Lab 2: Programming Fundamentals - JAVA

CLO: 1

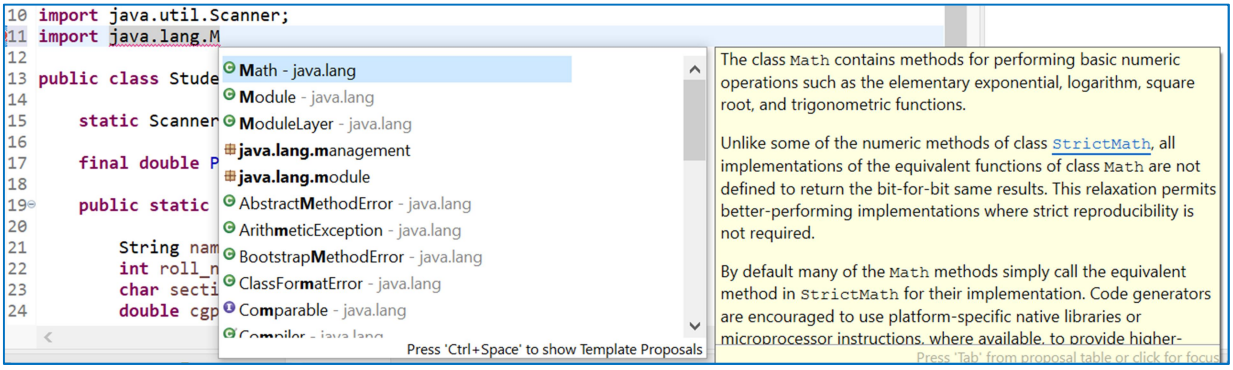
- The basic unit of a Java program is a class. class is used to create Java programs, it is used to group a set of related operations; and it is used to allow users to create their own data types. A Java class can have at most one method main. When you execute (run) a Java (application) program, execution always begins with the method main. This is the example of simplest java program.

Example # 1

```
public class Student {
    public static void main(String[] args) {
        System.out.println("I am learning JAVA Programming");
    }
}
```

- To make use of the existing classes, methods, and identifiers, you must tell the program which package contains the appropriate information. The import statement helps you do this.

Example # 2



Basic of Java Programming

➢ Comments

Adding comments in the code is a good programming practice. Give brief explanation what the program is intended to do. This practice makes developer friendly code for later enhancements.

Example # 3

```
// This is single line comment

/*
 * This is multiple line comment
 */
```

➢ Identifiers

Identifiers are names of things, such as variables, constants, and methods that appear in programs. A Java identifier consists of letters, digits, the underscore character (_), and the dollar sign ($) and must begin with a letter, underscore, or the dollar sign.

Example # 4

Variables A memory location whose content may change during program execution.	**Named constant** A memory location whose content is not allowed to change during program execution.

`int roll_no;`
`double amountDue;`
`char section;`
	`final int NO_OF_STUDENTS = 29;`

➢ Data Types

Data type is defined as a set of values together with a set of operations on those values.

Primitive Data type

There are three categories of primitive data types:

- **Integral**: this data type deals with integers, or numbers without a decimal part (and characters). It is further classified into five categories: char, byte, short, int, and long.
- **Floating-point**: this data type deals with decimal numbers. Decimal numbers categories: float and double.
- **Boolean**: this data type deals with logical values. In Java, boolean is reserved word for this data type.

Data Type	Values	Storage (in bytes)
`char`	0 to 65535 (= $2^{16} - 1$)	2 (16 bits)
`byte`	−128 (= -2^7) to 127 (= $2^7 - 1$)	1 (8 bits)
`short`	−32768 (= -2^{15}) to 32767 (= $2^{15} - 1$)	2 (16 bits)
`int`	−2147483648 (= -2^{31}) to 2147483647 (= $2^{31} - 1$)	4 (32 bits)
`long`	−9223372036854775808 (= -2^{63}) to 9223372036854775807 (= $2^{63} - 1$)	8 (64 bits)

➢ Casting or Type conversion

When a value of one data type is automatically treated as another data type, implicit type coercion has occurred. If you are not careful about data types, implicit type coercion can generate unexpected results.

To avoid implicit type coercion, Java provides for explicit type conversion through the use of a cast operator.

Form: (dataTypeName) expression

Example # 5

(int)(8.9) evaluate to 8, (double)(15 / 2) = (double)(7) (because 15 / 2 = 7)= 7.0

- **String Data type**

To process strings effectively, Java provides the class String. The class String contains various operations to manipulate a string. Technically, the class String is not a primitive type.

String	"Sunny Day"								
Character in the string	'S'	'u'	'n'	'n'	'y'	' '	'D'	'a'	'y'
Position of the character in the string	0	1	2	3	4	5	6	7	8

The length of the string "Sunny Day" is 9.

Example # 6

str1.compareTo(str2) =	an integer value less than 0 if string str1 is less than string str2 0 if string str1 is equal to string str2 an integer value greater than 0 if string str1 is greater than string str2	
str1.compareTo(str2)	< 0	str1 = "Hello" and str2 = "Hi". The first character of str1 and str2 are the same, but the second character 'e' of str1 is less than the second character 'i' of str2. Therefore, str1.compareTo(str2) < 0.

- **Operators**

Relational Operators

Java includes six relational operators that enable you to make comparisons. These operators evaluate to true or false.

==	**equal to**
!=	**not equal to**
<	**less than**
<=	**less than or equal to**
>	**greater than**
>=	**greater than or equal to**

Arithmetic operators

You can use the standard arithmetic operators to manipulate integral and floating-point data types. Java has five arithmetic operators: Arithmetic Operators: **+ (addition), - (subtraction or negation), * (multiplication), / (division), % (mod, (modulus or remainder))**

According to the order of precedence rules for arithmetic operators: ***, /, %** have a higher level of precedence than **+, -**

Note that the operators *, /, and % have the same level of precedence. Similarly, the operators + and - have the same level of precedence. When arithmetic operators have the same level of precedence, operations are performed from left to right.

Example # 7

```
System.out.println("5.0 + 3.5 = " + (5.0 + 3.5));
System.out.println("3.0 + 9.4 = " + (3.0 + 9.4));
```

Sample Run:

```
5.0 + 3.5 = 8.5
3.0 + 9.4 = 12.4
```

Logical Boolean Operators

Operator	Description
!	not
&&	and
||	or

➢ Input & Output

Java, output on the standard output device is accomplished by using the standard output object System.out. The following example highlights syntax for output and input.

Example # 8

```
System.out.printf(formatString);

System.out.printf("Hello there!");

System.out.printf(formatString, argumentList);

System.out.printf("There are %.2f inches in %d centimeters.%n",
                  centimeters / 2.54, centimeters);
```

's'	general	The result is a string
'c'	character	The result is a Unicode character
'd'	integral	The result is formatted as a (decimal) integer
'e'	floating point	The result is formatted as a decimal number in computerized scientific notation
'f'	floating point	The result is formatted as a decimal number
'%'	percent	The result is '%'
'n'	line separator	The result is the platform-specific line separator

To put data into variables from the standard input device, Java provides the class Scanner. Using this class, first we create an input stream object and associate it with the standard input device. In Example # 9

```
import java.util.*;

public class Student {

    static Scanner input = new Scanner (System.in);

    final double PASSING_GPA = 2.5;

    public static void main(String[] args) {

        String name;
        int roll_no;
        char section;
        double cgpa;

        System.out.println("Enter Student Name:");
        name = input.next();
        System.out.println("Enter Roll Number:");
        roll_no = input.nextInt();
        System.out.println("Enter Section:");
        section = input.next().charAt(0);
        System.out.println("Enter CGPA:");
        cgpa = input.nextDouble();

        System.out.println("STUDENT INFORMATION");
        System.out.println("Name: " + name);
        System.out.println("Roll# " + roll_no);
        System.out.println("Section: " + section);
        System.out.println("CGPA: " + cgpa);
    }
}
```

Command	Meaning
console.nextInt()	next input token interpreted as an integer,
console.nextDouble()	next input token interpreted as a floating-point number
console.next()	next input token interpreted as a string
console.nextLine()	next input token interpreted as a string until the end of the line
console.next().charAt(0)	next input token interpreted as a single printable character,

➢ File Reading/Writing CLO2

Step 1: Import necessary classes from the packages java.util and java.io.

Step 2: Create and associate appropriate class variables with the input/output sources.

Step 3: Read the data from the input file using the variables created and write the data in files

Step 4: Close the input and output files.

Example # 10

Suppose an input file, say employeeData.txt, consists of the following data:
Emily Johnson 45 13.50

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

   //Create and associate the Scanner object to the input source
Scanner inFile = new Scanner(new FileReader("employeeData.txt"));

String firstName;   //variable to store first name
String lastName;    //variable to store last name

double hoursWorked; //variable to store hours worked
double payRate;     //variable to store pay rate
double wages;       //variable to store wages

firstName = inFile.next();  //get the first name
lastName = inFile.next();   //get the last name

hoursWorked = inFile.nextDouble(); //get hours worked
payRate = inFile.nextDouble();     //get pay rate

wages = hoursWorked * payRate;
```

The following statement closes the input file to which `inFile` is associated:

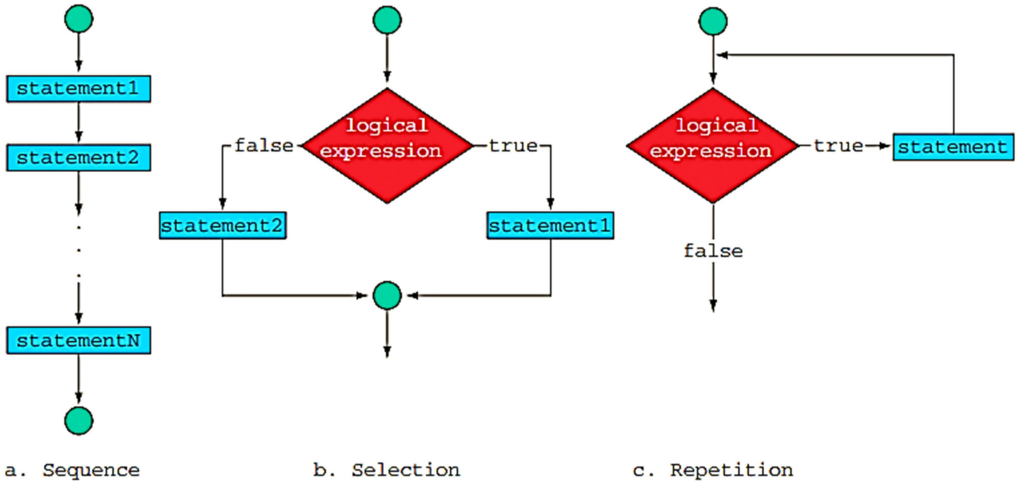
```
inFile.close();  //close the input file

PrintWriter outFile = new PrintWriter("prog.out");

outFile.println("The paycheck is: $"+ wages);

outFile.close();
```

➢ Control Structure



a. Sequence b. Selection c. Repetition

Selection

- **One way selection**

Example # 11

```
if (cgpa < 2.00) {
    grade = 'F';
}
```

- **Two way selection**

Example # 12

```
if (section == 'A') {
    System.out.println("This section is learning JAVA");
}else {
    System.out.println("This section is learning C++");
}
```

- **Compound Selection**

Example # 13

```
if (temperature >= 50)
    if (temperature >= 80)
    System.out.println("Good day for swimming.");
    else
    System.out.println("Good day for golfing.");
else
    System.out.println("Good day to play tennis.");
```

- Switch Structure

Example # 14

```
switch (grade)
{
case 'A':
   System.out.println("The grade is A.");
   break;

case 'B':
   System.out.println("The grade is B.");
   break;

case 'C':
   System.out.println("The grade is C.");
   break;

case 'D':
   System.out.println("The grade is D.");
   break;

case 'F':
   System.out.println("The grade is F.");
   break;

default:
   System.out.println("The grade is invalid.");
}
```

Repetition

1. While Loop

- Counter Controlled while loop

Example # 15

```
int i = 20;

while (i < 20)
{
    System.out.print(i + " ");
    i = i + 5;
}

System.out.println();
```

- Sentinel Controlled while loop

```
while (variable != sentinel)
{
    .
    .
    .
    input a data item into variable

}
```

Example # 16

```
static final int SENTINEL = -999;
while (number != SENTINEL)
{
    sum = sum + number;
    count++;
}
```

- Flag controlled while loop

```
found = false;

while (!found)
{
    .
    .
    .

    if (logical expression)
        found = true;
    .
    .
    .
}
```

Example # 17

```
boolean done;  //boolean variable to control the loop
done = false;

while (!done)
{
    System.out.print ("Enter an integer greater"
                    + " than or equal to 0 and "
                    + "less than 100: ");
    guess = console.nextInt();
    System.out.println();

    if (guess == num)
    {
        System.out.println("You guessed the "
                         + "correct number.");
        done = true;
    }
}
```

2. Do-while Loop

```
do
    statement
while (logical expression);
```

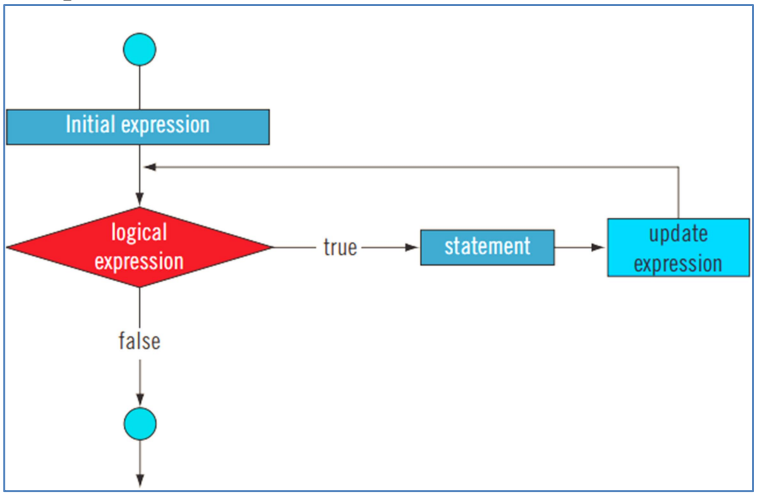
Example # 18

```
i = 0;
do
{
    System.out.print(i + " ");
    i = i + 5;
}
while (i <= 20);
```

The output of this code is:

```
0 5 10 15 20
```

3. For Loop




```
for (initial expression; logical expression; update expression)
    statement
```

Example # 19

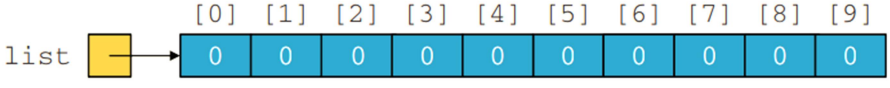
```
for (i = 0; i < 10; i++)
    System.out.print(i + " " );
```

➢ ARRAYS

```
dataType[] arrayName = new dataType[intExp];
```

Example # 20

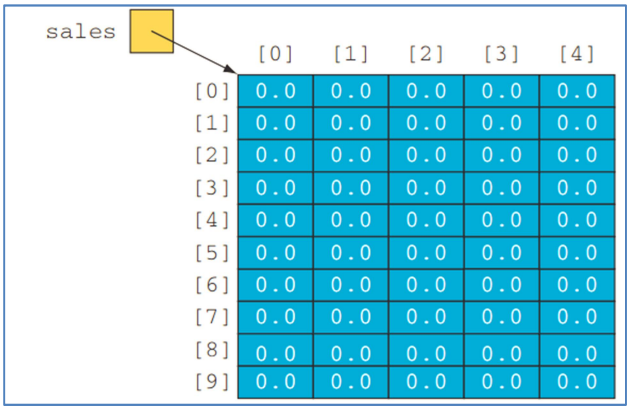
```
int[] list = new int[10];
```



A syntax for declaring a two-dimensional array is:

```
dataType[][] arrayName = new dataType[intExp1][intExp2];
```

Example # 21



➢ Array List

Example # 22

```
import java.util.ArrayList;

public class PeacticeLab1 {

    public static void main(String[] args) {
        // Create an ArrayList object
        ArrayList<String> cars = new ArrayList<String>();

        //Add Items
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");
        System.out.println(cars);

        // Insert element at the beginning of the list (0)
        cars.add(0, "Mazda");
        System.out.println(cars);

        //Access an Item
        System.out.println(cars.get(0));

        //Change an Item
        cars.set(0, "Opel");
        System.out.println(cars);

        //Remove an Item
        cars.remove(0);
        System.out.println(cars);

        //To remove all the elements in the ArrayList
        cars.clear();
        System.out.println(cars);

        //ArrayList Size
        cars.size();
        System.out.println(cars);

        // Loop Through an ArrayList
        for (int i = 0; i < cars.size(); i++) {
            System.out.println(cars.get(i));
        }
    }
}
```

➢ Linked List

Example # 23

```
import java.util.LinkedList;

public class PeacticeLab1 {

    public static void main(String[] args) {
        // Create an LinkedList object
        LinkedList<String> cars = new LinkedList<String>();
        cars.add("Volvo");
        cars.add("BMW");
        cars.add("Ford");
        cars.add("Mazda");

     // Use addFirst() to add the item to the beginning
        cars.addFirst("Mazda");
        System.out.println(cars);

     // Use addLast() to add the item to the end
        cars.addLast("BMW");
        System.out.println(cars);

     // Use removeFirst() remove the first item from the list
        cars.removeFirst();
        System.out.println(cars);

     // Use removeLast() remove the last item from the list
        cars.removeLast();
        System.out.println(cars);

    // Use getFirst() to display the first item in the list
        System.out.println(cars.getFirst());

     // Use getLast() to display the last item in the list
        System.out.println(cars.getLast());

    }
}
```

➢ HashMap

Example # 24

```
import java.util.HashMap;;
public class PeacticeLab1 {

    public static void main(String[] args) {
        HashMap<String, Integer> phoneBook = new HashMap<>();

        // Add key-value pairs (Name, Phone number)
        phoneBook.put("Alice", 123456789);
        phoneBook.put("Bob", 987654321);
        phoneBook.put("Charlie", 555666777);

        // Print the HashMap
        System.out.println("Phone Book: " + phoneBook);

        // Access a value using its key
        System.out.println("Bob's number: " + phoneBook.get("Bob"));

        // Check if a key exists
        if (phoneBook.containsKey("Alice")) {
            System.out.println("Alice's number is in the phone book.");
        }

        // Check if a value exists
        if (phoneBook.containsValue(555666777)) {
            System.out.println("Phone number 555666777 is in the phone book.");
        }

       // Check if a value exists
       if (phoneBook.containsValue(555666777)) {
           System.out.println("Phone number 555666777 is in the phone book.");
       }

       // Remove a key-value pair
       phoneBook.remove("Charlie");
       System.out.println("After removing Charlie: " + phoneBook);

       // Get the size of the HashMap
       System.out.println("Size of Phone Book: " + phoneBook.size());

       // Loop through the HashMap
       for (String key : phoneBook.keySet()) {
           System.out.println("Name: " + key + ", Phone: " + phoneBook.get(key));
       }

       // Clear the HashMap
       phoneBook.clear();
       System.out.println("Is the Phone Book empty? " + phoneBook.isEmpty());

    }
}
```

➢ HashSet

Example # 25

```
import java.util.HashSet;
public class PeacticeLab1 {

    public static void main(String[] args) {
            HashSet<String> fruits = new HashSet<>();

            // Add elements to the HashSet
            fruits.add("Apple");
            fruits.add("Banana");
            fruits.add("Orange");
            fruits.add("Mango");

            // Duplicate elements won't be added
            fruits.add("Apple");

            // Print the HashSet
            System.out.println("Fruits: " + fruits);

            // Check if a specific element exists
            if (fruits.contains("Banana")) {
                System.out.println("Banana is in the HashSet.");
            } else {
                System.out.println("Banana is not in the HashSet.");
            }

            // Remove an element
            fruits.remove("Mango");
            System.out.println("After removing Mango: " + fruits);

            // Get the size of the HashSet
            System.out.println("Size of HashSet: " + fruits.size());

            // Clear the HashSet
            fruits.clear();
            System.out.println("Is the HashSet empty? " + fruits.isEmpty());
    }
}
```

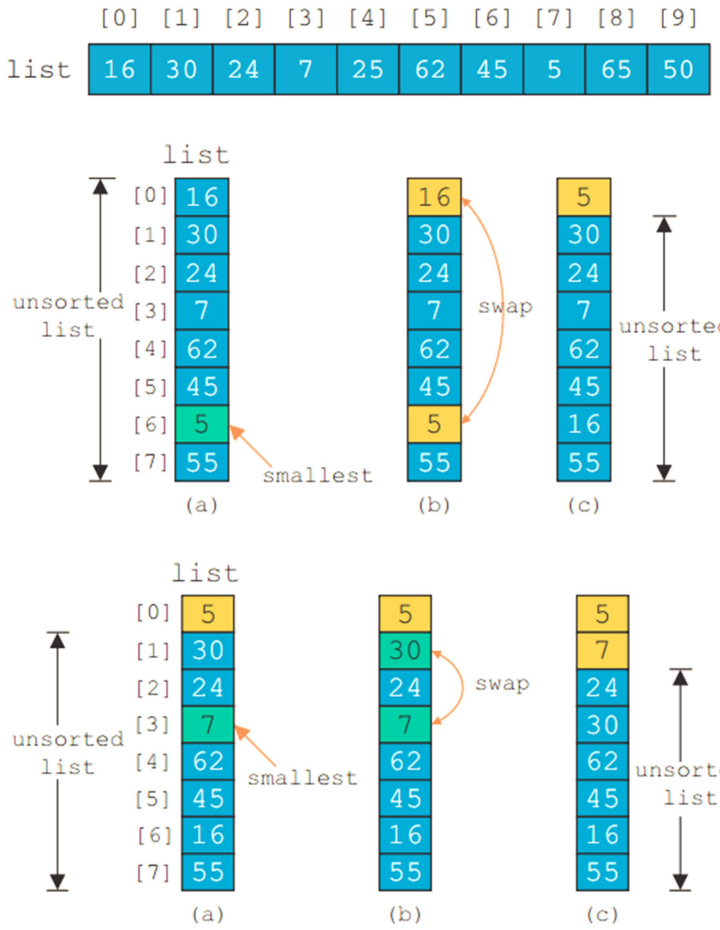
TASKS

CLO1 & CLO2

Task 1

Write Java program for Selection Sort

Selection Sort



In the unsorted portion of the list:

a. Find the location of the smallest element.
b. Move the smallest element to the beginning of the unsorted list.

```
for (index = 0; index < listLength - 1; index++)
{
   a. find the location, smallestIndex, of the smallest element in
      list[index]...list[listLength].
   b. Swap the smallest element with list[index]. That is, swap
      list[smallestIndex] with list[index].
}
```

Task 2

Write a program that reads a set of integers, from a file and then finds and prints the sum of the even and odd integers to a file.

Task 3

Consider the following sequence of numbers:

```
1, 1, 2, 3, 5, 8, 13, 21, 34, ....
```

Given the first two numbers of the sequence (say, a_1 and a_2), the nth number a_n, $n >= 3$, of this sequence is given by:

$a_n = a_{n-1} + a_{n-2}$

Thus:

$a_3 = a_2 + a_1 = 1 + 1 = 2,$

$a_4 = a_3 + a_2 = 2 + 1 = 3,$

and so on.

Such a sequence is called a **Fibonacci sequence**. In the preceding sequence, $a_2 = 1$ and $a_1 = 1$. However, given any first two numbers, using this process, you can determine the nth number, a_n, $n >= 3$, of the sequence. The number determined this way is called the nth **Fibonacci number**. Suppose $a_2 = 6$ and $a_1 = 3$.

Then:

$a_3 = a_2 + a_1 = 6 + 3 = 9;\ a_4 = a_3 + a_2 = 9 + 6 = 15.$

Next, we write a program that determines the nth Fibonacci number given the first two numbers.

Input: The first two numbers of the Fibonacci sequence and the position of the desired Fibonacci number in the Fibonacci sequence

Output: The nth Fibonacci number

Task 4

To make a profit, a local store marks up the prices of its items by a certain percentage. Write a Java program that reads the original price of the item sold the percentage of the marked-up price, and the sales tax rate. The program then outputs the original price of the item; the marked-up percentage of the item, the store's selling price of the item, the sales tax rate, the sales tax, and the final price of the item. (The final price of the item is the selling price plus the sales tax.)

Task 5

You found an exciting summer job for five weeks. It pays $15.50 per hour. Suppose that the total tax you pay on your summer job income is 14%. After paying the taxes, you spend 10% of your net income to buy new clothes and other accessories for the next school year and 1% to buy school supplies. After buying clothes and school supplies, you use 25% of the remaining money to buy savings bonds. For each

dollar you spend to buy savings bonds, your parents spend $0.50 to buy additional savings bonds for you. Write a program that prompts the user to enter the pay rate for an hour and the number of hours you worked each week. The program then outputs the following:

- Your income before and after taxes from your summer job
- The money you spend on clothes and other accessories
- The money you spend on school supplies
- The money you spend to buy savings bonds
- The money your parents spend to buy additional savings bonds for you

Task 6

Find the Minimum and Maximum Elements in Array, Array List, Linked List, Hash Map, and Hash Set