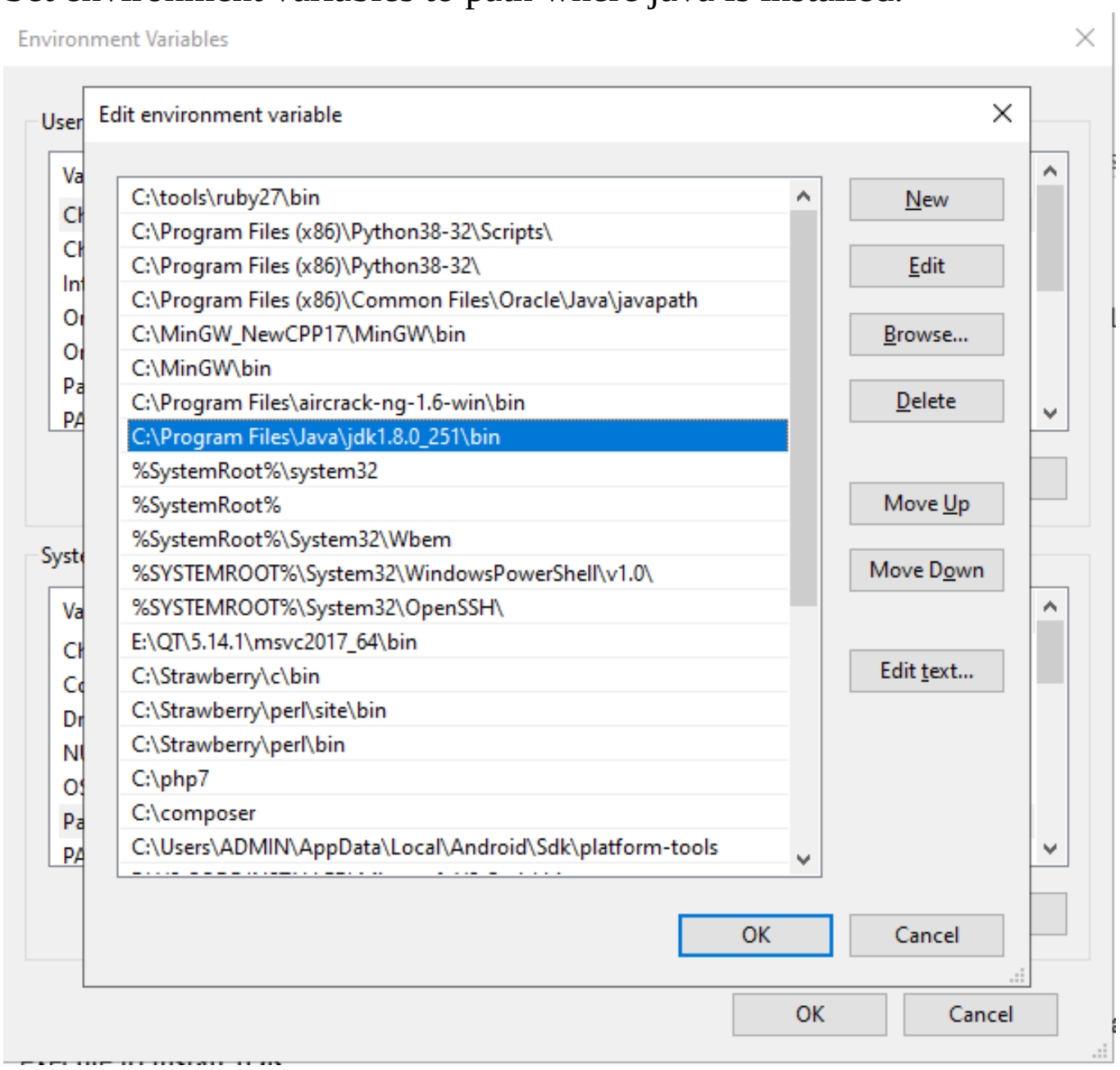
Steps:

1.1 Check whether latest version of Java (at least JDK 1.8) is installed or not. If not then download and install it.

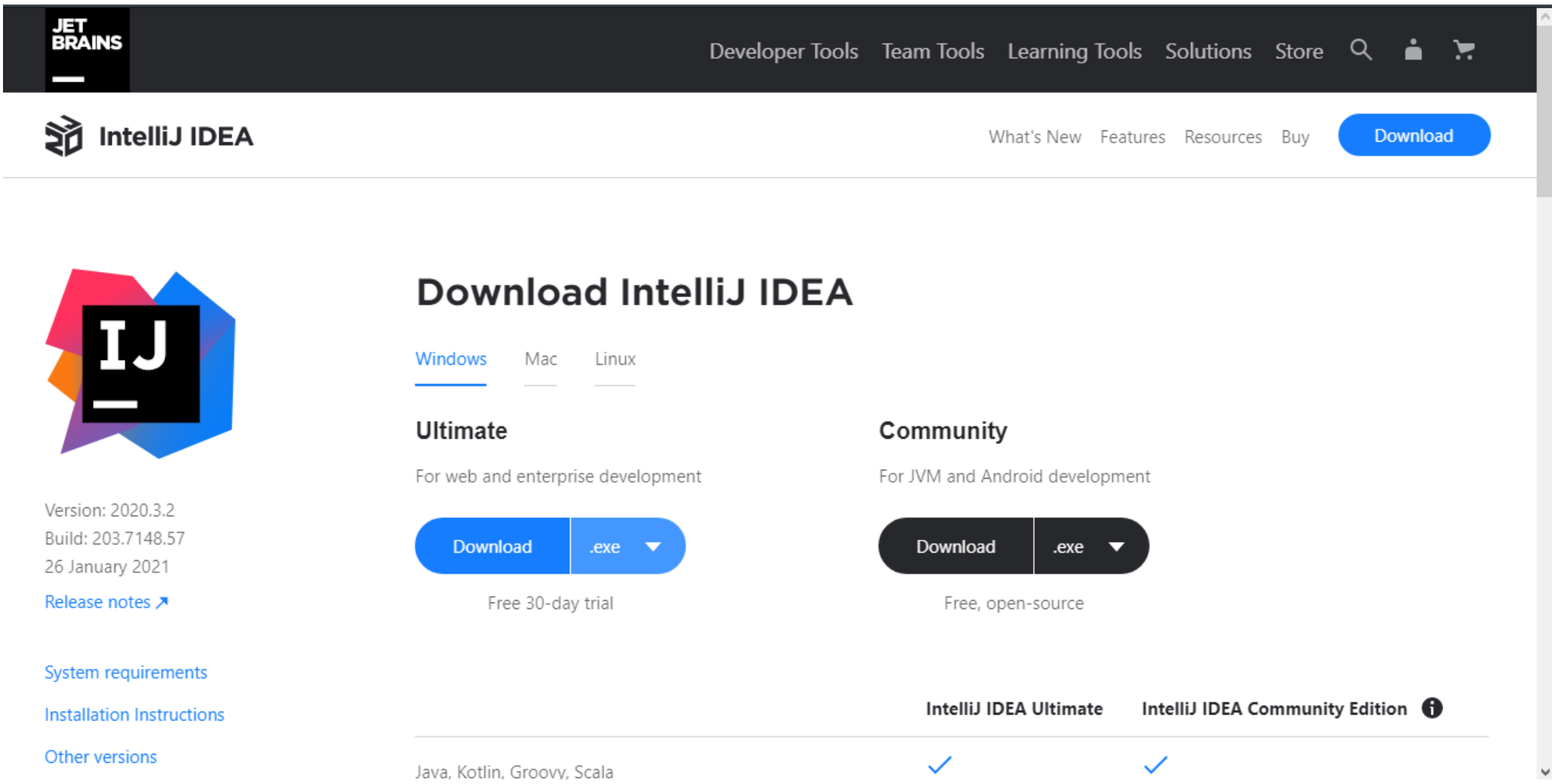
- Check if java compiler(javac) and jvm (java) are installed properly on system.
  Type these commands:
  javac -version
  java -version

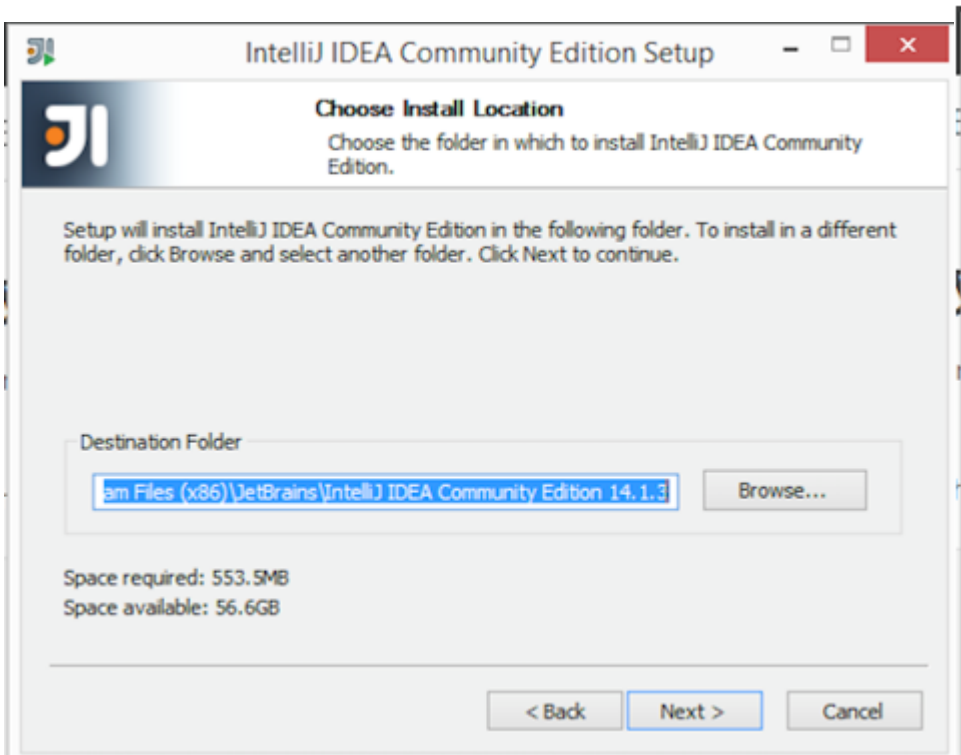


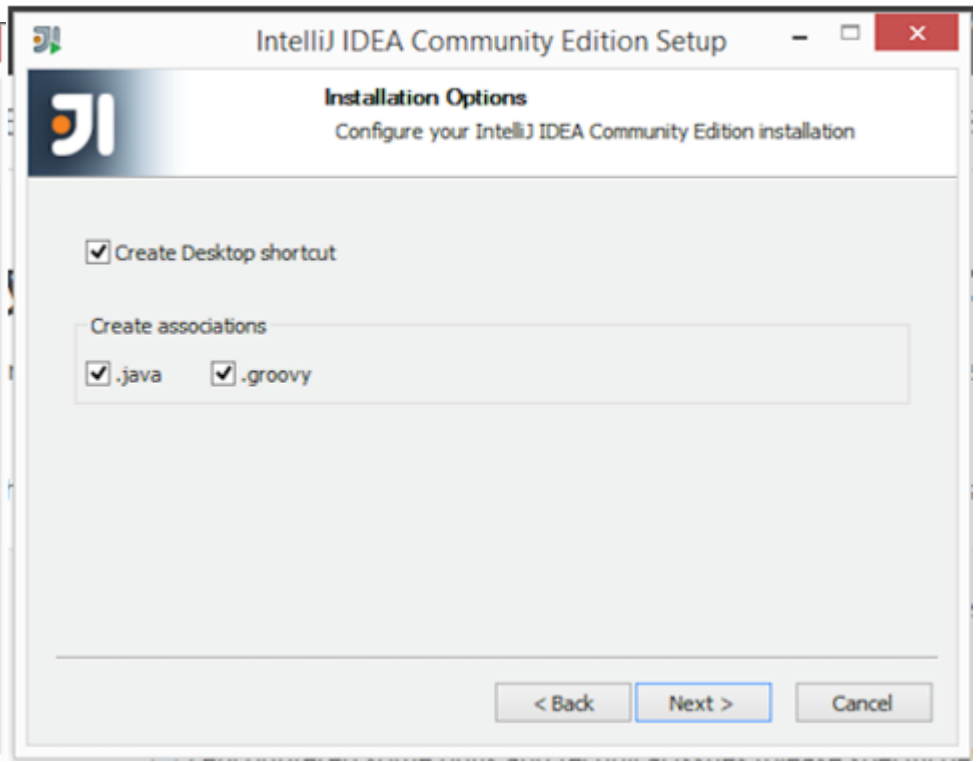
- If java compiler or jvm is not installed on system, download executibles from Oracle.com and execute to install JDK.
- Set environment variables to path where java is installed.

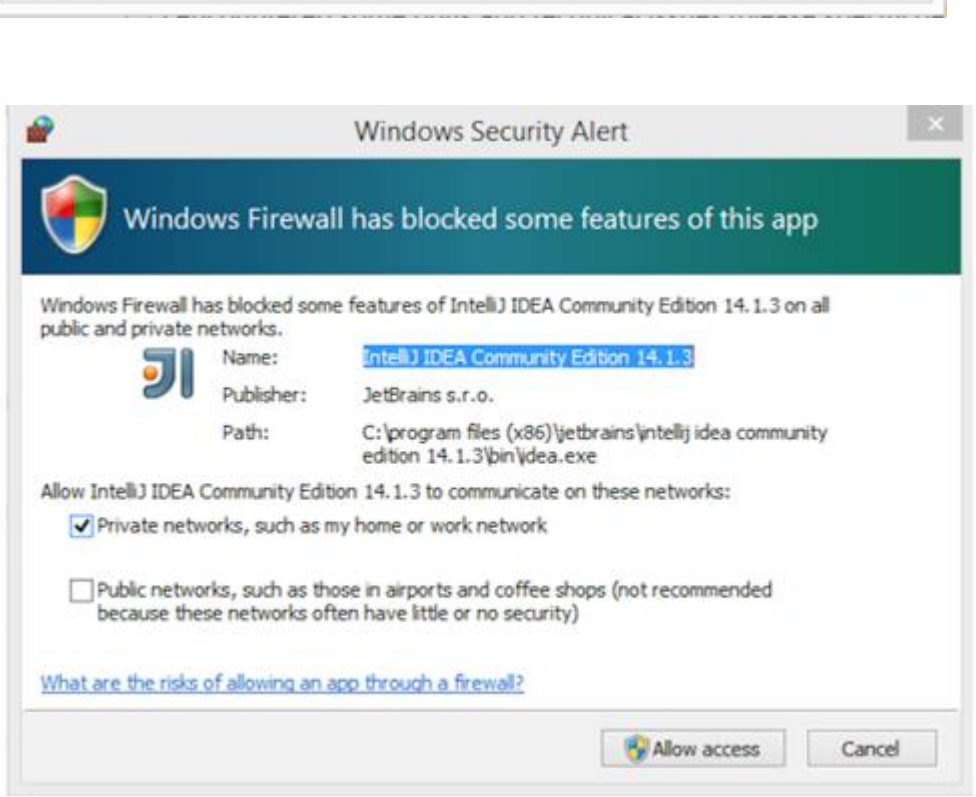


1.2 Download and install the IntelliJ IDEA Community Edition/ NetBeans IDE 8.1/ Eclipse Neon or later version of IDE

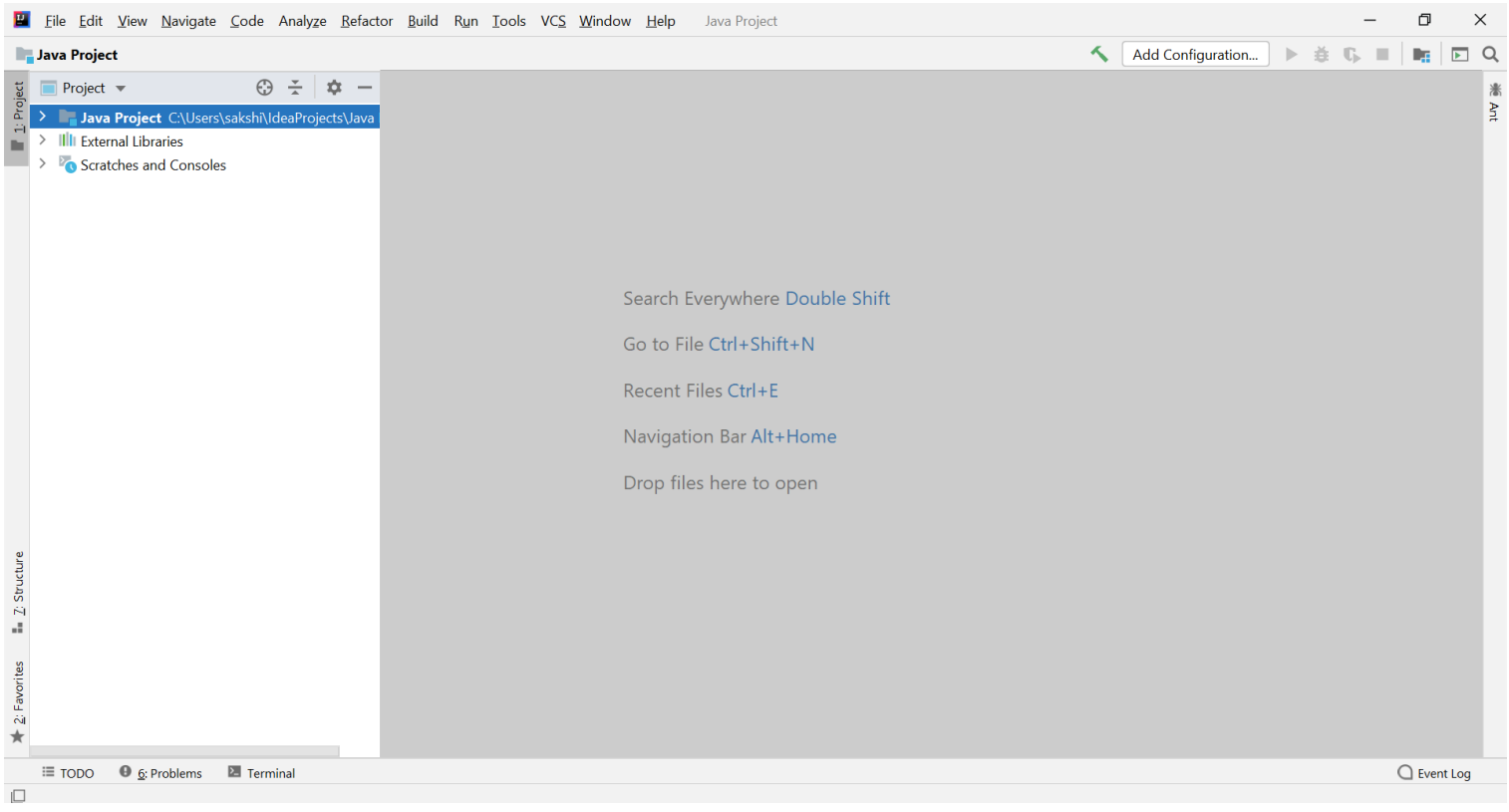








1.3 Create a Java Project/ Application in the IDE

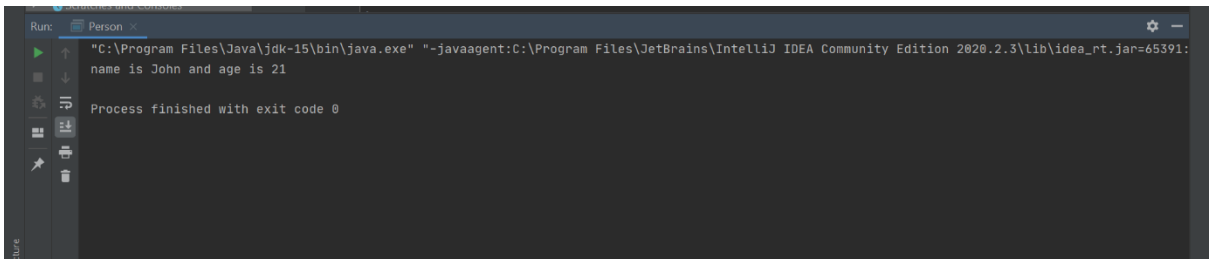


1.4 Create a Java class Person containing two variables name and yearOfBirth of appropriate data types, take inputs from the command line argument, a method to display the name and age of the person.

```
public class Person
{
   String name;
   int yearOfBirth;

   public static void disp_details(String a,int b)
   {
     int current_year=2021;
     int age=current_year-b;
     System.out.println("name is "+a+" and age is "+age);

   }


   public static void main(String [] args )
   {
      disp_details("sakshi",2004);
   }
}
```

1.5 Save the project and run it.

output :



1.6 Explore all the features (the menu and shortcuts) of the IDE. Learn about their use

Shortcut	Action
Double Shift	Search Everywhere

	Find anything related to IntelliJ IDEA or your project and open it, execute it, or jump to it.
Ctrl+Shift+A	Find Action Find a command and execute it, open a tool window or search for a setting.
Alt+Enter	Show intention actions and quick-fixes Fix highlighted error or warning, improve or optimize a code construct.
F2 Shift+F2	Navigate between code issues Jump to the next or previous highlighted error.
Ctrl+E	View recent files Select a recently opened file from the list.
Ctrl+Shift+Enter	Complete current statement Insert any necessary trailing symbols and put the caret where you can start typing the next statement.
Ctrl+Alt+L	Reformat code Reformat the whole file or the selected fragment according to the current code style settings.
Ctrl+Alt+Shift+T	Invoke refactoring Refactor the element under the caret, for example, safe delete, copy, move, rename, and so on.
Ctrl+W Ctrl+Shift+W	Extend or shrink selection Increase or decrease the scope of selection according to specific code constructs.
Ctrl+/ Ctrl+Shift+/	Add/remove line or block comment Comment out a line or block of code.
Ctrl+B	Go to declaration Navigate to the initial declaration of the instantiated class, called method, or field.
Alt+F7	Find usages Show all places where a code element is used across your project.
Alt+1	Focus the Project tool window
Escape	Focus the editor

Conclusion: Thus, we verified/installed java and IntelliJ IDE and ran first program on it. We also explored various features of the IDE.

Practical no. 2

FS19CO042

Aim: Perform following programs.

2.1 Write a program to print “Hello World”.

2.2 Write a program to print addition of two integers.

2.3 Write a program to convert a numeric string into int.

2.4 Write a program to print addition of two integers input from command line arguments.

2.5 Write a program to take two integers from command line, subtract the smaller number from the greater and print the result.

2.6 Write a program to take n integers from command line and print their sum of product (product of first number and last number added to product of second number and second last number and so on).

2.7 Consider any two integers. Write a program to print sum of their squares.

2.8 Write a program to find square root of a given positive integer using Heron’s method to find square root.

2.9 Write a program to sort and print the names of students taken from command line in alphabetical order.

2.10 Write a program to print total numbers of vowels and consonants in a given string.

2.11 Given two English words, write a program to check if the first word is anagram of the second word. (An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. (Example: Anagram of TOM MARVOLO RIDDLE is I AM LORD VOLDEMORT.)

2.12 Write a program to print a missing number in a sorted integer array.

2.13 Write a program to find all the pairs of numbers on an integer array whose sum is equal to a given number.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

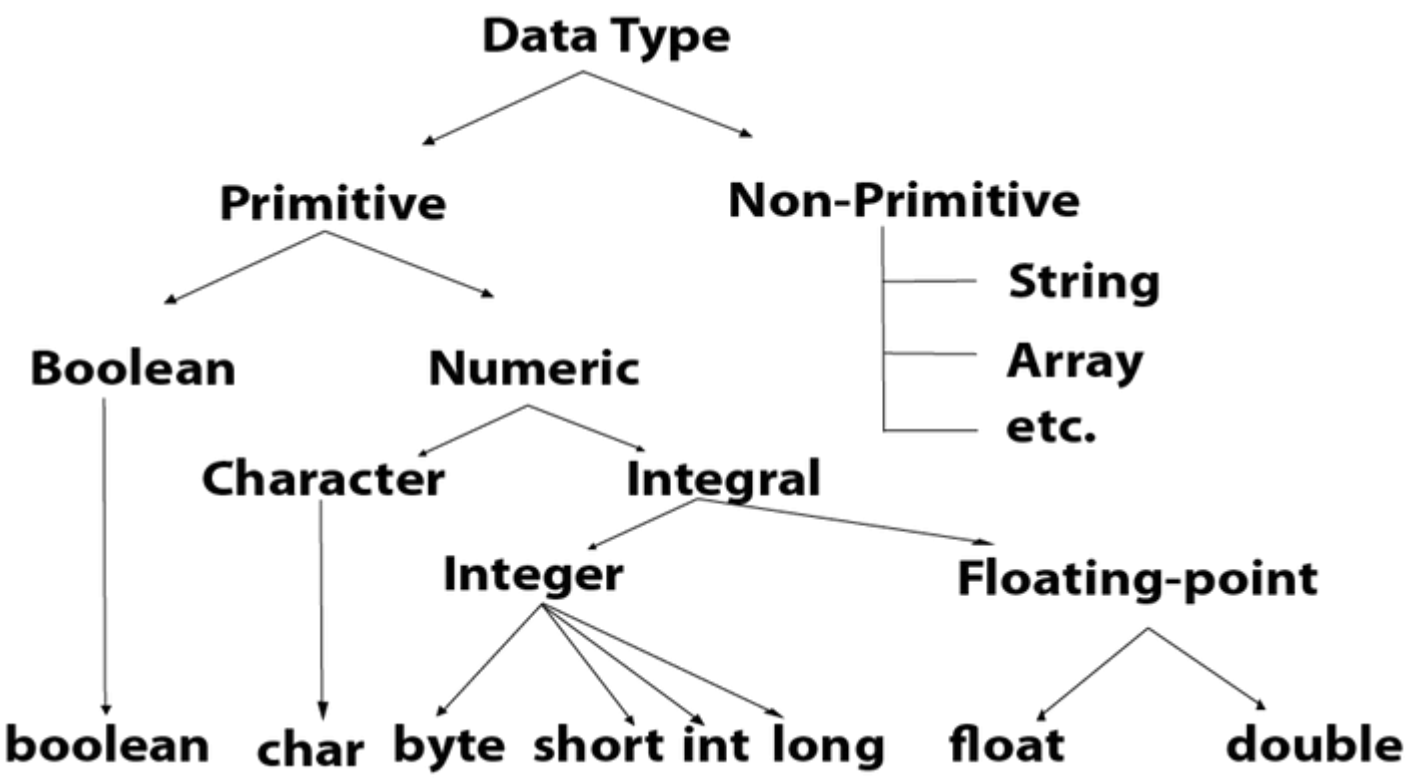
Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type

- short data type
- int data type
- long data type
- float data type
- double data type



Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Java Operators

In this tutorial, you'll learn about different types of operators in Java, their syntax and how to use them with the help of examples.

Operators are symbols that perform operations on variables and values. For example, `+` is an operator used for addition, while `*` is also an operator used for multiplication.

Operators in Java can be classified into 5 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators

---

1. Java Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example, a + b;

Here, the `+` operator is used to add two variables `a` and `b`. Similarly, there are various other arithmetic operators in Java.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. Java Assignment Operators

Assignment operators are used in Java to assign values to variables. For example,

int age;

age = 5;

Here, `=` is the assignment operator. It assigns the value on its right to the variable on its left. That is, **5** is assigned to the variable `age`.

Let's see some more assignment operators available in Java.

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

3. Java Relational Operators

Relational operators are used to check the relationship between two operands. For example,

```
// check is a is less than b

a < b;
```

Here, `>` operator is the relational operator. It checks if `a` is less than `b` or not.

It returns either `true` or `false`.

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **false**

4. Java Logical Operators

Logical operators are used to check whether an expression is `true` or `false`. They are used in decision making.

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

5. Java Unary Operators

Unary operators are used with only one operand. For example, `++` is a unary operator that increases the value of a variable by **1**. That is, `++5` will return **6**.

Different types of unary operators are:

Operator	Meaning
`+`	**Unary plus**: not necessary to use since numbers are positive without using it
`-`	**Unary minus**: inverts the sign of an expression
`++`	**Increment operator**: increments value by 1
`--`	**Decrement operator**: decrements value by 1
`!`	**Logical complement operator**: inverts the value of a boolean

Increment and Decrement Operators

Java also provides increment and decrement operators: `++` and `--` respectively. `++` increases the value of the operand by **1**, while `--` decrease it by **1**. For example,

```
int num = 5;

// increase num by 1

++num;
```

Here, the value of `num` gets increased to **6** from its initial value of **5**.

Example 5: Increment and Decrement Operators

In the above program, we have used the ++ and -- operator as **prefixes (++a, --b)**. We can also use these operators as **postfix (a++, b++)**.

There is a slight difference when these operators are used as prefix versus when they are used as a postfix.

6. Java Bitwise Operators

Bitwise operators in Java are used to perform operations on individual bits. For example,

```
Bitwise complement Operation of 35
35 = 00100011 (In Binary)
~ 00100011
_________
  11011100  = 220 (In decimal)
```

Here, `~` is a bitwise operator. It inverts the value of each bit (**0** to **1** and **1** to **0**).

The various bitwise operators present in Java are:

Operator	Description
`~`	Bitwise Complement

<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

These operators are not generally used in Java. To learn more, visit Java Bitwise and Bit Shift Operators.

Other operators

Besides these operators, there are other additional operators in Java.

Java instanceof Operator

The `instanceof` operator checks whether an object is an instanceof a particular class.

Here, `str` is an instance of the `String` class. Hence, the `instanceof` operator returns `true`.

Java Ternary Operator

The ternary operator (conditional operator) is shorthand for the `if-then-else` statement. For example,

variable = Expression ? expression1 : expression2

Here's how it works.

- If the Expression is true, expression1 is assigned to the variable.
- If the Expression is false, expression2 is assigned to the variable

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

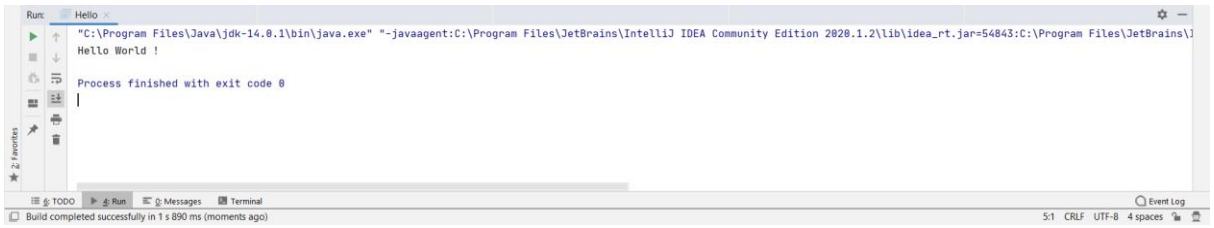
Code:

2.1 Write a program to print “Hello World”.

Code:

```
public class Hello {

   public static void main(String[] args) {
      System.out.println("Hello World !");
   }
}
```

Output:

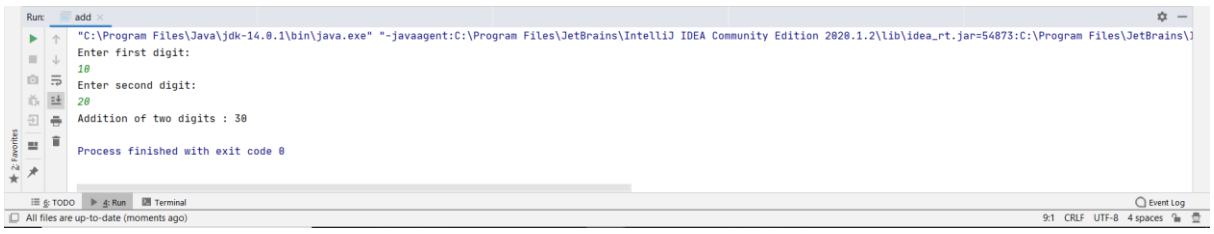


2.2 Write a program to print addition of two integers.

Code:

```
import java.util.Scanner;
public class add {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter first digit:");

      int a = scanner.nextInt();

      System.out.println("Enter second digit:");

      int b = scanner.nextInt();

      int c = a+b;

      System.out.println("Addition of two digits : " + c);

   }

}
```

Output:

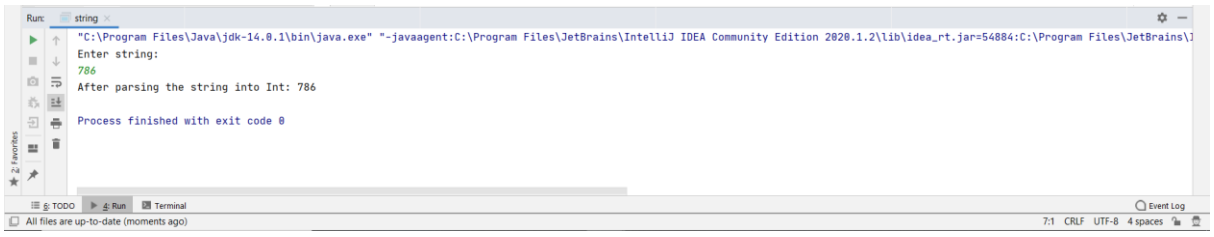


2.3 Write a program to convert a numeric string into int.

Code:

```
import java.util.Scanner;
public class string {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Enter string:");

      String s = scanner.nextLine();

      int number = Integer.parseInt(s);

      System.out.println("After parsing the string into Int: " + number);

   }

}
```

Output:



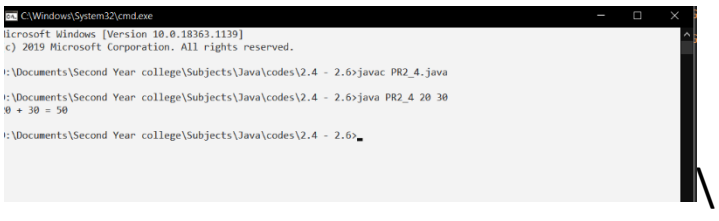
2.4 Write a program to print addition of two integers input from command line arguments.

Code:

```
public class PR2_4 {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      int num1, num2, sum;
```

```
        num1 = Integer.parseInt(args[0]);

        num2 = Integer.parseInt(args[1]);
        sum = num1+num2;
        System.out.println(num1+" + "+num2+" = "+sum);
    }
}
```

Output:

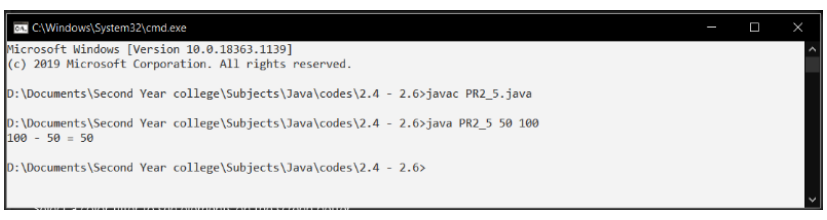


2.5 Write a program to take two integers from command line, subtract the smaller number from the greater and print the result.

Code:

```
public class PR2_5 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num1, num2;
         num1 = Integer.parseInt(args[0]);
         num2 = Integer.parseInt(args[1]);
        System.out.println(num1>num2 ? num1 +" - "+num2+" = "+(num1-num2) : num2 +" - "+num1+" =
"+(num2-num1));
    }
}
```

Output :



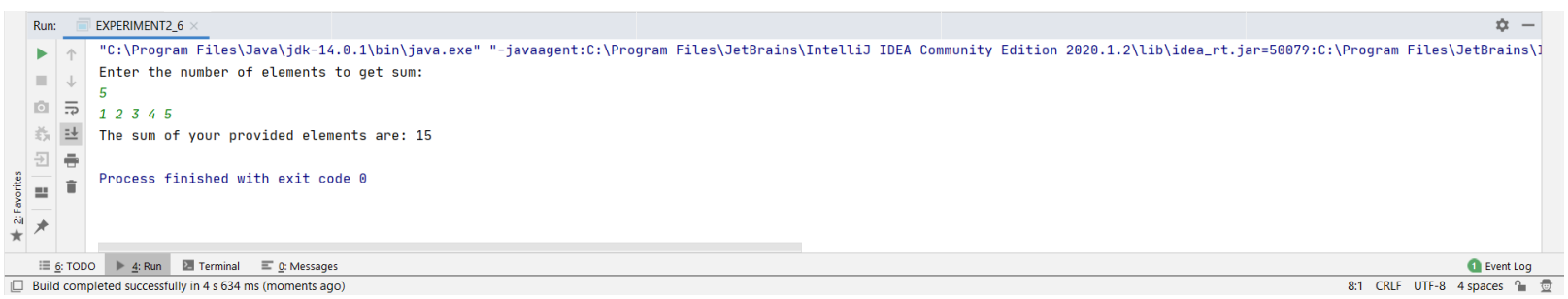
2.6 Write a program to take n integers from command line and print their sum of product (product of first number and last number added to product of second number and second last number and so on).

Code:

```
public class EXPERIMENT2_6 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter the number of elements to get sum:");

int n = sc.nextInt();
int [] arr = new int[n];
int sum = 0;

for(int k=0; k<arr.length; k++)
   arr[k] = sc.nextInt();

for(int i=0; i<arr.length/2; i++){

    int result = arr[i] + arr[arr.length-1-i];
    sum += result;
}

if(n%2 != 0){
        int middleIndex = ((n-1)/2);
        sum += arr[middleIndex];
    }
    System.out.println("The sum of your provided elements are: "+sum);

  }
}
```

Output:



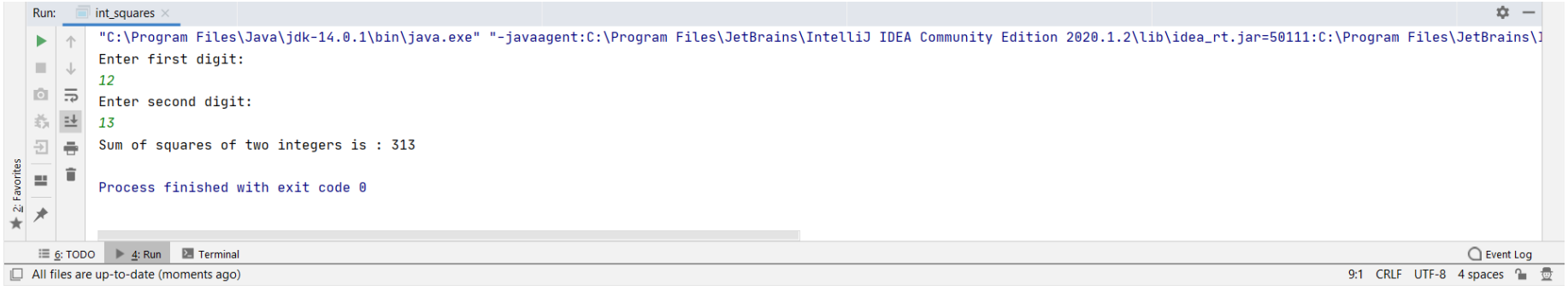
2.7 Consider any two integers. Write a program to print sum of their squares.

Code:

```
import java.util.Scanner;
public class int_squares {

   public static void main(String[] args) {

       Scanner sc = new Scanner(System.in);
       int a,b,c,d,e;
       System.out.println("Enter first digit:");

       a=sc.nextInt();
       b = a*a;
       System.out.println("Enter second digit:");

       c = sc.nextInt();
```

```
        d = c*c;
        e = b+d;
        System.out.println("Sum of squares of two integers is : " + e);
    }
}
```

Output:



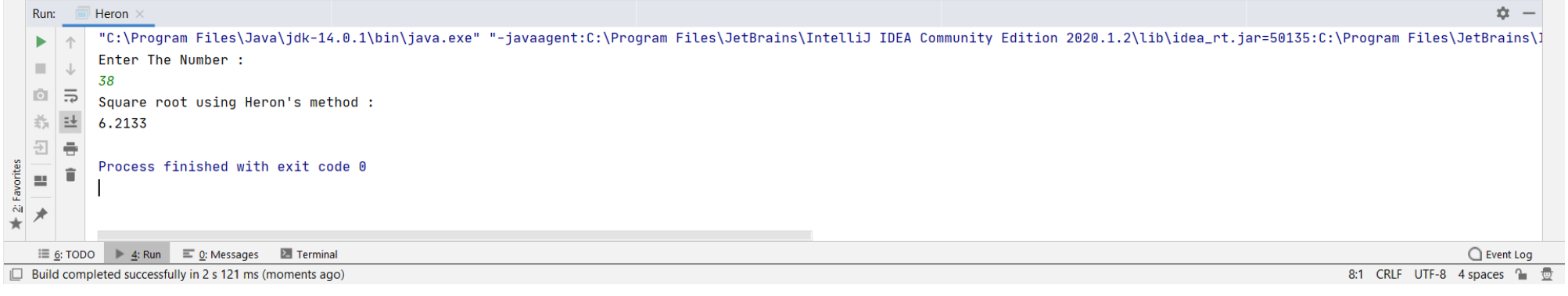
2.8 Write a program to find square root of a given positive integer using Heron's method to find square root.

Code:

```
import java.util.Scanner;
public class Heron {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter The Number : ");

        int a = scanner.nextInt();
        System.out.println((double)Math.round(heron(a) * 10000d) / 10000d);
    }

    public static int ClosetNumber(int a)   {
        int i;
    a = a - 1;
    while (a != 0) {

        for (i = 1; i * i <= a; i++)
        {

            if (i * i == a)
                return a;
        }
        a = a - 1;
    }
    return 0;
}
public static double heron(int x)
{
```

```
        double a, i;
        a = ClosetNumber(x);
        for (i = 0; i < 4; i++)
            a = 0.5 * (a + x / a);
        return a;
    }
}
```

Output :



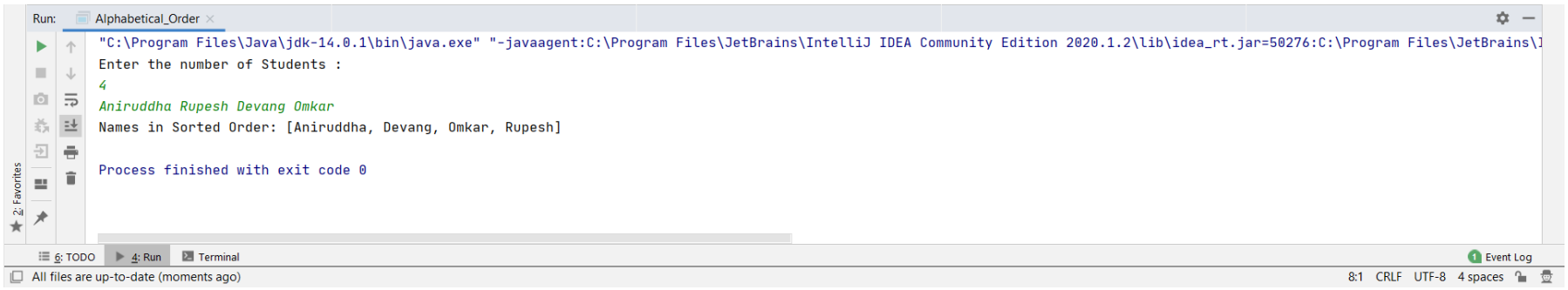
2.9 Write a program to sort and print the names of students taken from command line in alphabetical order.

Code :

```
import java.util.Scanner;
public class Alphabetical_Order
{
   public static void main(String[] args)
   {
      int n;
      String temp;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter number of names you want to enter:");
      n = s.nextInt();
      String names[] = new String[n];
      Scanner s1 = new Scanner(System.in);
      System.out.println("Enter all the names:");
      for(int i = 0; i < n; i++)
         names[i] = s1.nextLine();

      for (int i = 0; i < n; i++)     {
         for (int j = i + 1; j < n; j++)        {
            if (names[i].compareTo(names[j])>0)      {
               temp = names[i];
               names[i] = names[j];
               names[j] = temp;
            }
         }
      }
```

```
        System.out.print("Names in Sorted Order:");
        for (int i = 0; i < n - 1; i++)
        {
            System.out.print(names[i] + ",");
        }
        System.out.print(names[n - 1]);
    }
}
```

Output:



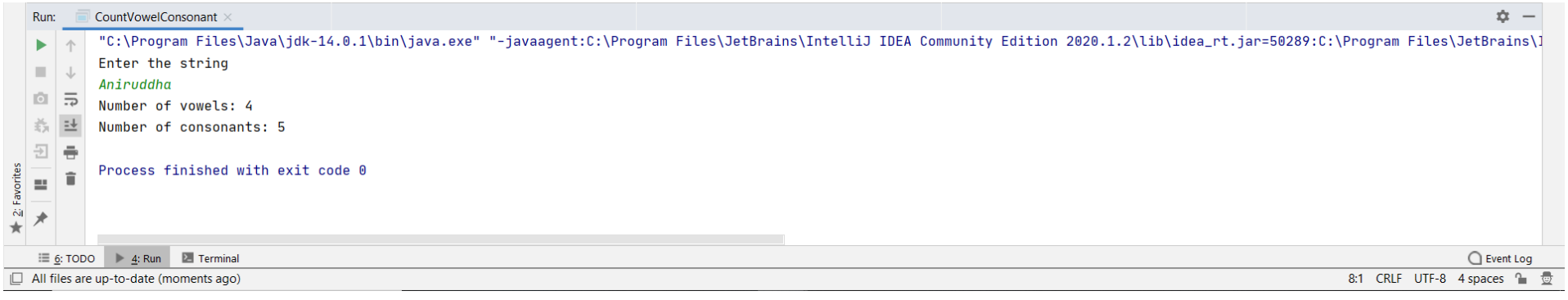
2.10 Write a program to print total numbers of vowels and consonants in a given string.

Code:

```
import java.util.Scanner;



public class CountVowelConsonant {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int vCount = 0, cCount = 0;

        System.out.println("Enter the string ");
        String str = sc.nextLine();

        str = str.toLowerCase();
        for(int i = 0; i < str.length(); i++) {

            if(str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i' || str.charAt(i) == 'o' ||
str.charAt(i) == 'u')
                vCount++;

            else if(str.charAt(i) >= 'a' && str.charAt(i)<='z')
                cCount++;

        }
```

```
        System.out.println("Number of vowels: " + vCount);
        System.out.println("Number of consonants: " + cCount);
    }
}
```

Output:



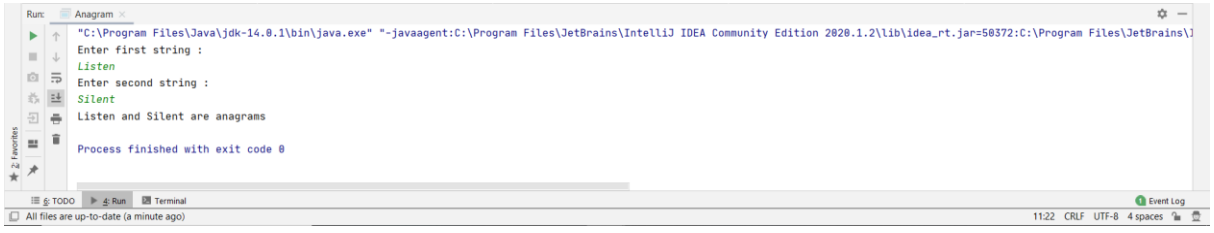
2.11 Given two English words, write a program to check if the first word is anagram of the second word. (An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. (Example: Anagram of TOM MARVOLO RIDDLE is I AM LORD VOLDEMORT.)

Code:

```
import java.util.Arrays;
import java.util.Scanner;

public class Anagram {

    static void areAnagram(String str1, String str2) {
        String s1 = str1.replaceAll("\\s", "");
        String s2 = str2.replaceAll("\\s", "");

        boolean status = true;
        int n1 = s1.length();
        int n2 = s2.length();

        if (n1 != n2)
            status = false;

        char[] ArrayS1 = s1.toLowerCase().toCharArray();
        char[] ArrayS2 = s2.toLowerCase().toCharArray();

        Arrays.sort(ArrayS1);
        Arrays.sort(ArrayS2);

        status = Arrays.equals(ArrayS1, ArrayS2);
        if (status)
            System.out.println(str1 + " and " + str2 + " are anagrams");
        else
```

```
        System.out.println(str1 + " and " + str2 + " are not anagrams");
    }

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter first string :");
        String str1 = in.nextLine();
        System.out.println("Enter second string :");
        String str2 = in.nextLine();
            areAnagram(str1, str2);
        }
    }
```

Output:



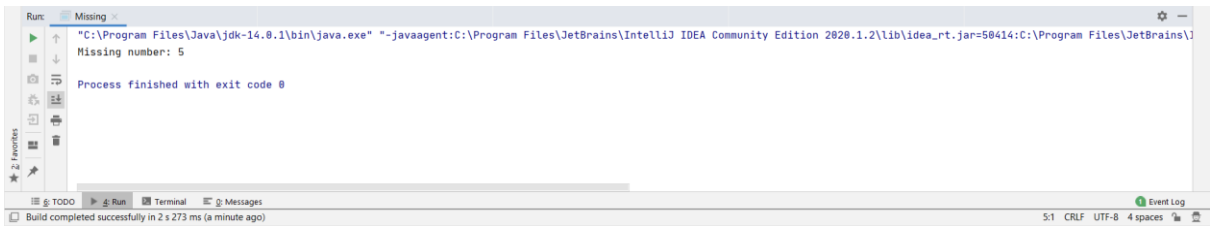
2.12 Write a program to print a missing number in a sorted integer array.

Code:

```
public class Missing {
    static int search(int arr1[], int size)
    {
        int a = 0, b = size - 1;
        int mid = 0;
        while ((b - a) > 1)
```

```
{
    mid = (a + b) / 2;
    if ((arr1[a] - a) != (arr1[mid] - mid))
        b = mid;
    else if ((arr1[b] - b) != (arr1[mid] - mid))
        a = mid;
}
return (arr1[mid] + 1);
    }
    public static void main (String[] args)
    {
        int array[] = { 1, 2, 3, 4, 6, 7, 8, 9, 10 };
        int size = array.length;
        System.out.println("Missing number: " + search(array, size));
    }


}
```

Output:

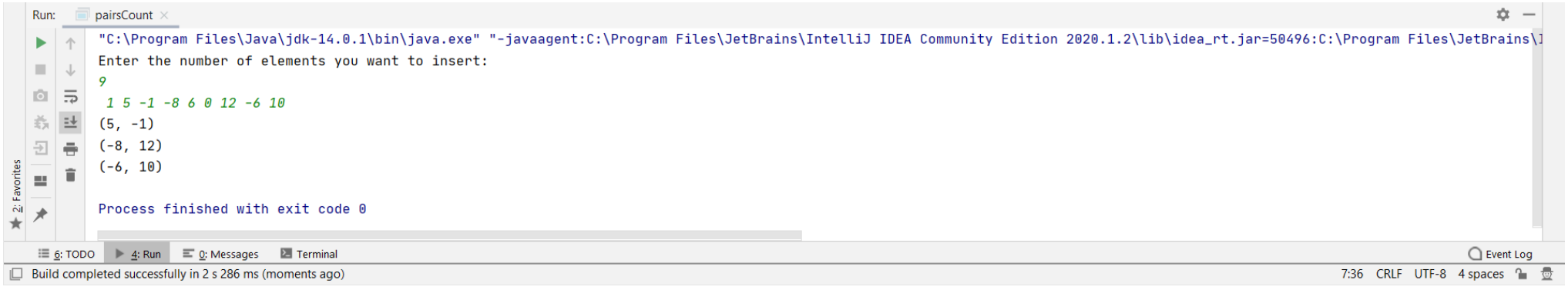


2.13 Write a program to find all the pairs of numbers on an integer array whose sum is equal to a given number.

Code:

```
import java.util.Scanner;


public class pairsCount {

   static void showPairs(int arr[], int n, int k) {

      for (int i = 0; i < n; i++)

         for (int j = i + 1; j < n; j++)

            if (arr[i] + arr[j] == k)

               System.out.println("(" + arr[i] + ", " + arr[j] + ")");
}


public static void main(String[] arg) {

   Scanner sc = new Scanner(System.in);

   System.out.println("Enter the number of elements you want to insert: ");

   int n = sc.nextInt();

   int arr[] = new int[n];

   for(int i=0; i<arr.length; i++){

      arr[i]=sc.nextInt();

      }

      int k = 4;

      showPairs(arr, n, k);

   }
}
```

Output:



Conclusion: We understood and performed various programs using Java.

Practical no. 3

FS19CO042

Aim:

3.1 Define the following classes/ interfaces with the help of above shortcuts:

1. Person(id, name, dateOfBirth, age, street, city, pin : default and parameterized constructors and setters and getters)

2. Department(id, name, dateOfEstablishment, headOfficeLocation, headId, numberOfEmployees : default and parameterized constructors and setters and getters)

3. Point(x, y, z : default and parameterized constructors and setters and getters)

4. Vehicle(registrationNumber, rcBookNumber, manufacturer, numberOfWheels, vehicleType, model, numberOfSeats : default and parameterized constructors and setters and getters)

5. Laptop(imeiNumber, processorName, processorSpeed, primaryMemoryType, primaryMemoryCapacity, secondaryStorageType, secondaryStorageCapaciry, screenResolution, screenType, isLED, listOfPorts, osInstalled : default and parameterized constructors and setters and getters)

6. interface Taxable(public int cost(), public intpercentGST())

3.2 Check whether feature of Encapsulation has been followed in 3.1. If not make necessary changes.

3.3 Define classes Car, Train and Truck with necessary member fields, constructors and methods. Make them extend class Vehicle.

3.4 Define a class Gadget with necessary member fields, constrictors and methods. Modify the class Laptop to extend the class Gadget.

3.5 In main method, declare a reference variable vehicle of class Vehicle and create an object of class Car which will be referenced by vehicle. Call getName() method on the object. (Hint: Reference Variable Casting)

3.6 Modify the classes Vehicle and Gadget implement the interface Taxable. Hence override respective methods.

3.7 Modify the classes Car and Laptop to override the implemented methods in 3.6.

3.8 Modify the class Gadget to add a data member gadgetCount such that its value will incremented as soon as a new object is initialized. Create 5 objects of the class Print its value after initializing each object.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects

- Instance
- Method
- Message Passing

Object − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Example

```
public class Dog {
   String breed;
   int age;
   String color;
   void barking() {
   }
   void hungry() {
   }
   void sleeping() {
   }
}
```

A class can contain any of the following variable types.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

Constructors

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor −

Example

```
public class Puppy {
   public Puppy() {
   }

   public Puppy(String name) {
      // This constructor has one parameter, name.
   }
}
```

Java also supports Singleton Classes where you would be able to create only one instance of a class.

Note − We have two different types of constructors. We are going to discuss constructors in detail in the subsequent chapters.

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface −

Example

Following is an example of an interface −

```
/* File name : NameOfInterface.java */

import java.lang.*;

// Any number of import statements

public interface NameOfInterface {

   // Any number of final, static fields

   // Any number of abstract method declarations\

}Interfaces have the following properties −
```

- An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example

```
/* File name : Animal.java */

interface Animal {

   public void eat();

   public void travel();

}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
/* File name : MammalInt.java */

public class MammalInt implements Animal {

   public void eat() {

      System.out.println("Mammal eats");

   }

   public void travel() {

      System.out.println("Mammal travels");

   }

   public int noOfLegs() {

      return 0;

   }

   public static void main(String args[]) {

      MammalInt m = new MammalInt();

      m.eat();

      m.travel();

   }

}
```

This will produce the following result −

Output

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces, there are several rules to be followed −

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules −

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.

- An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```
// Filename: Sports.java

public interface Sports {

   public void setHomeTeam(String name);

   public void setVisitingTeam(String name);

}



// Filename: Football.java

public interface Football extends Sports {

   public void homeTeamScored(int points);

   public void visitingTeamScored(int points);

   public void endOfQuarter(int quarter);

}

// Filename: Hockey.java

public interface Hockey extends Sports {

   public void homeGoalScored();

   public void visitingGoalScored();

   public void endOfPeriod(int period);

   public void overtimePeriod(int ot);

}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as −

Example

```
public interface Hockey extends Sports, Event
```

Code:

3.1 Define the following classes/ interfaces with the help of above shortcuts

1. Person(id, name, dateOfBirth, age, street, city, pin : default and parameterized constructors and setters and getters)

Code :

```
class Person {

    int dateOfBirth, age, id, pin;
    String name, street, city;

    Person() {

    }

    Person(int a, int b, int c, int d, String s, String s2, String s3) {
        this.dateOfBirth = c;
        this.age = b;
        this.id = a;
        this.pin = d;
        this.name = s;
        this.street = s2;
        this.city = s3;
    }

    public int getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(int dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public int getPin() {
        return pin;
    }

    public void setPin(int pin) {
        this.pin = pin;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    void setAge(int a) {
        age = a;
    }

}
```

2. Department(id, name, dateOfEstablishment, headOfficeLocation, headId, numberOfEmployees : default and parameterized constructors and setters and getters)

```
class Department {
    int id, headId, numberOfEmployees, dateOfEstablishment;
    String headOfficeLocation;
    String name;

    Department() {

    }

    public Department(int id, int headId, int numberOfEmployees, int dateOfEstablishment, String headOfficeLocation, String
name) {
        this.id = id;
        this.headId = headId;
        this.numberOfEmployees = numberOfEmployees;
```

```
        this.dateOfEstablishment = dateOfEstablishment;
        this.headOfficeLocation = headOfficeLocation;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getHeadId() {
        return headId;
    }

    public void setHeadId(int headId) {
        this.headId = headId;
    }

    public int getNumberOfEmployees() {
        return numberOfEmployees;
    }

    public void setNumberOfEmployees(int numberOfEmployees) {
        this.numberOfEmployees = numberOfEmployees;
    }

    public int getDateOfEstablishment() {
        return dateOfEstablishment;
    }

    public void setDateOfEstablishment(int dateOfEstablishment) {
        this.dateOfEstablishment = dateOfEstablishment;
    }

    public String getHeadOfficeLocation() {
        return headOfficeLocation;
    }

    public void setHeadOfficeLocation(String headOfficeLocation) {
        this.headOfficeLocation = headOfficeLocation;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


}
```

Point(x, y, z : default and parameterized constructors and setters and getters)

Code :

```
class Point {
    int x, y, z;

    Point() {
    }

    public Point(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getZ() {
        return z;
    }

    public void setZ(int z) {
        this.z = z;
    }
```

```
}
```

4. Vehicle(registrationNumber, rcBookNumber, manufacturer, numberOfWheels, vehicleType, model, numberOfSeats : default and parameterized constructors and setters and getters)

Code :

```
class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }
public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getRegistrationNumber() {
    return registrationNumber;
}

public void setRegistrationNumber(int registrationNumber) {
    this.registrationNumber = registrationNumber;
}

public int getRcBookNumber() {
    return rcBookNumber;
}

public void setRcBookNumber(int rcBookNumber) {
    this.rcBookNumber = rcBookNumber;
}

public int getManufacturer() {
    return manufacturer;
}
public void setManufacturer(int manufacturer) {
    this.manufacturer = manufacturer;
}

public int getNumberOfWheels() {
    return numberOfWheels;
}

public void setNumberOfWheels(int numberOfWheels) {
    this.numberOfWheels = numberOfWheels;
}

public int getNumberOfSeats() {
    return numberOfSeats;
}

public void setNumberOfSeats(int numberOfSeats) {
    this.numberOfSeats = numberOfSeats;
}

public String getVehicleType() {
    return vehicleType;
}

public void setVehicleType(String vehicleType) {
    this.vehicleType = vehicleType;
}

public String getModel() {
    return model;
```

```
    }

    public void setModel(String model) {
        this.model = model;
    }


    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

5. Laptop(imeiNumber, processorName, processorSpeed, primaryMemoryType, primaryMemoryCapacity, secondaryStorageType, secondaryStorageCapaciry, screenResolution, screenType, isLED, listOfPorts, osInstalled : default and parameterized constructors and setters and getters)

Code :

```
class Laptop extends Gadget {
    int imeiNumber;
    String processorName, primaryMemoryType, secondaryStorageType, screenType;
    boolean isLED, osInstalled;
    float processorSpeed, primaryMemoryCapacity, secondaryStorageCapaciry, screenResolution;
    String listOfPorts;
    int cost;

    Laptop() {

    }

    public Laptop(int imeiNumber, String processorName, String primaryMemoryType, String secondaryStorageType, String
screenType, boolean isLED, boolean osInstalled, float processorSpeed, float primaryMemoryCapacity, float
secondaryStorageCapaciry, float screenResolution, String listOfPorts) {
        this.imeiNumber = imeiNumber;
        this.processorName = processorName;
        this.primaryMemoryType = primaryMemoryType;
        this.secondaryStorageType = secondaryStorageType;
        this.screenType = screenType;
        this.isLED = isLED;
        this.osInstalled = osInstalled;
        this.processorSpeed = processorSpeed;
        this.primaryMemoryCapacity = primaryMemoryCapacity;
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
        this.screenResolution = screenResolution;
        this.listOfPorts = listOfPorts;

    }

    public boolean isLED() {
        return isLED;
    }

    public void setLED(boolean LED) {
        isLED = LED;
    }

    public boolean isOsInstalled() {
        return osInstalled;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public int getImeiNumber() {
        return imeiNumber;
    }

    public void setImeiNumber(int imeiNumber) {
        this.imeiNumber = imeiNumber;
    }

    public String getProcessorName() {
        return processorName;
    }

    public void setProcessorName(String processorName) {
        this.processorName = processorName;
    }
```

```
public String getPrimaryMemoryType() {
    return primaryMemoryType;
}

public void setPrimaryMemoryType(String primaryMemoryType) {
    this.primaryMemoryType = primaryMemoryType;
}

public String getSecondaryStorageType() {
    return secondaryStorageType;
}

public void setSecondaryStorageType(String secondaryStorageType) {
    this.secondaryStorageType = secondaryStorageType;
}

public String getScreenType() {
    return screenType;
}

public void setScreenType(String screenType) {
    this.screenType = screenType;
}

public boolean getIsLED() {
    return isLED;
}

public void setIsLED(Boolean isLED) {
    this.isLED = isLED;
}
```

```
public boolean getOsInstalled() {
    return osInstalled;
}

public void setOsInstalled(boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public void setOsInstalled(Boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public float getProcessorSpeed() {
    return processorSpeed;
}

public void setProcessorSpeed(float processorSpeed) {
    this.processorSpeed = processorSpeed;
}

public float getPrimaryMemoryCapacity() {
    return primaryMemoryCapacity;
}

public void setPrimaryMemoryCapacity(float primaryMemoryCapacity) {
    this.primaryMemoryCapacity = primaryMemoryCapacity;
}

public float getSecondaryStorageCapaciry() {
    return secondaryStorageCapaciry;
}

public void setSecondaryStorageCapaciry(float secondaryStorageCapaciry) {
    this.secondaryStorageCapaciry = secondaryStorageCapaciry;
}
```

```
    public float getScreenResolution() {
        return screenResolution;
    }

    public void setScreenResolution(float screenResolution) {
        this.screenResolution = screenResolution;
    }

    public String getListOfPorts() {
        return listOfPorts;
    }

    public void setListOfPorts(String listOfPorts) {
        this.listOfPorts = listOfPorts;
    }

    void print() {
        System.out.println("emi no is " + getImeiNumber() + "\n Processor name is: " + getProcessorName() + "\n led :" +
getIsLED() + "\n Ports are :" + getListOfPorts() + "\n OS :" + getOsInstalled() + "\n Meomory Capacity :" +
getPrimaryMemoryCapacity());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
```

```
        return percentGST;
    }

}
```

6. interface Taxable(public int cost(), public intpercentGST())

Code :

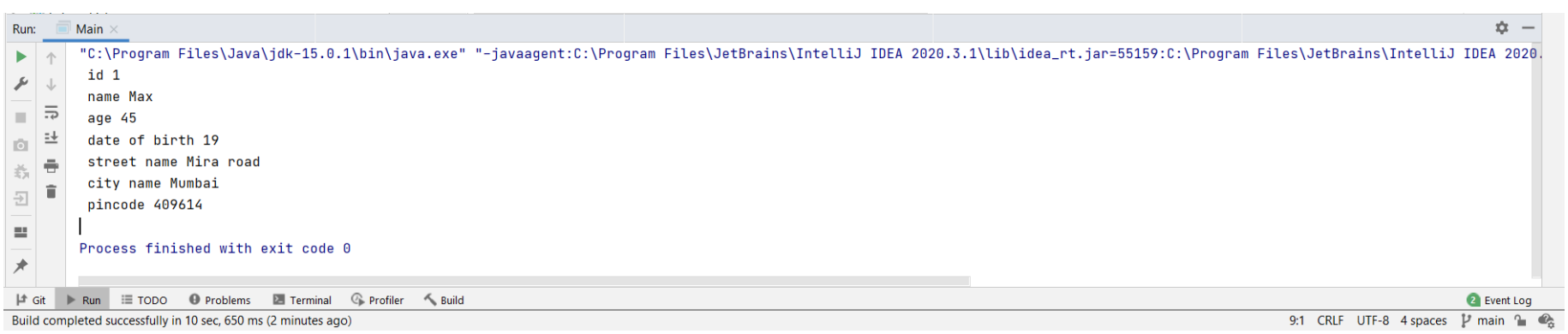
```
interface Taxable {
    int cost();

    int percentGST();
}
```

3.2 Check whether feature of Encapsulation has been followed in 3.1. If not make necessary changes.

Code :

```
public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setAge(45);
        person.setId(01);
        person.setName("Max");
        person.setDateOfBirth(19);
        person.setStreet("Mira road");
        person.setCity("Mumbai");
        person.setPin(409614);
        System.out.println(" id " + person.getId() + "\n name " + person.getName() + "\n age " + person.getAge() + "\n date of
birth " + person.getDateOfBirth() + "\n street name " + person.getStreet() + "\n city name " + person.getCity() + "\n pincode "
+ person.getPin());
    }
}
```

Output :



3.3 Define classes Car, Train and Truck with necessary member fields, constructors and methods. Make them extend class Vehicle.

Code :

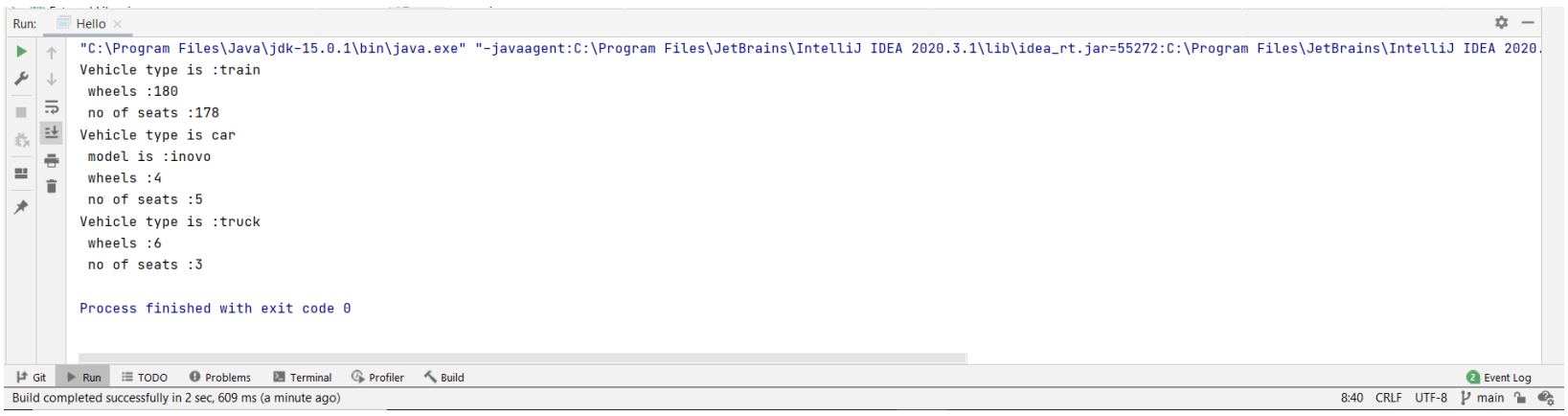
```
class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }
```

```
    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}



class Train extends Vehicle {
    void show() {
        System.out.println("vehicle type is :" + getVehicleType() + "\n wheels :" + getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }


}



class Truck extends Vehicle {


    void show() {
        System.out.println("vehicle type is :" + getVehicleType() + "\n wheels :" + getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }
}
```

Main method :

```
public static void main(String[] args) {

    Train h = new Train();
    h.setVehicleType("train");
    h.setNumberOfSeats(178);
    h.setNumberOfWheels(180);
    h.setRegistrationNumber(90764);
    h.show();

    Car c = new Car();
    c.setVehicleType("car");
    c.setModel("inovo");
    c.setNumberOfSeats(5);
    c.setNumberOfWheels(4);
    c.setRegistrationNumber(90671);
    c.show();

    Truck t = new Truck();
    t.setVehicleType("truck");
    t.setNumberOfSeats(3);
    t.setNumberOfWheels(6);
    t.show();
}
```

Output :



3.4 Define a class Gadget with necessary member fields, constrictors and methods. Modify the class Laptop to extend the class Gadget.

Code :

```
public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
```

```
    this.gadgetName = gadgetName;
}

void disp() {
    System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
}

public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }

}
```

Main method :

```
public static void main(String args[]) {
    Laptop l = new Laptop();
    l.setGadgetName("Laptop");
    l.setImeiNumber(2345);
    l.setIsLED(true);
    l.setListOfPorts("2 USB Port,1 Charging port,1 Pendrive Port");
    l.setOsInstalled(true);
    l.setPrimaryMemoryCapacity(1500);
    l.setProcessorName("intel core i5");
    l.Show();
    l.print();
}
```

Output :

3.5 In main method, declare a reference variable vehicle of class Vehicle and create an object of class Car which will be referenced by vehicle. Call getName() method on the object. (Hint: Reference Variable Casting)

Code :

```
class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
```

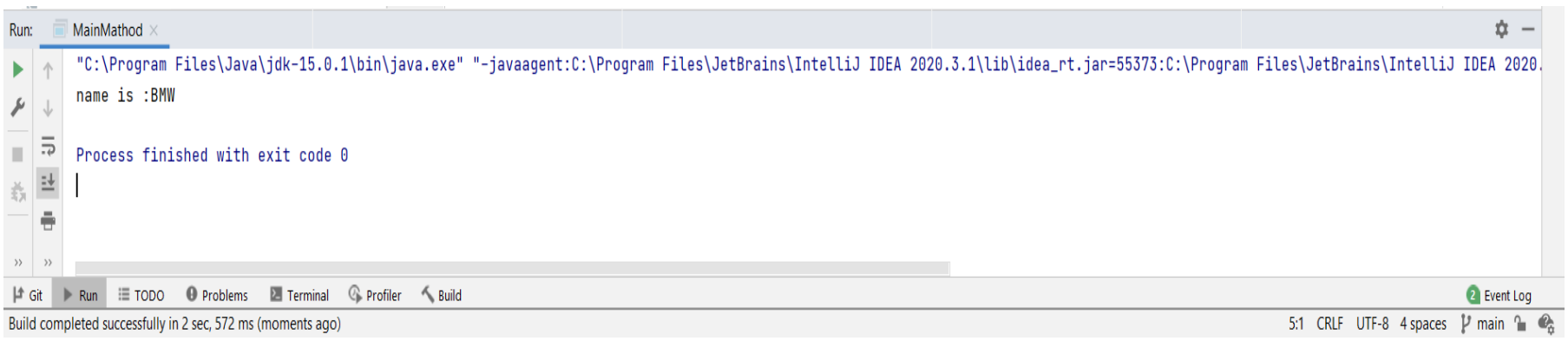
```
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getRegistrationNumber() {
        return registrationNumber;
    }

    public void setRegistrationNumber(int registrationNumber) {
        this.registrationNumber = registrationNumber;
    }

    public int getRcBookNumber() {
        return rcBookNumber;
    }

    public void setRcBookNumber(int rcBookNumber) {
        this.rcBookNumber = rcBookNumber;
    }

    public int getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(int manufacturer) {
        this.manufacturer = manufacturer;
    }

    public int getNumberOfWheels() {
        return numberOfWheels;
    }

    public void setNumberOfWheels(int numberOfWheels) {
        this.numberOfWheels = numberOfWheels;
    }

    public int getNumberOfSeats() {
        return numberOfSeats;
    }

    public void setNumberOfSeats(int numberOfSeats) {
        this.numberOfSeats = numberOfSeats;
    }

    public String getVehicleType() {
        return vehicleType;
    }

    public void setVehicleType(String vehicleType) {
        this.vehicleType = vehicleType;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }


    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

```
class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("Vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}
```

Main method :

```
public static void main(String [] args)
{
    Vehicle vehicle;
    Car c1=new Car();
    c1.setName("BMW");
    c1.disp();

}
```

Output :



3.6 Modify the classes Vehicle and Gadget implement the interface Taxable. Hence override respective methods.

Code :

```
interface Taxable {
    int cost();

    int percentGST();
}

public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
        this.gadgetName = gadgetName;
    }

    void disp() {
        System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
    }
```

```
public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
}

public int percentGST() {
    float a = 0.18f;
    int percentGST = (int) (getCost() + (getCost() * a));
    return percentGST;
    }

}


class Vehicle implements Taxable {
    int registrationNumber, rcBookNumber, manufacturer, numberOfWheels, numberOfSeats;
    String vehicleType, model, name;
    int cost;

    Vehicle() {

    }

    public Vehicle(int registrationNumber, int rcBookNumber, int manufacturer, int numberOfWheels, int numberOfSeats, String
vehicleType, String model) {
        this.registrationNumber = registrationNumber;
        this.rcBookNumber = rcBookNumber;
        this.manufacturer = manufacturer;
        this.numberOfWheels = numberOfWheels;
        this.numberOfSeats = numberOfSeats;
        this.vehicleType = vehicleType;
        this.model = model;
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        this.cost = cost;
    }

    public String getName() {
        return name;
}

public void setName(String name) {
    this.name = name;
}

public int getRegistrationNumber() {
    return registrationNumber;
}

public void setRegistrationNumber(int registrationNumber) {
    this.registrationNumber = registrationNumber;
}

public int getRcBookNumber() {
    return rcBookNumber;
}

public void setRcBookNumber(int rcBookNumber) {
    this.rcBookNumber = rcBookNumber;
}

public int getManufacturer() {
    return manufacturer;
}

public void setManufacturer(int manufacturer) {
    this.manufacturer = manufacturer;
}

public int getNumberOfWheels() {
```

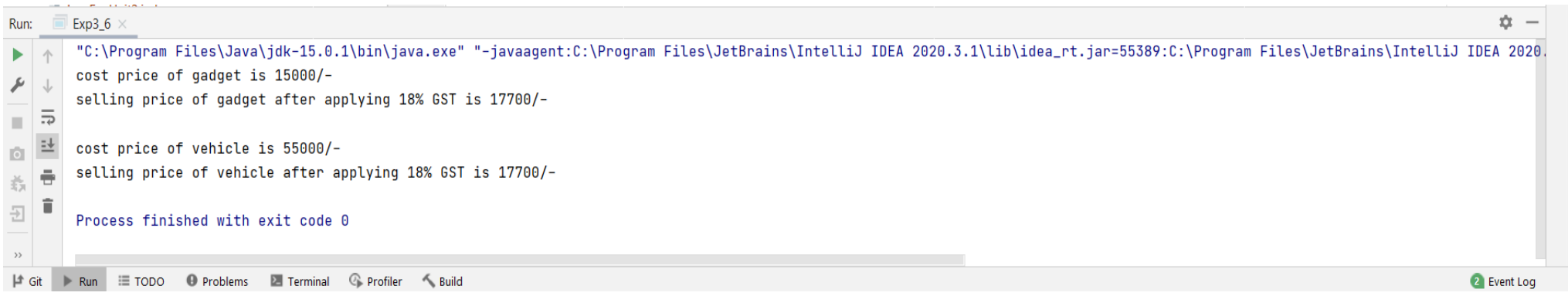
```
    return numberOfWheels;
}

public void setNumberOfWheels(int numberOfWheels) {
    this.numberOfWheels = numberOfWheels;
}

public int getNumberOfSeats() {
    return numberOfSeats;
}

public void setNumberOfSeats(int numberOfSeats) {
    this.numberOfSeats = numberOfSeats;
}

public String getVehicleType() {
    return vehicleType;
}

public void setVehicleType(String vehicleType) {
    this.vehicleType = vehicleType;
}

public String getModel() {
    return model;
}

public void setModel(String model) {
    this.model = model;
}
```

```
    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }


}
```

Main method :

```
public static void main(String args[]) {
    Gadget g = new Gadget();
    g.setCost(15000);
    Vehicle v = new Vehicle();
    v.setCost(55000);
    System.out.println("cost price of gadget is " + g.cost() + "/-");
    System.out.println("selling price of gadget after applying 18% GST is " + g.percentGST() + "/-");
    System.out.println();
    System.out.println("cost price of vehicle is " + v.cost() + "/-");
    System.out.println("selling price of vehicle after applying 18% GST is " + g.percentGST() + "/-");

}
```

Output :



3.7 Modify the classes Car and Laptop to override the implemented methods in 3.6.

Code :

```
class Laptop extends Gadget {
    int imeiNumber;
    String processorName, primaryMemoryType, secondaryStorageType, screenType;
    boolean isLED, osInstalled;
    float processorSpeed, primaryMemoryCapacity, secondaryStorageCapaciry, screenResolution;
    String listOfPorts;
    int cost;

    Laptop() {

    }

    public Laptop(int imeiNumber, String processorName, String primaryMemoryType, String secondaryStorageType, String
```

```
screenType, boolean isLED, boolean osInstalled, float processorSpeed, float primaryMemoryCapacity, float
secondaryStorageCapaciry, float screenResolution, String listOfPorts) {
        this.imeiNumber = imeiNumber;
        this.processorName = processorName;
        this.primaryMemoryType = primaryMemoryType;
        this.secondaryStorageType = secondaryStorageType;
        this.screenType = screenType;
        this.isLED = isLED;
        this.osInstalled = osInstalled;
        this.processorSpeed = processorSpeed;
        this.primaryMemoryCapacity = primaryMemoryCapacity;
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
        this.screenResolution = screenResolution;
        this.listOfPorts = listOfPorts;

    }

    public boolean isLED() {
        return isLED;
    }

    public void setLED(boolean LED) {
        isLED = LED;
    }

    public boolean isOsInstalled() {
        return osInstalled;
    }

    public int getCost() {
        return cost;
    }

public void setCost(int cost) {
    this.cost = cost;
}

public int getImeiNumber() {
    return imeiNumber;
}

public void setImeiNumber(int imeiNumber) {
    this.imeiNumber = imeiNumber;
}

public String getProcessorName() {
    return processorName;
}

public void setProcessorName(String processorName) {
    this.processorName = processorName;
}

public String getPrimaryMemoryType() {
    return primaryMemoryType;
}

public void setPrimaryMemoryType(String primaryMemoryType) {
    this.primaryMemoryType = primaryMemoryType;
}

public String getSecondaryStorageType() {
    return secondaryStorageType;
}

public void setSecondaryStorageType(String secondaryStorageType) {
    this.secondaryStorageType = secondaryStorageType;
}

public String getScreenType() {
    return screenType;
}

public void setScreenType(String screenType) {
    this.screenType = screenType;
}

public boolean getIsLED() {
    return isLED;
}

public void setIsLED(Boolean isLED) {
    this.isLED = isLED;
}

public boolean getOsInstalled() {
    return osInstalled;
}

public void setOsInstalled(boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public void setOsInstalled(Boolean osInstalled) {
    this.osInstalled = osInstalled;
}

public float getProcessorSpeed() {
```

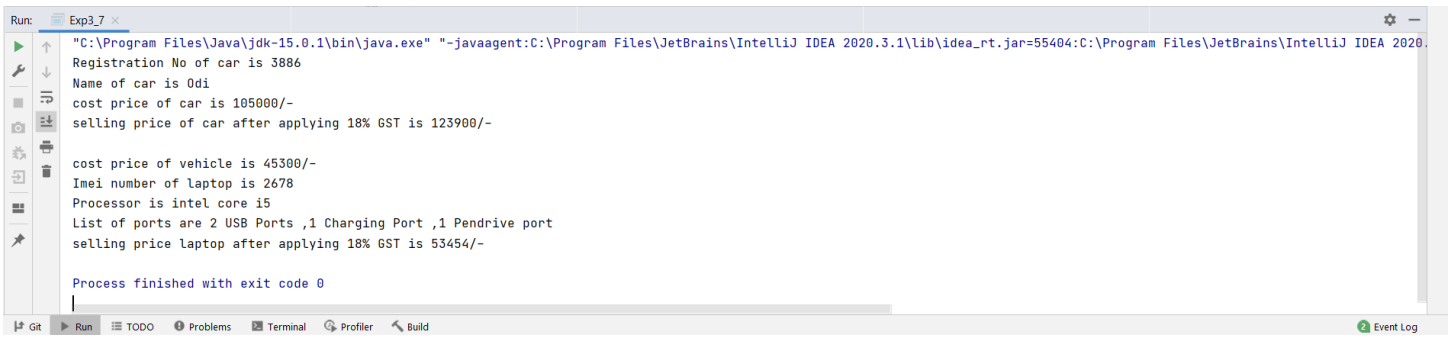
```
        return processorSpeed;
    }

    public void setProcessorSpeed(float processorSpeed) {
        this.processorSpeed = processorSpeed;
    }

    public float getPrimaryMemoryCapacity() {
        return primaryMemoryCapacity;
    }

    public void setPrimaryMemoryCapacity(float primaryMemoryCapacity) {
        this.primaryMemoryCapacity = primaryMemoryCapacity;
    }

    public float getSecondaryStorageCapaciry() {
        return secondaryStorageCapaciry;
    }

    public void setSecondaryStorageCapaciry(float secondaryStorageCapaciry) {
        this.secondaryStorageCapaciry = secondaryStorageCapaciry;
    }

    public float getScreenResolution() {
        return screenResolution;
    }

    public void setScreenResolution(float screenResolution) {
        this.screenResolution = screenResolution;
    }

    public String getListOfPorts() {
        return listOfPorts;
    }

    public void setListOfPorts(String listOfPorts) {
        this.listOfPorts = listOfPorts;
    }

    void print() {
        System.out.println("emi no is " + getImeiNumber() + "\n Processor name is: " + getProcessorName() + "\n led :" +
getIsLED() + "\n Ports are :" + getListOfPorts() + "\n OS :" + getOsInstalled() + "\n Meomory Capacity :" +
getPrimaryMemoryCapacity());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }

}


class Car extends Vehicle {
    int cost;

    @Override
    public int getCost() {
        return cost;
    }

    @Override
    public void setCost(int cost) {
        this.cost = cost;
    }

    void show() {
        System.out.println("Vehicle type is " + getVehicleType() + "\n model is :" + getModel() + "\n wheels :" +
getNumberOfWheels()
                + "\n no of seats :" + getNumberOfSeats() + "");
    }

    void disp() {
        System.out.println("name is :" + getName());
    }

    public int cost() {
        int cost = getCost();
        return cost;
    }

    public int percentGST() {
        float a = 0.18f;
        int percentGST = (int) (getCost() + (getCost() * a));
        return percentGST;
    }
}
```

Main method :

```
public static void main(String args[])
{
```

```
    Car c=new Car();
    c.setName("Odi");
    c.setCost(105000);
    c.setRegistrationNumber(07456);
    Laptop l=new Laptop();
    l.setCost(45300);
    l.setProcessorName("intel core i5");
    l.setImeiNumber(2678);
    l.setListOfPorts("2 USB Ports ,1 Charging Port ,1 Pendrive port ");
    System.out.println("Registration No of car is "+c.getRegistrationNumber()+"\nName of car is "+ c.getName()+"\ncost price of
car is "+c.cost()+"/-");
    System.out.println("selling price of car after applying 18% GST is "+c.percentGST()+"/-");
    System.out.println();
    System.out.println("cost price of vehicle is "+l.cost()+"/-");
    System.out.println("Imei number of laptop is "+l.getImeiNumber()+"\nProcessor is "+l.getProcessorName()+"\nList of ports are
"+l.getListOfPorts()+"\nselling price laptop after applying 18% GST is "+l.percentGST()+"/-");


}
```

Output :



3.8 Modify the class Gadget to add a data member gadgetCount such that its value will incremented as soon as a new object is initialized. Create 5 objects of the class Print its value after initializing each object.

Code :

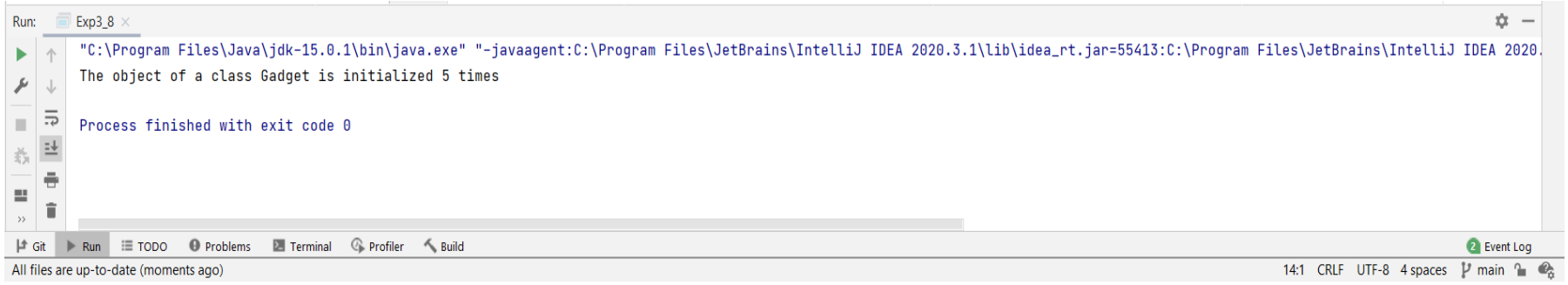
```
public class Gadget implements Taxable {
    static int gadgetcount = 0;
    String gadgetName;
    int cost;

    {
        gadgetcount += 1;
    }

    Gadget() {

    }

    public Gadget(String gadgetName) {
        this.gadgetName = gadgetName;
    }

    void disp() {
        System.out.println("The object of a class Gadget is initialized " + gadgetcount + " times");
    }

public String getGadgetName() {
    return gadgetName;
}

public void setGadgetName(String gadgetName) {
    this.gadgetName = gadgetName;
}

public int getCost() {
    return cost;
}

public void setCost(int cost) {
    this.cost = cost;
}

void Show() {
    System.out.println("This is gadget :" + getGadgetName());

}


public int cost() {
    int cost = getCost();
    return cost;
}

public int percentGST() {
    float a = 0.18f;
    int percentGST = (int) (getCost() + (getCost() * a));
    return percentGST;
}
```

```
}
```

Main method :

```
   public static void main(String args[])
    {
        Gadget g=new Gadget();
        Gadget g1=new Gadget();
        Gadget g2=new Gadget();
        Gadget g3=new Gadget();
        Gadget g4=new Gadget();
        g.disp();
    }
}
```

Output :



Conclusion: Thus, we understood and executed various programs using classes, interfaces, etc. and explored various concepts related to these topics.

Practical no. 4

FS19CO042

Aim:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
  }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. javac -d directory javafilename

For **example**

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
```

```
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Code:

4.1 Create a package com.gpm.complex. Create an interface Complex in it with following member methods: realPart(), imgPart(), magnitude() and argument() along with default methods plus(), minus(), into() and divideBy() having appropriate parameters and return types.

Code :

```
package com.gpm.complex;

public interface Complex {
    void realPart();

    void imgPart();

    void magnitude();

    void argument();

    default float plus(float a, float b) {
        return a + b;
    }

    default float minus(float a, float b) {
        return a - b;
    }

    default float into(float a, float b) {
        return a * b;
    }

    default float divideBy(float a, float b) {
        return a / b;
    }

}
```

4.2 In the same package create class CartesianComplex with real and img and class PolarComplex with r and theta as their member fields. Make the classes implement the Complex interface. Override all non-default methods in the interface. Also override toString().

```
package com.gpm.complex;

public class CartesianComplex implements Complex {
    CartesianComplex real;
    CartesianComplex img;

    @Override
    public String toString() {
        return "CartesianComplex";
    }

    @Override
    public void realPart() {

    }

    @Override
    public void imgPart() {

    }
```

```
    @Override
    public void magnitude() {

    }

    @Override
    public void argument() {

    }

}
```

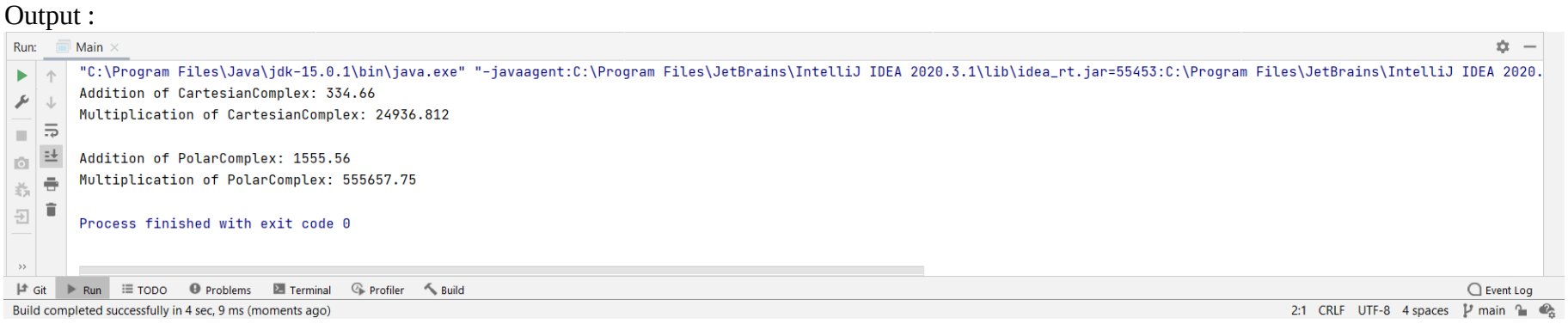
```
package com.gpm.complex;

public class PolarComplex implements Complex {
    PolarComplex r;
    PolarComplex theta;

    @Override
    public String toString() {
        return "PolarComplex";
    }

    @Override
    public void realPart() {

    }

    @Override
    public void imgPart() {

    }

    @Override
    public void magnitude() {

    }

    @Override
    public void argument() {

    }
}
```

4.3 Now in main(), create one objects of both the classes defined in 4.2 and print their addition and multiplication.

Code :

```
package com.gpm.complex;

public class Main {

    public static void main(String[] args) {

        CartesianComplex cartesianComplex = new CartesianComplex();
        PolarComplex polarComplex = new PolarComplex();

        System.out.println("Addition of CartesianComplex: "+cartesianComplex.plus(111.99f, 222.67f));
        System.out.println("Multiplication of CartesianComplex: "+cartesianComplex.into(111.99f, 222.67f)+"\n");

        System.out.println("Addition of PolarComplex: "+polarComplex.plus(555.78f, 999.78f));
        System.out.println("Multiplication of PolarComplex: "+polarComplex.into(555.78f, 999.78f));

    }
}
```

Output :



4.4 Create a Java swing frame by creating a subclass of javax.swing.JFrame class. Add a java.awt.event.MouseListener by passing an object of an anonymous subclass of java.awt.event.MouseAdapter on the JFrame. Display the coordinates of point at which mouse is clicked.

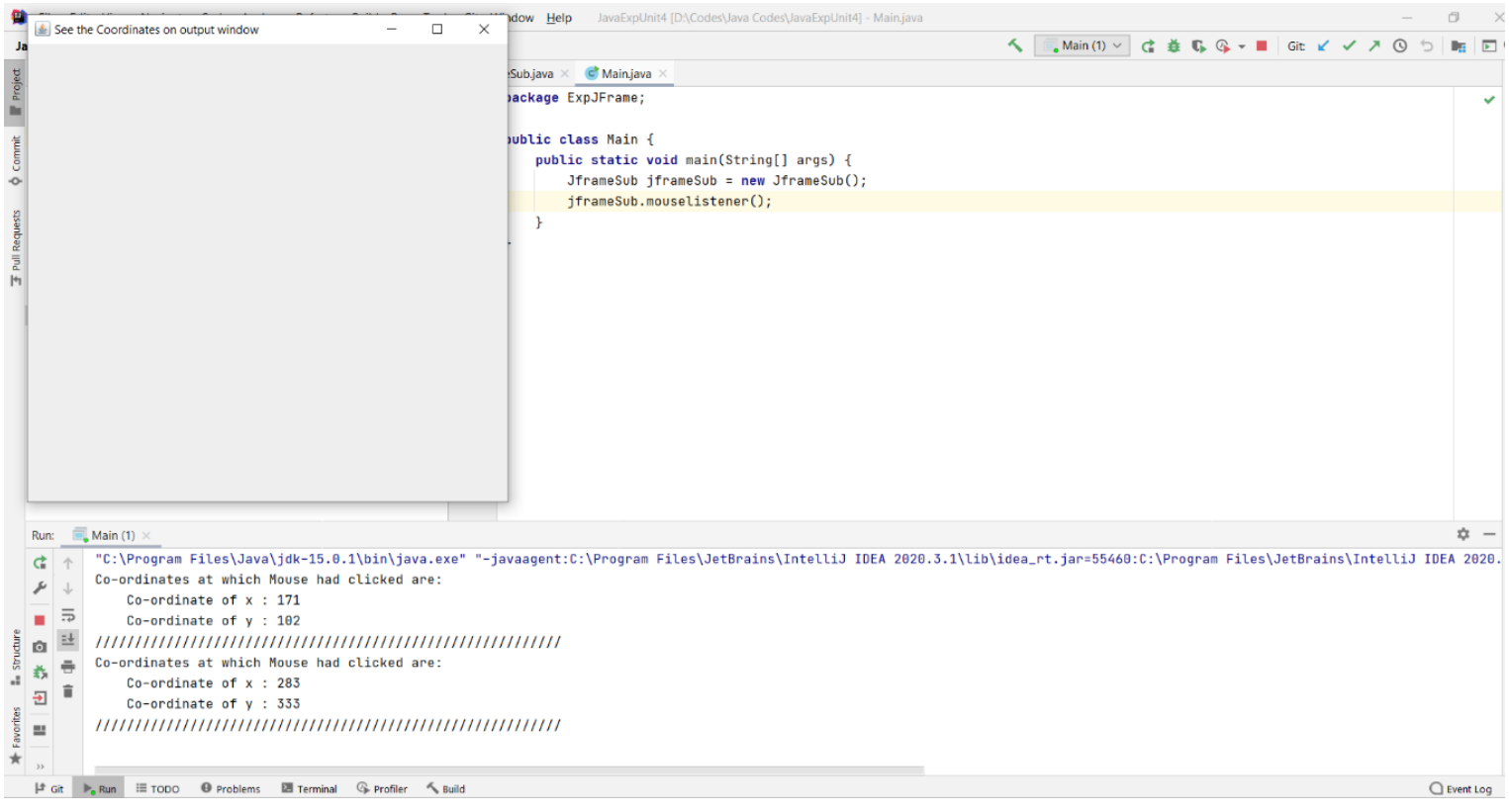
Code :

```
package ExpJFrame;

import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class JframeSub extends JFrame {

    public JframeSub() {

    }

    public void mouselistener() {

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                int x = e.getX();
                int y = e.getY();
                System.out.println("Co-ordinates at which Mouse had clicked are: \n" +
                        "\tCo-ordinate of x : " + x +
                        "\n\tCo-ordinate of y : " + y);
                System.out.println("////////////////////////////////////////////////////////////");
            }
        });

        setTitle("See the Coordinates on output window");
        setLayout(null);
        setVisible(true);
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }
}
```

Main method :

```
package ExpJFrame;
public class Main {
    public static void main(String[] args) {
        JframeSub jframeSub = new JframeSub();
        jframeSub.mouselistener();
    }
}
```

Output :



Conclusion: Thus, we understood and executed programs regarding interfaces and swing framework.

Practical no. 5

FS19CO042

Aim: Using Stream API implement following programs.

- 5.1 Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).
- 5.2 Write a method which takes a list of words as an argument, groups the words by their lengths and returns the groupings in the form of Map>. (The keys in the map are the lengths and the values are the lists of words of that length.)
- 5.3 Given a List<List> write a program to convert it into a List. (Hint: Use flatMap method in Stream interface)
- 5.4 Given: class Album{ public final String name; public final int yearOfRelease; public final List tracks; }
  class Track{ public final int rating; }
  a) Write a method which takes a list of albums as an argument and returns a list of names of all albums sorted by the year of release.
  b) Write a method which takes a list of albums as an argument and returns a list of names of all albums containing at least one track having rating more than four. The returned list should be sorted by the year of release.

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Stream In Java

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

- A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
- Streams don't change the original data structure, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Different Operations On Streams-

Intermediate Operations:

1. **map:** The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.

```
List number = Arrays.asList(2,3,4,5);
List square = number.stream().map(x->x*x).collect(Collectors.toList());
```

2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
```

3. **sorted:** The sorted method is used to sort the stream.

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().sorted().collect(Collectors.toList());
```

Terminal Operations:

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.

```
List number = Arrays.asList(2,3,4,5,3);
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

2. **forEach:** The forEach method is used to iterate through every element of the stream.

```
List number = Arrays.asList(2,3,4,5);
number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

3. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.

   The reduce method takes a BinaryOperator as a parameter.

```
List number = Arrays.asList(2,3,4,5);
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);
```

   Here ans variable is assigned 0 as the initial value and i is added to it .

Code:

- **5.1 Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes).**

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;


public class  exp5_1 {

static class Student {
   String name = "";
   public int roll = 0;
   public int marks = 0;
   public Student(String name, int roll, int marks) {
      this.name = name;
      this.roll = roll;
      this.marks = marks;
   }
}

public static <T extends  Number> long  evenNumbers(List<T> list) {
   Stream<T> stream = list.stream();
   return stream.filter(number -> number.doubleValue() % 2 != 0).count();
```

```
}

public static <T extends Student> long  numberOfPassedStudents(List<? extends Student> list) {
   Stream<T> stream = (Stream<T>) list.stream();
   return stream.filter(student -> student.marks >= 35).count();

}

public static void main(String[] args) {

   Student s1 = new Student("Roy", 43, 60);
   Student s2 = new Student("Niel", 44, 49);
   Student s3 = new Student("Leo", 30, 75);
   Student s4 = new Student("lisa", 35, 30);
      Student s5 = new Student("Russ", 40, 28);

      List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      List<Student> list1 = Arrays.asList(s1, s2, s3, s4, s5);

      long evenNumbers = evenNumbers(list) ;
      long numberOfPassedStudents = numberOfPassedStudents(list1) ;

      System.out.println("There are "+ evenNumbers +" even numbers.");
      System.out.println(numberOfPassedStudents+" students have passed the exam.");
   }
}
```

Output:

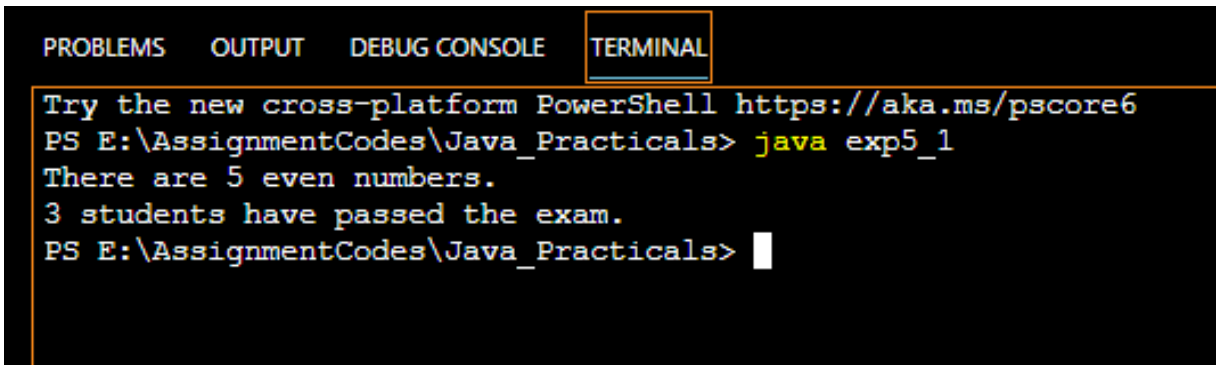
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS E:\AssignmentCodes\Java_Practicals> java exp5_1
There are 5 even numbers.
3 students have passed the exam.
PS E:\AssignmentCodes\Java_Practicals>
```

- **5.2 Write a method which takes a list of words as an argument, groups the words by their lengths and returns the groupings in the form of Map>. (The keys in the map are the lengths and the values are the lists of words of that length.)**

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

// main class and method
public class exp5_2{

   // main Driver
   private static Stream<String> stream;
```

```
    public static void main(String[] args)
    {

        // create a list

List<String> strings = Arrays.asList("this", "is", "a", "long", "list", "of",
        "strings", "to", "use", "as", "a", "trial");

stream = strings.stream();
Map<Integer, List<String>> lengthMap = stream.collect(Collectors.groupingBy(String::length));

lengthMap.forEach((k,v) -> System.out.printf("%d: %s%n", k, v));
    }
}
```

Output:



- **5.3 Given a List<List> write a program to convert it into a List. (Hint: Use flatMap method in Stream interface)**

```
import java.util.*;

import java.util.stream.*;

class exp5_3 {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList();

list1.add("Government");

list1.add("Polytechnic");

list1.add("Mumbai");


ArrayList<String> list2 = new ArrayList();

list2.add("A.Y. Jung Marg");

list2.add("Kherwadi");

list2.add("Bandra (E)");


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

listOflist.add(list1);

listOflist.add(list2);
```

```
        System.out.println("LIST1 : " + list1);
        System.out.println("LIST2 : " + list2);
        System.out.println("LIST<LIST<String>> :" + listOflist);

        ArrayList<String> result = new ArrayList();
        listOflist.forEach(result::addAll);

        System.out.println("RESULT : " + result);
    }
}
```

```
PROBLEMS 453   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: Code
PS E:\AssignmentCodes\Java_Practicals> java exp5_3
LIST1 : [Government, Polytechnic, Mumbai]
LIST2 : [A.Y. Jung Marg, Kherwadi, Bandra (E)]
LIST<LIST<String>> :[[Government, Polytechnic, Mumbai], [A.Y. Jung Marg, Kherwadi, Bandra (E)]]
RESULT : [Government, Polytechnic, Mumbai, A.Y. Jung Marg, Kherwadi, Bandra (E)]
PS E:\AssignmentCodes\Java_Practicals>
```

- **5.4 Given: class Album{ public final String name; public final int yearOfRelease; public final List tracks;**

  class Track{ public final int rating; }

  a) Write a method which takes a list of albums as an argument and returns a list of names of all albums sorted by the year of release.

  b) Write a method which takes a list of albums as an argument and returns a list of names of all albums containing at least one track having rating more than four. The returned list should be sorted by the year of release.

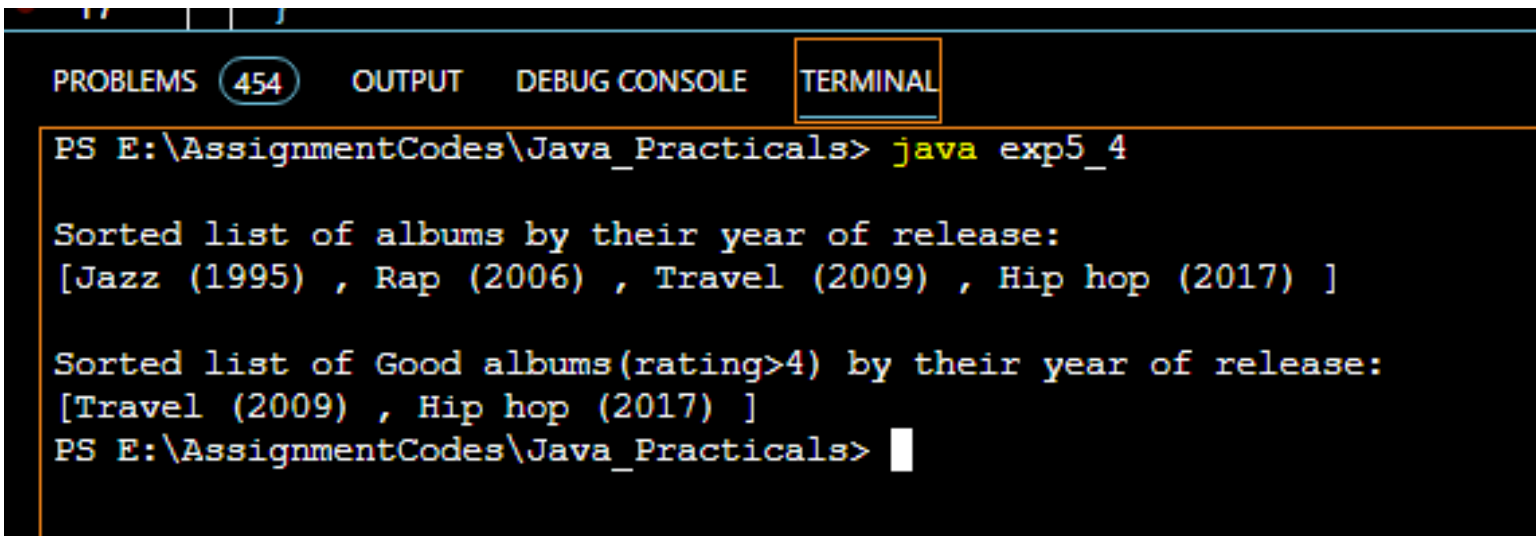
```
import java.util.*;
import java.util.stream.*;
public class exp5_4{
    static class Track{
        public final String name;
        public final int rating;
        public Track(String name, int rating){
            this.name = name;
            this.rating = rating;
        }
```

```
    public String toString(){
        return this.name + " (" + this.rating + ") ";
    }
}
static class Album{
    public final String name;
    private final List<Track> tracks;
    private int yearOfRelease;
    public int getYear(){
        return yearOfRelease;
    }
public List<Track> getTracks(){
    return tracks;
}
public int maxRating(){
    Track maxTrack = tracks.stream().reduce(new Track("temp",0), (maxTrackYet, currTrack) -> {
        if(maxTrackYet.rating < currTrack.rating)
            return currTrack;
        return maxTrackYet;
    });
    return maxTrack.rating;
}
public Album(String name, List<Track> trackList, int yearOfRelease){
    this.name = name;
        this.yearOfRelease = yearOfRelease;
        this.tracks = trackList;
    }
    public String toString(){
        return this.name+" ("+this.yearOfRelease+") ";
    }
}
static List<String>  sortAlbumsByYear(List<Album> albums){
    Stream albumStream = albums.stream();
    Object sorted[] = albumStream.sorted(Comparator.comparingInt(Album::getYear)).toArray();
```

```
    List<String> sortedList = new ArrayList<>();
    for(Object obj : sorted)
        sortedList.add(String.valueOf(obj));
    return sortedList;
}

static public List<String> filterGoodAlbums(List<Album> albums){
    List<String> goodAlbums = new ArrayList<>();
    albums.stream().forEach(album -> {
        if(album.maxRating() >4)
            goodAlbums.add(album.toString());
    });
     return goodAlbums;
}

public  static void main(String[] args) {
    System.out.println();
    // Preparing albums and tracks
    String[] travelNames = {"Ve maahi", "Duniya", "Bolna", "Kabira", "Vaaste"};
    int[] travelRatings= {5,6,4,8,5};
    List<Track> travelSongs = new  ArrayList<>();
    for(int i=0; i<travelNames.length; i++)
        travelSongs.add(new Track(travelNames[i], travelRatings[i]));
    Album travelAlbum = new Album("Travel",travelSongs, 2009);

String[] rapNames = {"Mirchi", "ChalBombay", "Kohinoor"};
int[] rapRatings = {1, 3,2};
List<Track> rapSongs = new  ArrayList<>();
for(int i=0; i<rapNames.length; i++)
        rapSongs.add(new Track(rapNames[i], rapRatings[i]));
Album rapAlbum = new Album("Rap",rapSongs, 2006);

String[] hipHopNames = {"Ve maahi", "Duniya", "Bolna", "Kabira", "Vaaste"};
int[] hiphopRatings = {5,9,9,4,7};
List<Track> hipHopSongs = new  ArrayList<>();
```

```
for(int i=0; i<hipHopNames.length; i++)
        hipHopSongs.add(new Track(hipHopNames[i], hiphopRatings[i]));
Album hipHopAlbum = new Album("Hip hop",hipHopSongs, 2017);


String[] jazzNames = {"Sham", "Masakali","Lovely"};
int[] jazzRatings = {3,1,2};
List<Track> jazzSongs = new  ArrayList<>();
for(int i=0; i<jazzNames.length; i++)
    jazzSongs.add(new Track(jazzNames[i], jazzRatings[i]));
Album jazzAlbum = new Album("Jazz",jazzSongs, 1995);
```

```
        List<String> sortedAlbums =  sortAlbumsByYear(Arrays.asList(travelAlbum, jazzAlbum,
hipHopAlbum, rapAlbum));
        System.out.println("Sorted list of albums by their year of release: \n"+sortedAlbums+"\n");


        List<String> goodAlbums =  filterGoodAlbums(Arrays.asList(travelAlbum, jazzAlbum,
hipHopAlbum, rapAlbum));
        System.out.println("Sorted list of Good albums(rating>4) by their year of release:
\n"+goodAlbums);


    }
}
```



Conclusion: In this experiment, we used various methods of Java Stream API and performed various programs.

Practical no. 6

FS19CO042

Aim: Implement programs related to File I/O

- 6.1 Create a csv file which will contain 10 integers in a spreadsheet. Read the file using class java.util.Scanner and display the sum of the numbers in the file. Handle all possible exceptions. Write a Java program to create, read and modify a file.
- 6.2 Create two objects of class Path viz., source and target. Perform the following operations a. Create a file at source b. Copy a file from source to target c. Move a file from source to target d. Delete a file from source e. Retrieve information about source and target

Tools used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Java file I/O

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

The two important streams are **FileInputStream** and **FileOutputStream**

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file −

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have *InputStream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file −

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Once you have *OutputStream* object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

Directories in Java

A directory is a File which can contain a list of other files and directories. You use **File** object to create directories, to list down files available in a directory. For complete detail, check a list of all the methods which you can call on File object and what are related to directories.

Creating Directories

There are two useful **File** utility methods, which can be used to create directories −

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Code:

- **Create a csv file which will contain 10 integers in a spreadsheet. Read the file using class java.util.Scanner and display the sum of the numbers in the file. Handle all possible exceptions. Write a Java program to create, read and modify a file.**

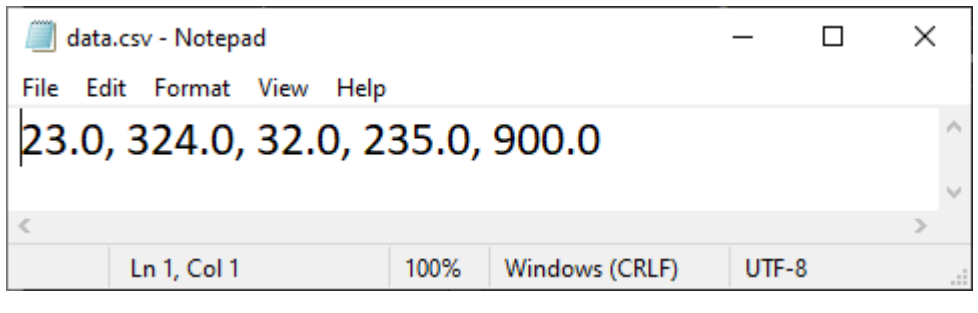
```
import java.util.*;
import java.io.*;


public class exp6_1 {
    public static void main(String[] args) {
        double sum = 0;
        List<Double> numbers = new ArrayList<>();


        String dataFilePath = "data.csv";
        String defaultData = "200,500,25,50\n100,50,25\n70,30,40,60\n40\n60,90,150\n20,40";
        Scanner inputScanner = new Scanner(System.in);
        char opted;

        File dataFile = new File(dataFilePath);


        // If file doesn't exist, creating one and writing default data to the file
        if (!dataFile.exists()) {
            System.out.println("CSV file could not be found, hence creating one !\nYou can put desired data in the file manually separated by commas");
            try {
                if (dataFile.createNewFile()) {
                    System.out.println("File created: " + dataFile.getAbsolutePath());
                    FileWriter defauFileWriter = new FileWriter(dataFile.getName());
                    defauFileWriter.write(defaultData);
```

```
            defauFileWriter.close();
        } else
            System.out.println("There was a problem creating new file, try creating one manually");

    } catch (Exception e) {
        System.out.println("An error occurred while writing default data to the file, try writing manually :)");
        e.printStackTrace();
    }
}

// Find sum of all numbers in csv file
// All numbers in CSV file:
displaySum(dataFile, sum, numbers);

// Modify the numbers in csv file

System.out.print("Do you want to write numbers to csv file? y/n: ");
opted = inputScanner.nextLine().trim().charAt(0);
if (opted == 'y' || opted == 'Y') {
    int n;
    double entity;
    System.out.println("How many decimal numbers do you wish to write to file(eg. 4 or 8): ");
    n = inputScanner.nextInt();
    System.out.println("Enter the numbers separated by spaces:");
    List<Double> newData = new ArrayList<>();
while (n-- > 0) {
    entity = inputScanner.nextDouble();
    newData.add(entity);
}
String data = newData.toString();
data = data.substring(1, data.length() - 1);
System.out.println(data);

try {
    FileWriter customWriter = new FileWriter(dataFile.getName());
    customWriter.write(data);
    customWriter.close();
```

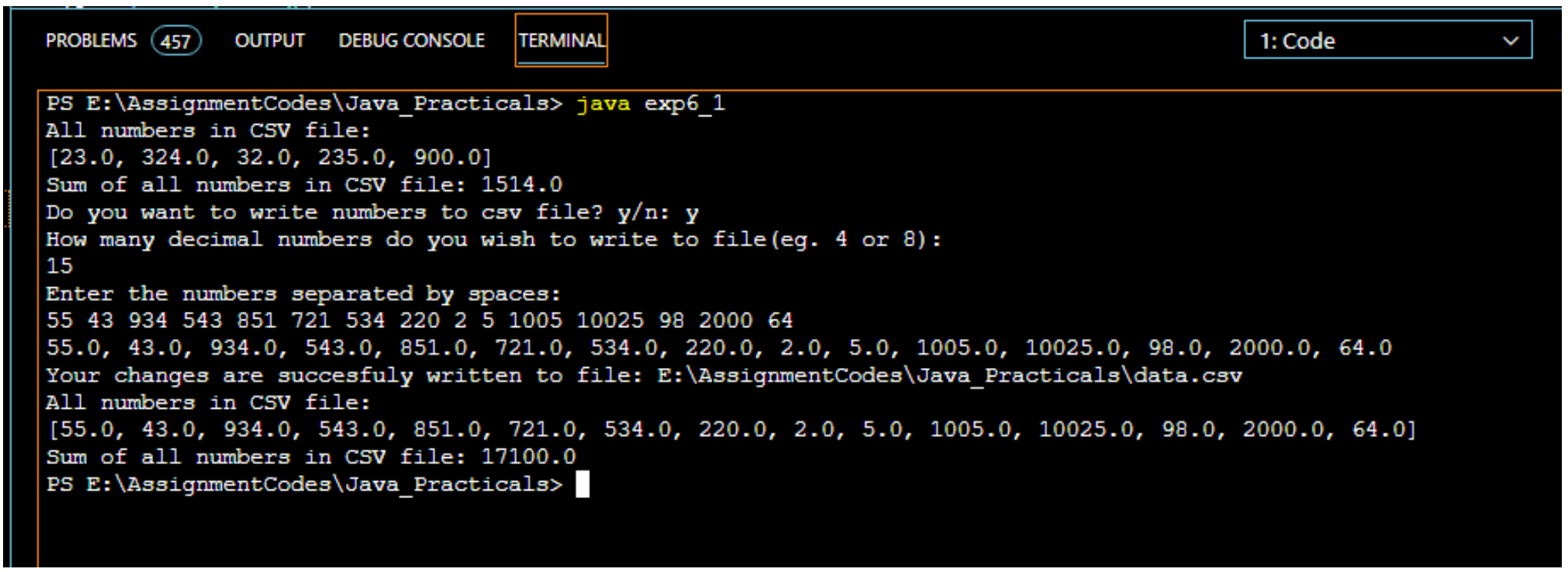
```
            System.out.println("Your changes are succesfuly written to file: " + dataFile.getAbsolutePath());
        } catch (Exception e) {
            System.out.println("xxxxxx ERROR xxxxxxxxx::: Close the CSV file And Try Again   :)");
            e.printStackTrace();
        }
    }
    numbers.clear();
    sum = 0;
    displaySum(dataFile, sum, numbers);                 // All numbers in CSV file:
    inputScanner.close();
}

private static void displaySum(File dataFile, double sum, List<Double> numbers) {
    try {
        Scanner csvScanner = new Scanner(dataFile);
        Scanner dataScanner = null;
        while (csvScanner.hasNextLine()) {
            dataScanner = new Scanner(csvScanner.nextLine());
            dataScanner.useDelimiter(",");

            while (dataScanner.hasNext()) {
                try {
                    String data = dataScanner.next().trim();
                    numbers.add(Double.parseDouble(data));
                    sum += Double.parseDouble(data);
                } catch (NumberFormatException ne) {
                    continue;
                }
            }
            dataScanner.close();
        }
        csvScanner.close();
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    }

    System.out.println("All numbers in CSV file:\n" + numbers.toString());
```

```
        System.out.println("Sum of all numbers in CSV file: " + sum);
    }
}
```

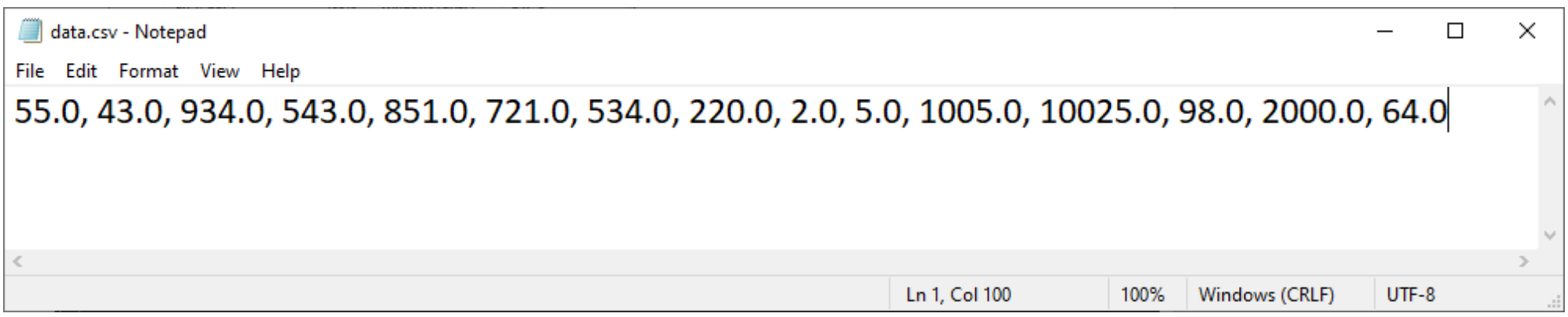
Previous CSV file:



Output:



CSV file after modifications:



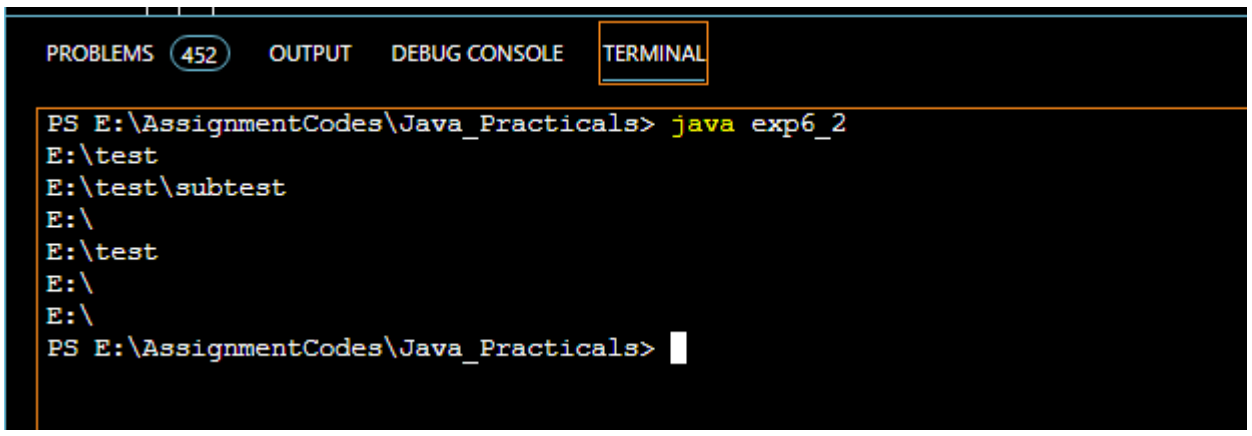
- **Create two objects of class Path viz., source and target. Perform the following operations a. Create a file at source b. Copy a file from source to target c. Move a file from source to target d. Delete a file from source e. Retrieve information about source and target**

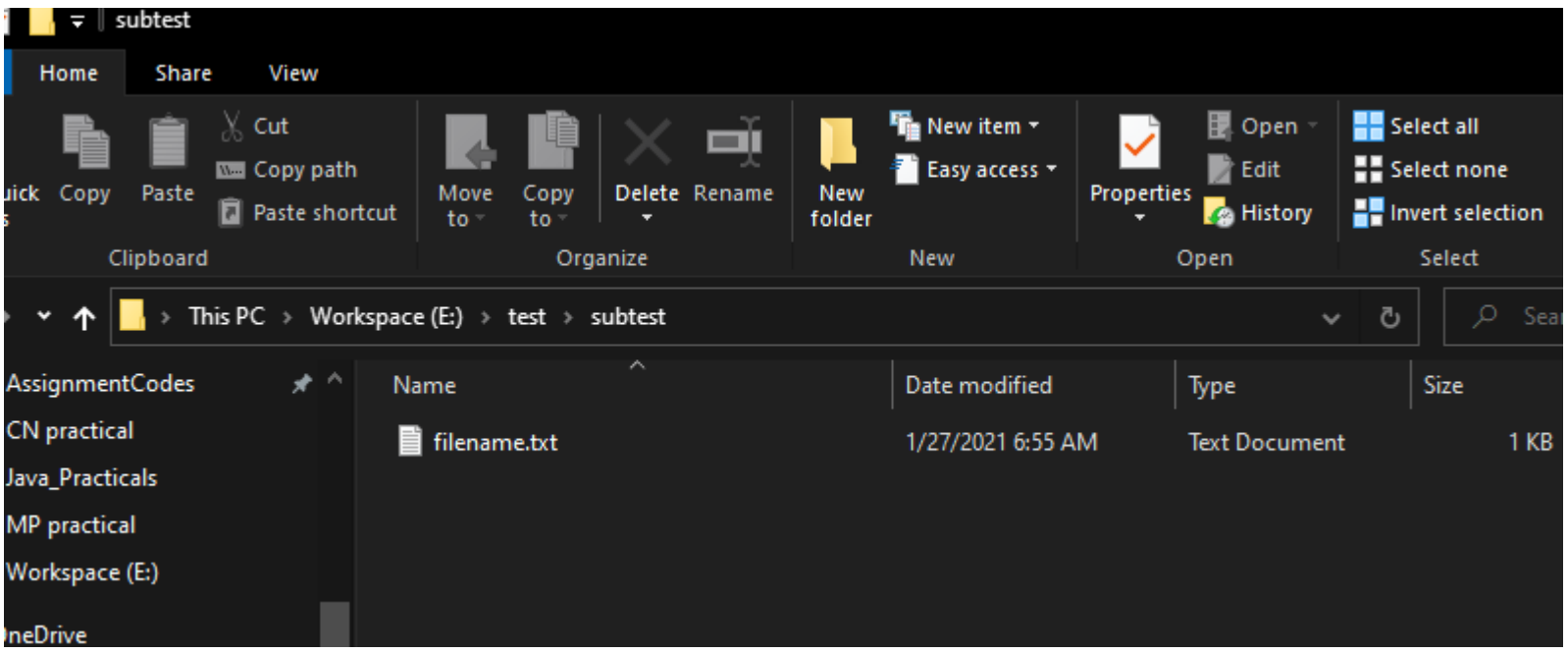
```
import java.nio.file.*;
import java.nio.file.Paths;
import java.io.*;
public class exp6_2 {

    public static void main(String[] args) {
```

```
try{
    Path source = Paths.get("E:\\test");
    Path target = Paths.get("E:\\test\\subtest");

    String fn1=source+"\\";
    String fn2=target+"\\";

    FileWriter myWriter = new FileWriter(fn1+"filename.txt");
    myWriter.write("Files in Java might be tricky, but it is fun enough!");
    myWriter.close();

    //copy
    InputStream is = null;
    OutputStream os = null;
    File s=new File(fn1+"filename.txt");
    File d=new File(fn2+"filename.txt");
    is = new FileInputStream(s);
    os = new FileOutputStream(d);
    byte[] buffer = new byte[1024];
    int length;
    while ((length = is.read(buffer)) > 0) {
       os.write(buffer, 0, length);
    }
    is.close();
    os.close();

    //move
    d.delete();
    Path temp = Files.move(Paths.get(fn1+"filename.txt"),  Paths.get(fn2+"filename.txt"));

    //delete
    s.delete();

    //retrieve
```

```
            System.out.println(source+"");
            System.out.println(target+"");
            System.out.println(source.getParent()+"");
            System.out.println(target.getParent()+"");
            System.out.println(source.getRoot()+"");
            System.out.println(target.getRoot()+"");
        }catch(Exception ex){
            System.out.println(ex+"");
        }
    }
}
```

Output:





Conclusion: In this experiment, we performed various File Read/write operations like editing csv files, copying/moving/deleting files, etc using FIleWriter, InputStream, OutputStream, etc.

Practical no. 7

FS19CO042

Aim: Generate complete Javadocs for any two of the above experiments

Tool used: Editor (Notepad/Intellij IDE), JDK and JRE

Theory:

Javadoc

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.

Following is a simple example where the lines inside /*….*/ are Java multi-line comments. Similarly, the line which preceeds // is Java single-line comment.

Example

```
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Omkar Phansopkar
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {
   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

You can include required HTML tags inside the description part. For instance, the following example makes use of <h1>....</h1> for heading and <p> has been used for creating paragraph break −

Example

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
*
* @author  Omkar Phansopkar
* @version 1.0
* @since   2014-03-31
```

```
*/
public class HelloWorld {


   public static void main(String[] args) {
      // Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

The javadoc Tags

The javadoc tool recognizes the following tags −

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.	{@code text}
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@inheritDoc}	Inherits a comment from the **nearest** inheritable class or implementable interface.	Inherits a comment from the immediate surperclass.
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
{@linkplain}	Identical to {@link}, except the link's label is displayed in plain text than code font.	{@linkplain package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to	@param parameter-name description

	the "Parameters" section.	
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField component.	@serialField field-name field-type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
{@value}	When {@value} is used in the doc comment of a static field, it displays the value of that constant.	{@value package.class#field}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

Command to generate html pages from javadocs:
javadoc -d .\pathToDestination .\pathToSrc.java file.

Code:

File 1, exp7a.java

```
import java.util.Scanner;
/**
 * in this class we are adding the square root of indivisual numbers;
 */
```

```
public class exp7a{
   /**
    * in this method we are taking the numbers as input and returning the addtion to main function;
    * @param x;
    * @return to main;
    */
   public static int add(int ...x)
   {
      return x[0]*x[0]+x[1]*x[1];
   }
   /**
    * this is the main method where the calling to add function is done and printing the result;
    * @param args
    */
  public static void main(String args[])
  {   //creating Scanner class object and passing system.in ;
      Scanner sc= new Scanner(System.in);
      System.out.println("enter the first number:");
      int n1=sc.nextInt();
      System.out.println("enter the second number:");
      int n2=sc.nextInt();
      int a=exp7a.add(n1,n2);
      System.out.print("the addtion of square root of indivisual is :"+a);
  }
}
```

File 2, exp7b.java

```
import java.util.Arrays;
/**
 * this is a assignment7 class for sorting the array;
 */
public class exp7b{
   /**
    * This is the main method where the arrays sorted and printed;
```

```
 *@param args
 */
public static void main(String args[])
{
    System.out.println("sorting the array.....");
    System.out.println("sorted array:");
    Arrays.sort(args);
    for(String i:args)
    {
      System.out.println(i);
    }
  }
}
```

Generating docs:
Perform following commands to generate doc:
javadoc -d .\pathToDestination .\pathToSrc.java file.

Actual commands:

javadoc -d .\javadocs\ .\exp7a.java

javadoc -d .\javadocs\ .\exp7b.java

Console output:

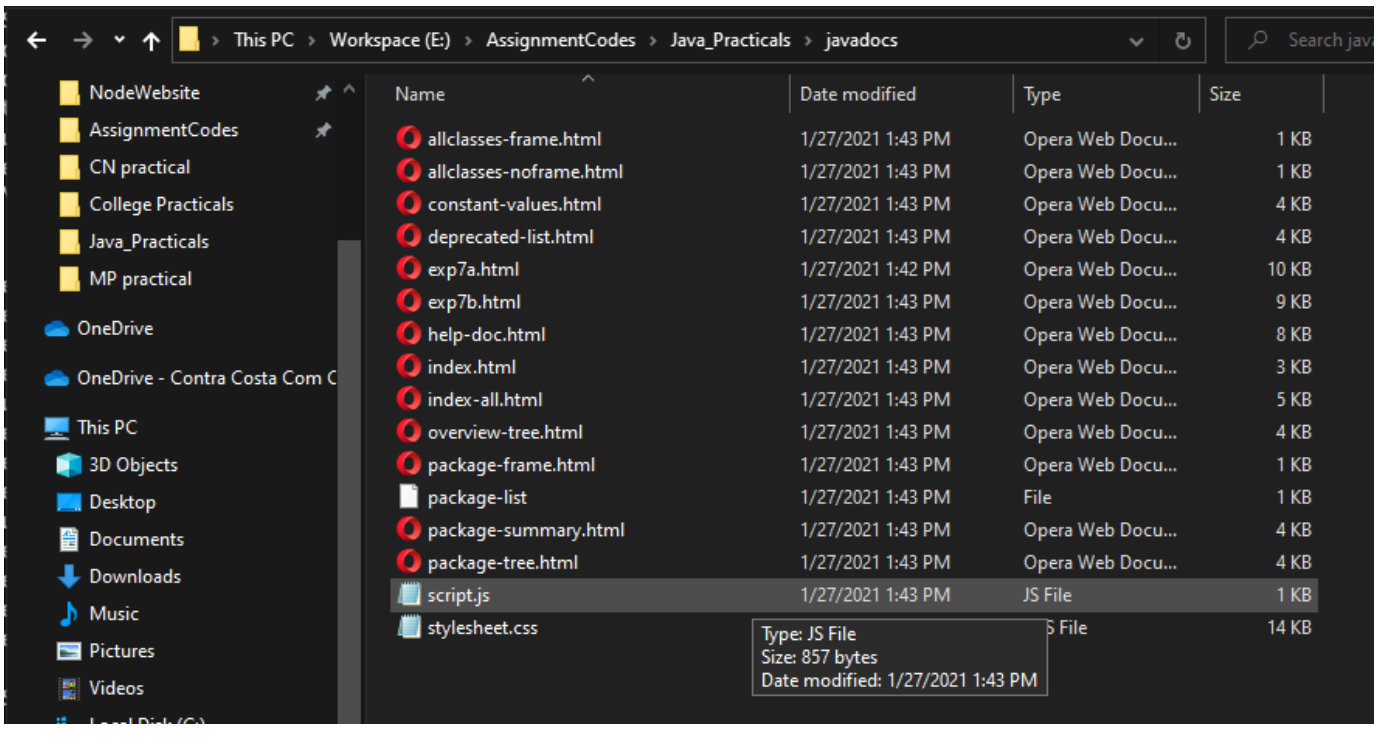
Windows PowerShell

```
PS E:\AssignmentCodes\Java_Practicals> javadoc -d .\javadocs\ .\exp7a.java
Loading source file .\exp7a.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_251
Building tree for all the packages and classes...
Generating .\javadocs\exp7a.html...
.\exp7a.java:17: warning: no description for @param
     * @param args
       ^
Generating .\javadocs\package-frame.html...
Generating .\javadocs\package-summary.html...
Generating .\javadocs\package-tree.html...
Generating .\javadocs\constant-values.html...
Building index for all the packages and classes...
Generating .\javadocs\overview-tree.html...
Generating .\javadocs\index-all.html...
Generating .\javadocs\deprecated-list.html...
Building index for all classes...
Generating .\javadocs\allclasses-frame.html...
Generating .\javadocs\allclasses-noframe.html...
Generating .\javadocs\index.html...
Generating .\javadocs\help-doc.html...
1 warning
PS E:\AssignmentCodes\Java_Practicals>
```

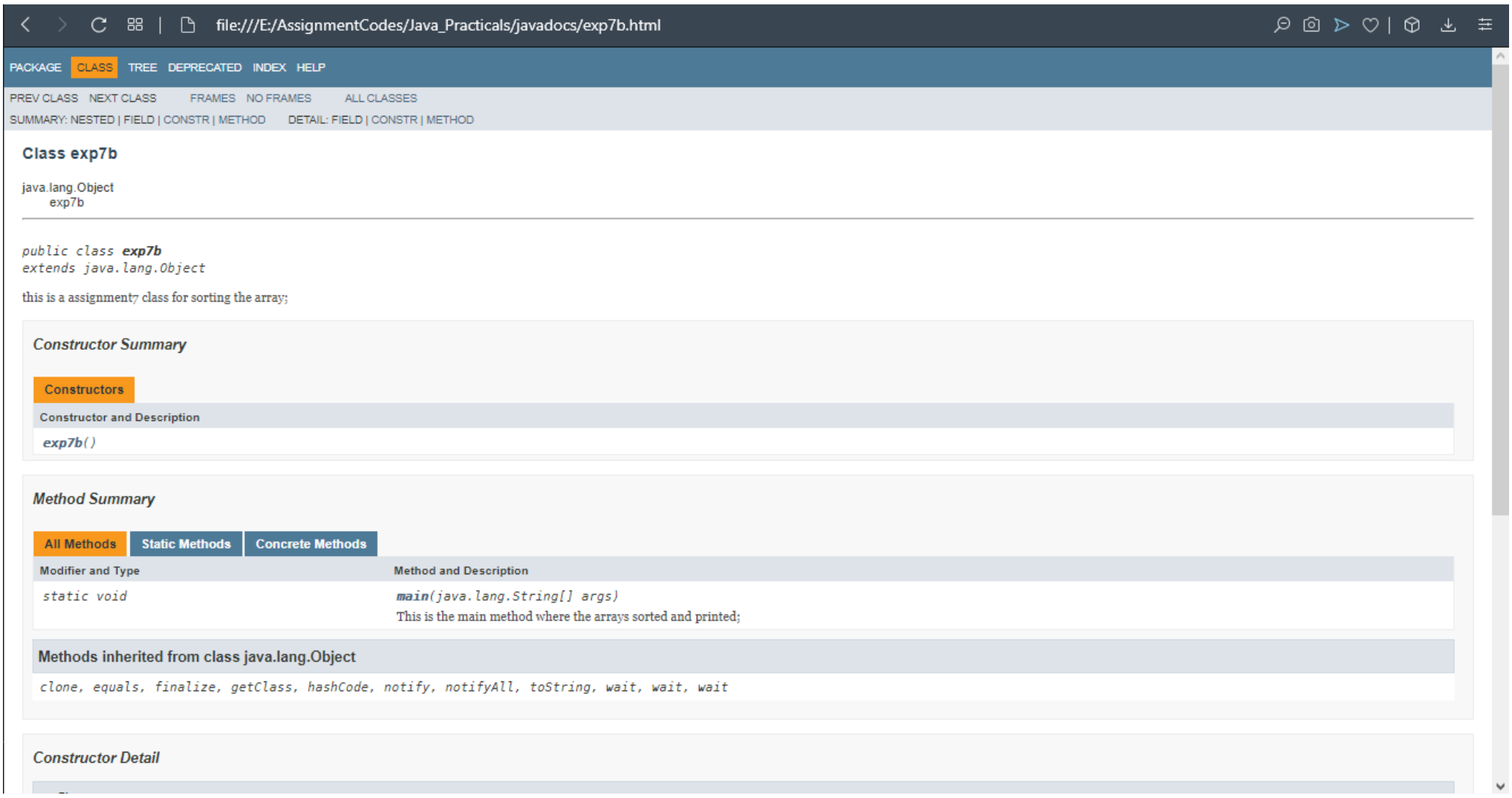
Windows PowerShell

```
PS E:\AssignmentCodes\Java_Practicals> javadoc -d .\javadocs\ .\exp7b.java
Loading source file .\exp7b.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_251
Building tree for all the packages and classes...
Generating .\javadocs\exp7b.html...
.\exp7b.java:8: warning: no description for @param
    *@param args
     ^
Generating .\javadocs\package-frame.html...
Generating .\javadocs\package-summary.html...
Generating .\javadocs\package-tree.html...
Generating .\javadocs\constant-values.html...
Building index for all the packages and classes...
Generating .\javadocs\overview-tree.html...
Generating .\javadocs\index-all.html...
Generating .\javadocs\deprecated-list.html...
Building index for all classes...
Generating .\javadocs\allclasses-frame.html...
Generating .\javadocs\allclasses-noframe.html...
Generating .\javadocs\index.html...
Generating .\javadocs\help-doc.html...
1 warning
PS E:\AssignmentCodes\Java_Practicals>
```

Docs Output:


This PC › Workspace (E:) › AssignmentCodes › Java_Practicals › javadocs
NodeWebsite
AssignmentCodes
CN practical
College Practicals
Java_Practicals
MP practical
OneDrive
OneDrive - Contra Costa Com C
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Name
Date modified
Type
Size
allclasses-frame.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
allclasses-noframe.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
constant-values.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
deprecated-list.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
exp7a.html 1/27/2021 1:42 PM Opera Web Docu... 10 KB
exp7b.html 1/27/2021 1:43 PM Opera Web Docu... 9 KB
help-doc.html 1/27/2021 1:43 PM Opera Web Docu... 8 KB
index.html 1/27/2021 1:43 PM Opera Web Docu... 3 KB
index-all.html 1/27/2021 1:43 PM Opera Web Docu... 5 KB
overview-tree.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
package-frame.html 1/27/2021 1:43 PM Opera Web Docu... 1 KB
package-list 1/27/2021 1:43 PM File 1 KB
package-summary.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
package-tree.html 1/27/2021 1:43 PM Opera Web Docu... 4 KB
script.js 1/27/2021 1:43 PM JS File 1 KB
stylesheet.css File 14 KB
Type: JS File
Size: 857 bytes
Date modified: 1/27/2021 1:43 PM

file:///E:/AssignmentCodes/Java_Practicals/javadocs/exp7a.html
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Class exp7a
java.lang.Object
exp7a
public class exp7a
extends java.lang.Object
in this class we are adding the square root of indivisual numbers;
Constructor Summary
Constructors
Constructor and Description
exp7a()
Method Summary
All Methods Static Methods Concrete Methods
Modifier and Type
Method and Description
static int
add(int... x)
in this method we are taking the numbers as input and returning the addtion to main function;
static void
main(java.lang.String[] args)
this is the main method where the calling to add function is done and printing the result;
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Conclusion: Thus we understood and successfully created javadocs for our project using various techniques used for commenting and documenting java experiments.