Assignment - 1

Name :- Tanishq Chauhan

Roll No. :- 2103184

Subject :- Object Oriented Programming (CER3C2)

Semester :- 3rd Semester

Branch :- Computer Science & Engineering (CS-B)

Chauhan

Tanishq Chauhan
2103184
CS-B

Chauhan
Date
Page 01

Object Oriented Programming (CER3C2)

Assignment - 1

1. Write a Java program to display the following pattern:

Answer of Q. No-1

```
class Pattern
{
   public static void main (String args[])
   {
      System.out.println ("   J   a   v   v   a ");
      System.out.println (   J a a   v v   a a ");
      System.out.println ( J J aaa   VV   aaa ");
      System.out.println ( JJ  a a   V   a a ");
   }
}
```

Output

```
   J   a   v   v   a
   J a a   v v   a a
J  J aaa   VV   aaa
JJ  a a   V   a a
```

Tanishq Chauhan
21C3184
CS-B

@Chauhan

02

Q.2. Write a class Car with instance members like Registration No, Model, Owner. Use this class as super class and create subclass by adding price as new member. Use constructors in these classes. Complete this program by writing main method and print values of different objects.

Answer of Q. No. 2

```
import java.util.Scanner;

class car

{

    private int Registration_No;
    private String Model;
    private String Owner;

    public car()

    {
        System.out.print("Enter the Registration No: ");
        Scanner sc = new Scanner(System.in);
        Registration_No = sc.nextInt();
        System.out.print("Enter Model No: ");
        Model = sc.next();
        System.out.print("Enter Owner's Name: ");
        Owner = sc.next();
    }
```

Tanishq Chauhan
21C3184
CS-B


```
    public void show()
    {
        System.out.println ("Registration No : " + Registration_No);
        System.out.println ("Model : " + Model);
        System.out.println ("Owner : " + Owner);
    }
}

class CarWithPrice extends Car
{
    private float price;

    public CarWithPrice ()
    {
        super();
        Scanner sc = new Scanner (System.in);
        System.out.print ("Enter the Price of Car : ");
        price = sc.nextFloat();
    }

    public void show()
    {
        super.show();
        System.out.println ("Price : " + price);
    }
}
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 04

```
class testCheck
{
    public static void main (String [] args)
    {
        CarWithPrice carA = new CarWithPrice();
        carA.show();

        CarWithPrice carB = new CarWithPrice();
        carB.show();
    }
}
```

Output

```
Enter the Registration No : 1234
Enter Model No : A
Enter Owner's Name : Tanishq
Enter the Price of Car : 500000.0

Registration No : 1234
Model : A
Owner : Tanishq
Price : 500000.0

Enter the Registration No : 9876
Enter Model No : B
Enter Owner's Name : Yash
Enter the Price of Car : 400000.0

Registration No : 9876
Model : B
Owner : Yash
Price : 400000.0
```

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 05

Q.3. Write a Java program to find the common elements between two arrays.

Answer of Q. No. 3

```
import java.util.*;

public class CommonElement
{
    public static void FindCommonElements (int[] array1,
                                            int[] array2 )

    {
        Set<Integer> set1  = new HashSet<>();
        Set<Integer> set2  = new HashSet<>();

        for (int i : array1)
        {
            set1.add (i);
        }

        for (int i : array2)
        {
            set2.add(i);
        }

        set1.retainAll (set2);
        System.out.println ("Common Elements are: " +
                                                set1);
    }
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Date
Page 06

```
public static void main (String[] args)
{
    int[] array1 = {25, 55, 89, 12, 67, 23, 1};

    int[] array2 = {67, 55, 1, 45, 97, 81, 25, 60};

    System.out.println ("Array1: " + Arrays.toString
                                    (array1));

    System.out.println ("Array 2: " + Arrays.toString
                                    (array2));

    FindCommonElements (array1, array2);

    }
}
```

Output

```
Array1: [25, 55, 89, 12, 67, 23, 1]
Array2: [67, 55, 1, 45, 97, 81, 25, 60]
Common Elements are: [1, 67, 55, 25]
```

Tanishq Chauhan
21C3184
CS-B

Q.4. Write a Java program to count and print all the duplicate characters in the input.

Answer of Q. No. 4

```
public class DuplicateCharacters
{
    public static void main (String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the String : ");
        String str1 = sc.nextLine();
        int count, cnt=0;
        char str[] = str1.toCharArray();
        System.out.println("Duplicate Characters in a
                                given String : ");

        for(int i=0 ; i<str.length ; i++)
        {
            count = 1;
            for(int j=i+1; j<str.length ; j++)
            {
                if(str[i] == str[j] && str[i] != ' ')
                {
                    count++;
                    str[j] = '0';
                }
            }
```

Tanishq Chauhan
21C3184
CS-B


```
            if (count >1 && str[i] != '0')
            {
                System.out.println (str[i]);
                cnt++;
            }
        }
        System.out.print("Count is : "+cnt);
    }
}
```

Output

```
Enter the String : Tanishq Chauhan
Duplicate Characters in a given String :
a
n
h
Count is : 3
```

Tanishq Chauhan
21C3184
CS-B

tchauhan
Date
Page 09

Q.5. Write a program that accepts three numbers from the user and prints "increasing" if the numbers are in increasing order, "decreasing" if the numbers are in decreasing order, and "Neither increasing or decreasing order" otherwise.

Answer of Q. No. 5.

```
import java.util.Scanner;

public class Order
{
    public static void main (String [] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print ("Enter first number: ");
        double x = in.nextDouble();
        System.out.print ("Enter second number: ");
        double y = in.nextDouble();
        System.out.print ("Enter third number: ");
        double z = in.nextDouble();

        if (x < y && y < z)
        {
            System.out.println ("Increasing");
        }
```

Tanishq Chauhan
21C3184
CS-B


```
else if (x>y && y>z)
{
  System.out.println ("Decreasing");
}

else
{
  System.out.println ("Neither increasing
                       or decreasing order");
}
}
}
```

Output

```
Enter first number : 121
Enter second number : 234
Enter third number : 345

Increasing
```

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 11

Q.6. What is Java Virtual Machine (JVM) and its role in Java Programming environment? Explain.

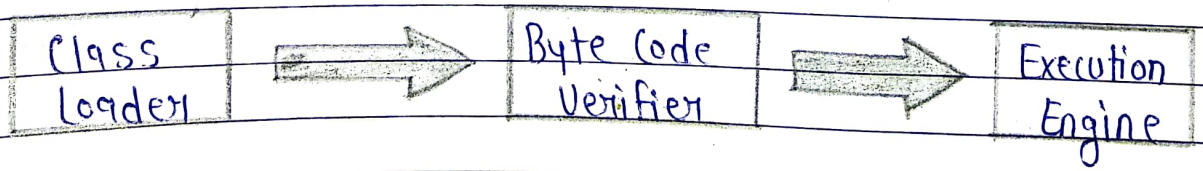
Answer of Q. No. 6

Java Virtual Machine

- Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java code or applications. It converts Java bytecode into machine language. JVM is a part of Java Runtime Environment. (JRE).
- In other programming languages, the compiler produce machine code for a particular system. However, Java compiler produce code for a Virtual Machine known as Java Virtual Machine.

How JVM Works

- First, Java code is compiled into bytecode. This bytecode gets interpreted on different machines.
- Between host system and Java source, Bytecode is an intermediary language.
- JVM in Java is responsible for allocating memory space.



Working of Java Virtual Machine (JVM)

Tanishq Chauhan
21C3184
CS-B

* The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM Role in Java Programming

It allows Java programs to run on any device or operating system (known as the "Write Once, Run Anywhere" principle) and to manage and optimize program memory.

The most common interaction with a running JVM is to check the memory usage in the heap and stack. The most common adjustment is tuning the JVM's memory settings.

It also support in Garbage Collection. Before Java, all program memory was managed by the programmer. In ~~memory through a process~~ Java, program memory is managed by the JVM. The JVM manages memory through a process called garbage collection, which continuously identifies and eliminates unused memory in Java programs. Garbage collection inside a running JVM.

Tanishq Chauhan
21C3184
CS-B

Q7. With the help of a complete Java program explain data type conversion rules in Java. Use appropriate comments in program to show the effect of type conversion.

Answer of Q. No: 7

- Data type conversion which is also known as the type casting.
- In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically.
- The automatic conversion is done by the compiler and manual conversion performed by the programmer.

There are two types of type casting:

(i) Widening Type Casting

- Converting a lower data type into higher one is called widening type casting. It is also known as implicit conversion or casting down. It is done automatically.
- It is safe because there is no chance to loss data.
- It takes place when:

(i) Both data type must be compatible with each other.
(ii) The target type must be larger than the source type.

* byte → short → char → int → long → float → double

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 14

Example of Widening Type Casting

```
public class WideningTypeCasting
{
   public static void main (String[] args)
   {
      int x = 7;
      // automatically converts the integer type into long type

      long y = x;
      // automatically converts the long type into float type

      float z = y;

      System.out.println("Before conversion, int value:" +x);
      System.out.println("After conversion, long value:" +y);
      System.out.println("After conversion, float value:" +z);
   }
}
```

Output

```
Before conversion, int value: 7
After conversion, long value: 7
After conversion, float value: 7.0
```

Tanishq Chauhan
21C3184
CS-B

(ii) Narrowing Type Casting

- Converting a higher data type into a lower one is called narrowing type casting.
- It is also known as explicit conversion or casting up.
- It is done manually by the programmer.
- If we do not perform casting then the compiler reports a compile-time-error.

double → float → long → int → char → short → byte

Example of Narrowing Type Casting

```
public class NarrowingTypeCasting
{
    public static void main (String args[])
    {
        double d = 166.66;
        // converting double data type into long data type

        long l = (long)d;
        // converting long data type into int data type

        int i = (int)l;

        System.out.println ("Before conversion : "+d);
        // fractional part lost

        System.out.println ("After conversion into long type: "+l);
```

Tanisha Chauhan
2103184
CS-B


```
        //fractional part lost
        System.out.println("After conversion into int type:"
                                                    +i);
    }
}
```

Output

```
Before conversion : 166.66
After conversion into long type : 166
After conversion into int type : 166
```

Tanishq Chauhan
2103184
CS-B

Chauhan

Date
Page 17

Q.8. Explain the concept of default and parameterized constructor with suitable java program.

Default Constructor

- A constructor that has no parameter is known as the default constructor.
- If we don't define a constructor in a class, then the compiler creates default constructor (with no arguments) for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.
- Default constructor provides the default values to the object like 0, null, etc. depending on the type.

Example

```
class Student
{
    int id;
    String name;

    //method to display the value of id and name

    void display()

    {

    System.out.println(id+"  "+name);
    }
```

Tanishq Chauhan
2103184
CS-B


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

    Student s1 = new Student();
    Student s2 = new Student();

    //displaying values of the object

    s1.display();
    s2.display();
    }
}
```

Output

```
0 null
0 null
```

Parameterized Constructor

- A constructor that has parameters is known as parameterized constructor.
- If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Tanishq Chauhan
21C3184
CS-B

Example:

```
class Student
{
    int id;
    String name;

    // creating a parameterized constructor

    Student (int i, String n)
    {
        id = i;
        name = n;
    }

    // method to display the values

    void display()
    {
        System.out.println(id + "  " + name);
    }

    public static void main (String args[])
    {
        // creating objects and passing values
        Student S1 = new Student (111, "Karan");
        Student S2 = new Student (222, "Aryan");
```

Tanisha Chauhan
21C3184
CS-B

Chauhan
Date
Page 20

```
//calling method to display the values of object

        s1.display();
        s2.display();
    }
}
```

Output

```
111  Karan
222  Aryan
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Page 21

Q.9. What is Inner class ? Explain different types of Inner classes ?

Answer of Q. No. 9 :

In Java, inner class refers to the class that is declared inside class or interface which were mainly introduced, to sum up, same logically relatable classes as Java is purely object-oriented so bringing it closer to the real world.

Advantage of Java Inner Class

- Making code clean and readable
- Private methods of the outer class can be accessed, so bringing a new dimension and making it closer to the real world.
- Optimizing the code module

Types of Inner Classes

There are basically four types of inner classes in Java.

1. Nested Inner Class
2. Method Local Inner Class
3. Static Nested Classes
4. Anonymous Inner Classes

Tanishq Chauhan
21C3184
CS-B

1. Nested Inner Class

It can access any private instance variable of the outer class. Like any other instance variable, we can have access modifier private, protected, public, and default modifier. Like class, an interface can also be nested and can have access specifiers.

Example

```
//Class1
// Helper Classes

class Outer
{
    //class 2  // Simple nested inner class
    class Inner
    {
        //show() method of inner class
        public void show()
        {
            System.out.println ("In a nested class
                                 method");
        }
    }
}

//Class 2
// Main class
```

Tanishq Chauhan
21C3184
CS-B


```
class Main
{
    public static void main (String [] args)
    {
        Outer . Inner in = new Outer(). new Inner();
        // calling show() method
        in . show();
    }
}
```

Output

In a nested class method

2. Method Local Inner Classes

Inner class can be declared within a method of an outer class ~~which we~~

Example :

```
// Class 1   // Outer Class
```

Tanishq Chauhan
21C3184
CS-B

Chauhan

Page 24

```
class Outer
{
    // Method inside outer class
    void outerMethod()
    {
        System.out.println("Inside outerMethod");

        // class 2  // Inner class  // It is local to outerMethod()

        class Inner
        {
            // Method inside inner class
            void innerMethod()
            {
                System.out.println("Inside innerMethod");
            }
        }
        // Creating object of inner class

        Inner y = new Inner();

        // Calling method defined inside it

        y.innerMethod();
    }
}

// Class 3  // Main Class
```

Tanishq Chauhan
21C3184
CS-B


```
class Main
{
    public static void main (String [] args)
    {
        // Creating object of outer class inside main()
        Outer x = new Outer();
        // calling method
        x.outerMethod();
    }
}
```

Output

```
inside outerMethod
inside innerMethod
```

3. Static Nested Classes

A static nested classes are not technically inner classes. They are like a static member of outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.

Tanishq Chauhan
21C3184
CS-B

Example

```
import java.util.*;

// Class1  // Outer Class

class Outer
{
    // Method
    private static void outerMethod()
    {
        System.out.println("inside outerMethod");
    }

    // Class 2  // Static inner class
    static class Inner
    {
        public static void display()
        {
            System.out.println("inside inner class
                                Method");

            outerMethod();
        }
    }
}

// Class 3
// Main Class
```

Tanishq Chauhan
21C3184
CS-B


```
class Main
{
    public static void main (String args[])
    {
        Outer.Inner obj = new Outer.Inner();
        obj.display();
    }
}
```

Output

```
inside inner class Method
inside outerMethod
```

4. Anonymous Inner Classes

Anonymous inner classes are declared without any name at all. They are created in two ways.

Way 1: As a subclass of the specified type

Example:

```
import java.util.*;

//class1
// Helper class
```

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 28

```
class Demo
{
    // Method of helper class
    void show()
    {
        System.out.println("I am in show method of
                                    super class");
    }
}

// Class 2

class FlavorIDemo
{
    // An anonymous class with Demo as base class
    static Demo d = new Demo()
    {
        // Method 1 show()
        void show()
        {
            // Calling method show() via super keyword
            // which refers to parent class
            super.show();

            System.out.println("I am in FlavorIDemo
                                    class");
        }
    };

    // Method 2
    // Main driver method
```

```
    public static void main (String[] args)
    {
        d.show();
    }
}
```

Output

```
I am in show method of super class
I am in FlavorDemo class
```

Way 2 : As an implementer of the specified interface

Example :

```
// Interface
interface Hello
{
    // Method defined inside interface
    void show();
}

class Main
{
    // Class implementing interface
    static Hello h = new Hello()
```

```
{
    //Method 1
    // show() method inside main class
    public void show()
    {
        System.out.println("I am in anonymous class");
    }
};

// Method 2
// Main driver method

public static void main(String[] args)
{
    // Calling show() method inside main()
    h.show();
}
}
```

Output

I am in anonymous class

Tanishq Chauhan
21C3184
CS-B

Q.10. Write short notes on following:

a. Bytecode in Java

Java bytecode is the instruction set for the Java Virtual Machine.
It acts similar to an assembler which is an alias representation of a C++ code.
As soon as a java program is compiled, java bytecode is generated.
Java bytecode is the machine code in form of a .class file.
With the help of java bytecode we achieve platform independence in java.

When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform we can do so.

Platform independence is one of the soul reason for which James Gosling started the formation of Java and it is this implementation of bytecode which help us to achieve this.

Hence bytecode is a very important component of any Java program.

Tanisha Chauhan
2103184
CS-B

b. Security

- When it comes to security, Java is always the first choice.
- With Java secure features it enables us to develop virus free, temper free system.
- Java program always run in Java runtime environment with almost null interaction with system OS, hence it is more secure.
- Java is more secure due to the following reasons:

(i) Java program run inside a virtual machine which is known as a sandbox.
(ii) Java does not support explicit pointer.
(iii) Byte-code verifier checks the code fragments for illegal code that can violate access right to object.
(iv) It provides java.security package implements explicit security.
(v) It provides library level safety
(vi) Run time security check takes place when we load new code.
(vii) It provides auto-memory manggement.
(viii) It provides exception handling.
(ix) It has security APIs.

Tanishq Chauhan
21C3184
CS-B

c. Garbage Collection

In Java, garbage means unreferenced objects.
Garbage collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
To do so, we were using free() function in C language and delete() in C++. But in Java it is performed automatically. So, Java provides better memory management.

Garge collection in Java is the process by which Java programs perform automatic memory management. Java program compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collecter finds these unused objects and deletes them to free up memory.

Advantage of Garbage Collection

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

Tanishq Chauhan
21C3184
CS-B

d. Applets

- An applet is a Java program that can be embedded into a web page.
- It runs inside the web browser and works at client side.
- An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server.
- Applets are used to make the website more dynamic and entertaining.
- Applet does not require a main function for its execution.
- Applet can only access the browser specific services. They don't have access to the local system.
- There are five essential methods that are needed for the life of the applet : init(), start(), paint(), stop() and destroy().
- Applet provides the web document security as well as the platform independency.

Advantages of Applet

- It works at client side so less responce time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac OS etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.