Experiment No. 2

Implementation of a problem statement/s using class and object

---

Instructions:

This manual consists of three parts: **A) Theory and Concepts, B) Problems for Implementation, and C) Write-up Questions.**

1. Students must understand the **theory and concepts** provided before implementing the problem statement(s) for **Experiment 2**.
2. They should **practice the given code snippets** within the theory section.
3. Later, they need to **implement the problems provided**.
4. **Write-up:** Students are required to **write answers** to the questions on journal pages, **maintain a file**, and get it checked regularly. The file should include index, write-up, and implementation code with results.
5. Referencing: Include proper sources or references for the content used.
6. Use of Generative AI: Clearly mention if you have used any AI tools (e.g., ChatGPT, Copilot, Bard) to generate text, explanations, or code. Cite the AI-generated content appropriately in the write-up.

Part A. Theory and Concepts:

- **Object**: Objects have states and behaviors. Example: A dog has states—color, name, breed—and behaviors such as wagging its tail, barking, and eating. An object is an instance of a class.
- **Class**: A class is a template/blueprint that describes the behaviors and states that objects of its type support.
- **Methods**: A method is a behavior. A class can contain many methods. It is within methods where logic is written, data is manipulated, and actions are executed.
- **Instance Variables**: Each object has its unique set of instance variables. An object's state is defined by the values assigned to these instance variables.

Basic **Syntax:**

In Java programs, the following points are important to keep in mind:

- **Case Sensitivity**: Java is case-sensitive, which means identifiers such as **Hello** and **hello** have different meanings in Java.
- **Class Names**: Class names should start with an uppercase letter. If multiple words are used, the first letter of each inner word should be uppercase. Example: **class MyFirstJavaClass**
- **Method Names**: Method names should start with a lowercase letter. If multiple words are used, the first letter of each inner word should be uppercase. Example: **public void myMethodName()**
- **Program File Name**: The program file name should match the class name exactly. When saving the file, it should be named using the class name and appended with **.java**. Example: If the class name is **MyFirstJavaProgram**, the file should be saved as **MyFirstJavaProgram.java**.
- **public static void main(String args[])**: The Java program starts from the **main()** method, which is mandatory for every Java program.

Constructor:

A constructor initializes an object immediately upon creation. It has the same name as the class it resides in and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes. There are three types of constructors in Java:

1. **Default Constructor**
2. **No-Args Constructor**
3. **Parameterized Constructor**

```
//Hello World Without Constructor:

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

```
//Hello world With Constructor:

public class HelloWorldC {
```

```
    // Constructor
    public HelloWorldC() {
        System.out.println("Hello, World!");
    }

    public static void main(String[] args) {
        // Creating an object of HelloWorld class to invoke the
constructor
        new HelloWorldC();
    }
}
```

```
//Hello world: writing constructor after the main().

public class HelloWorld {
    public static void main(String[] args) {
        // Creating an object of HelloWorld to invoke the
constructor
        new HelloWorld();
    }

    // Constructor (written after main)
    public HelloWorld() {
        System.out.println("Hello, World!");
    }
}
```

```
Output:
Hello, World!
```

Default Constructor in Java:

It is not always required to provide a constructor in the class code. If no constructor is provided, Java provides a default constructor implementation. A Default Constructor is a constructor that does not take any parameters and initializes objects with default values.

```
public class Data {
    public static void main(String[] args) {
        Data d = new Data();
    }
}
```

```
//1. Example of Default Constructor in Java

public class Car {
    String model;
    int year;
```

```
    // Default Constructor
    public Car() {
        model = "Unknown";
        year = 0;

        // model = "Toyota Innova";
        // year = 2024;
    }

    public static void main(String[] args) {
        // Creating an object of Car using the default constructor
        Car myCar = new Car();

        // Printing the values
        System.out.println("Car Model: " + myCar.model);
        System.out.println("Manufacturing Year: " + myCar.year);
    }
}
```

Output:

```
Car Model: Unknown
Manufacturing Year: 0
```

1. The default constructor's role is to initialize the object and return it to the calling code.
2. It is always without arguments and is provided by the Java compiler when no other constructor is defined.
3. Often, the default constructor is sufficient, as other properties can be initialized using getter and setter methods.

No-Argument Constructor:

A constructor without any argument is called a no-argument constructor. It is used for pre-initialization tasks such as checking resources, network connections, logging, etc.

```
public class Data {
    public Data() {  // No-args constructor
        System.out.println("No-Args Constructor");
    }
    public static void main(String[] args) {
        Data d = new Data();
    }
}
```

// 2.Example of Default Initialization Using No-Argument Constructor

```
public class Car {
    String model;
    int year;

    // No-Argument Constructor
    public Car() {
      model = "Unknown";
      year = 2024;
      System.out.println("Car object created with default values!");
    }

    public static void main(String[] args) {
        Car myCar = new Car();
        System.out.println("Model: " + myCar.model + ", Year: " +
myCar.year);
    }
}
```

Output:

```
Car object created with default values!
Model: Unknown, Year: 2024
```

Parameterized Constructor:

A constructor with arguments is called a parameterized constructor.

```
public class Data {
    private String name;
    public Data(String n) {
        System.out.println("Parameterized Constructor");
        this.name = n;
    }
    public String getName() {
        return name;
    }
    public static void main(String[] args) {
        Data d = new Data("Java");
        System.out.println(d.getName());
    }
}
```

3. Example of Parameterized Constructor

```
public class Car {
    String model;
    int year;
```

```
    //Parameterized Constructor
    public Car(String m, int y) {
        model = m;
        year = y;
    }

    // Using "this" keyword
    // public Car(String model, int year) {
    //     this.model = model;
    //     this.year = year;
    // }

    public static void main(String[] args) {
    //Creating an object of Car using the parameterized constructor
        Car myCar = new Car("Tesla", 2023);

        // Printing the values
        System.out.println("Car Model: " + myCar.model);
        System.out.println("Manufacturing Year: " + myCar.year);
    }
}
```

Output:

```
Car Model: Tesla
Manufacturing Year: 2023
```

```
// 4. Example: Using a Parameterized Constructor for Multiple Cars
// To go beyond default values and handle multiple cars with
different details, you should use a Parameterized Constructor. This
allows you to create objects with specific values rather than
relying on default values.

public class Car {
    String model;
    int year;

    // Parameterized Constructor
    public Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    public void displayCar() {
        System.out.println("Car Model: " + model);
```

```
        System.out.println("Manufacturing Year: " + year);
    }

    public static void main(String[] args) {
        // Creating multiple Car objects with different values
        Car car1 = new Car("Toyota Corolla", 2022);
        Car car2 = new Car("Honda Civic", 2020);
        Car car3 = new Car("Ford Mustang", 2023);

        // Displaying details of each car
        car1.displayCar();
        System.out.println();
        car2.displayCar();
        System.out.println();
        car3.displayCar();
    }
}
```

Output:

```
Car Model: Toyota Corolla
Manufacturing Year: 2022

Car Model: Honda Civic
Manufacturing Year: 2020

Car Model: Ford Mustang
Manufacturing Year: 2023
```

Static Variable:

A static variable is shared among all instances (or objects) of the class because it is a class-level variable. Only a single copy of the static variable is created and shared among all instances of the class. Memory allocation for such variables happens once when the class is loaded into memory.

```
static data_type variable_name;
```

Static Variable Initialization:

- Static variables are initialized when the class is loaded.
- Static variables are initialized before any objects of the class are created.
- Static variables are initialized before any static method of the class executes.
- Default values for static and non-static variables are the same:

- Primitive integers (long, short, etc.): **0**
- Primitive floating points (float, double): **0.0**
- Boolean: **false**
- Object references: **null**

Static Blocks:

A static block gets executed exactly once when the class is first loaded.

```
// Java program to demonstrate use of static blocks

class Test {
    static int a = 10;
    static int b;
    // Static block
    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }
    public static void main(String[] args) {
        System.out.println("from main");
        System.out.println("Value of a: " + a);
        System.out.println("Value of b: " + b);
    }
}
```

```
Output:

Static block initialized.
from main
Value of a: 10
Value of b: 40
```

Static Method:

When a method is declared with the **static** keyword, it is known as a static method. The most common example of a static method is the **main()** method.

```
class StaticDemo {
    static int age;
    static String name;
    static void disp() {
        System.out.println("Age is: " + age);
        System.out.println("Name is: " + name);
    }
     public static void main(String args[]) {  // This is also a
     static method
        age = 30;
        name = "Ram";
        disp();
```

```
    }
}
```

```
Output:

Age is: 30
Name is: Ram
```

Strings in Java:

A **String** is an object that represents a sequence of characters. The **java.lang.String** class is used to create a string object.

There are two ways to create a String object:

1. By string literal
2. By the **new** keyword

1) String Literal:

A string literal is created by using double quotes. For example:

String s = "welcome";

Each time you create a string literal, the JVM checks the string constant pool. If the string already exists, it returns a reference to the existing instance. Otherwise, it creates a new string and adds it to the pool.

String s1 = "Welcome";
String s2 = "Welcome"; // No new object created

In this example, only one object is created. The JVM does not create a new object if "Welcome" is already in the pool but returns the reference to the existing instance.

2) By new Keyword:

Using the **new** keyword creates a new string object in normal (non-pool) heap memory, and the literal is placed in the string constant pool.

String s = new String("Welcome"); // Creates two objects

String Example:

```
public class StringExample {
    public static void main(String args[]) {
      String s1 = "java"; // Creating string by Java string literal
      char[] ch = {'s', 't', 'r', 'i', 'n', 'g', 's'};
      String s2 = new String(ch);  // Converting char array to
      string
      String s3 = new String("example");  // Creating Java string by
      new keyword
         System.out.println(s1);
         System.out.println(s2);
         System.out.println(s3);
    }
}
```

Output:

```
java
strings
example
```

String Methods in Java:

1. **int length() -** Returns the number of characters in the string.
   "Hello".length(); // returns 5
2. **char charAt(int i) -** Returns the character at the ith index.
   "Hello".charAt(3); // returns 'l'
3. **String substring(int i) -** Returns the substring from the ith index character to the end.
   "Hello".substring(3); // returns "lo"
4. **String substring(int i, int j) -** Returns the substring from index i to j-1.
   "Hello".substring(1, 4); // returns "ell"
5. **String concat(String str) -** Concatenates the specified string to the end of this string.
   String s1 = "Hello";
   String s2 = "Hi";
   String output = s1.concat(s2); // returns "HelloHi"
6. **int indexOf(String s) -** Returns the index of the first occurrence of the specified string.
   String s = "Learn Share Learn";

int output = s.indexOf("Share"); // returns 6

7. **int indexOf(String s, int i) -** Returns the index of the first occurrence of the specified string, starting from the specified index i.
   String s = "Learn Share Learn";
   int output = s.indexOf("ea", 3); // returns 13
8. **int lastIndexOf(String s) -** Returns the index of the last occurrence of the specified string.
   String s = "Learn Share Learn";
   int output = s.lastIndexOf("a"); // returns 14
9. **boolean equals(Object otherObj) -** Compares this string to the specified object.
   boolean out = "Hello".equals("Hello"); // returns true
   boolean out = "Hello".equals("hello"); // returns false
10. **boolean equalsIgnoreCase(String anotherString) -** Compares two strings, ignoring case considerations.
    boolean out = "Hello".equalsIgnoreCase("Hello"); // returns true
    boolean out = "Hello".equalsIgnoreCase("hello"); // returns true
11. **int compareTo(String anotherString) -** Compare two strings lexicographically.
    int out = s1.compareTo(s2);
    // If out < 0: s1 comes before s2
    // If out = 0: s1 and s2 are equal
    // If out > 0: s1 comes after s2
12. **int compareToIgnoreCase(String anotherString) -** Compare two strings lexicographically, ignoring case considerations.
    int out = s1.compareToIgnoreCase(s2);
    // If out < 0: s1 comes before s2
    // If out = 0: s1 and s2 are equal
    // If out > 0: s1 comes after s2
13. **String toLowerCase() -** Converts all characters in the string to lowercase.
    String word1 = "HeLLo";
    String word3 = word1.toLowerCase(); // returns "hello"
14. **String toUpperCase() -** Converts all characters in the string to uppercase.
    String word1 = "HeLLo";
    String word2 = word1.toUpperCase(); // returns "HELLO"

15. **String trim() -** Returns a copy of the string with leading and trailing whitespaces removed.

    String word1 = " Learn Share Learn ";

    String word2 = word1.trim(); // returns "Learn Share Learn"

16. **String replace(char oldChar, char newChar) -** Returns a new string by replacing all occurrences of oldChar with newChar.

    String s1 = "fello";

    String s2 = s1.replace('f', 'h'); // returns "hello"

    // Note: s1 is still "fello", and s2 is "hello"

17. **boolean contains(String s) -** Returns true if the string contains the given substring.

    String s1 = "hellohi";

    String s2 = "hi";

    s1.contains(s2); // returns true

18. **char[] toCharArray() -** Converts the string to a new character array.

    String s1 = "hello";

    char[] ch = s1.toCharArray(); // returns ['h', 'e', 'l', 'l', 'o']

19. **boolean startsWith(String prefix) -** Returns true if the string starts with the specified prefix.

    String s1 = "hihello";

    String s2 = "hi";

    s1.startsWith(s2); // returns true

String Example Code for String Methods in Java:

```
//String Example for String Methods in Java
//Students are instructed to execute this code for each string
method independently


//String Example:
public class StringExample {
public static void main(String args[]) {
String s1 = "java";  // Creating string by Java string literal

char[] ch = {'s', 't', 'r', 'i', 'n', 'g', 's'};
String s2 = new String(ch);  // Converting char array to string

String s3 = new String("example");  // Creating Java string by new
keyword

char[] cha = {'k', 'a', 'p', 'i', 'l'};
String s4 = new String(cha);  // Converting char array to string

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
System.out.println(s4);


// String str = "Hello";  //create string
// int length = str.length();  // Store length of the string
// System.out.println("Length of the string: " + length);  //display
length

// (01)
// int length();
// Returns the number of characters in the string.
System.out.println("1. Length of 'Hello': " + "Hello".length());

// (02)
// char charAt(int i)
// Returns the character at the ith index.
System.out.println("2. Character at given position is': " +
"ABCDE".charAt(3));

// (03)
// String substring(int i)
// Returns the substring from the ith index character to the end.
```

```
System.out.println("3. substring': " + "Hello".substring(3));

// (04)
// String substring(int i, int j)
// Returns the substring from index i to j-1.
// "Hello".substring(1, 4); // returns "ell"
System.out.println("4. substring from index': " +
"Hello".substring(1,4));


// (05)
//concat string
String string1 = "Hello";
String string2 = "Hi";
String output1 = string1.concat(string2); // returns "HelloHi"
System.out.println("5. concat string': " + output1);

// (06)
//index of string
String s = "Learn Share Learn";
int output2 = s.indexOf("are"); // returns 6
System.out.println("6. index of string': " + output2);


// (07)
// int indexOf(String s, int i)
// Returns the index of the first occurrence of the specified
string, starting from the specified index i.
String string3 = "Learn Share Learn";
int output3 = string3.indexOf("ea", 3); // returns 13
System.out.println("7. index of string first occ':" + output3);


// (08)
// int lastIndexOf(String s)
// Returns the index of the last occurrence of the specified string.
String string4 = "Learn Share Learn";
int output4 = string4.lastIndexOf("a"); // returns 14
System.out.println("8. index of string last occ':" + output4);


// (09)
// boolean equals(Object otherObj)
// Compares this string to the specified object.
boolean out1 = "Hello".equals("Hello"); // returns true
boolean out2 = "Hello".equals("hello"); // returns false
boolean out3 = "kapil".equals("kapil"); // returns false
System.out.println("9. boolean: " + out1);
```

```
System.out.println("9. boolean: " + out2);
System.out.println("9. boolean: " + out3);

// (10)
// boolean equalsIgnoreCase(String anotherString)
// Compares two strings, ignoring case considerations.
boolean out4 = "Hello".equalsIgnoreCase("Hello"); // returns true
boolean out5 = "Hello".equalsIgnoreCase("hello"); // returns true
System.out.println("10. boolean: " + out4);
System.out.println("10. boolean: " + out5);


// (11)
// int compareTo(String anotherString)
// Compares two strings lexicographically.

String string5 = "HI";
String string6 = "Hello";
int out6 = string5.compareTo(string6);
System.out.println("11. compare: " + out6);
// If out < 0: s1 comes before s2
// If out = 0: s1 and s2 are equal
// If out > 0: s1 comes after s2

// (12)
// int compareToIgnoreCase(String anotherString)
// Compares two strings lexicographically, ignoring case
considerations.
int out7 = string5.compareToIgnoreCase(string6);
System.out.println("12. compare: " + out7);
// If out < 0: s1 comes before s2
// If out = 0: s1 and s2 are equal
// If out > 0: s1 comes after s2

// (13)
// String toLowerCase()
// Converts all characters in the string to lowercase.
String word1 = "HeLLo";
String word2 = word1.toLowerCase(); // returns "hello"
System.out.println("13. convert to lowercase: " + word2);

// (14)
// String toUpperCase()
// Converts all characters in the string to uppercase.
// String word1 = "HeLLo";
String word3 = word1.toUpperCase(); // returns "HELLO"
System.out.println("14. convert to UPPER Case: " + word3);
```

```
// (15)
// String trim()
// Returns a copy of the string with leading and trailing
whitespaces removed.
String word4 = " Learn Share Learn ";
String word5 = word4.trim(); // returns "Learn Share Learn"
System.out.println("15. Trim white space: " + word5);


// (16)
// String replace(char oldChar, char newChar)
// Returns a new string by replacing all occurrences of oldChar with
newChar.
String word6 = "fello";
String word7 = word6.replace('f', 'h'); // returns "hello"
// Note: s1 is still "fello", and s2 is "hello"
System.out.println("16. String Replace: " + word7);

// (17)
// boolean contains(String s)
// Returns true if the string contains the given substring.
String word8 = "hellohi";
String word9 = "hi";
word8.contains(word9); // returns true
System.out.println("17. String substirng: " +
word8.contains(word9));

// (18)
// char[] toCharArray()
// Converts the string to a new character array.
String str1 = "hello";
char[] char1 = str1.toCharArray();  // returns ['h', 'e', 'l', 'l',
'o']
System.out.println("18. char to array: " + char1);
System.out.println("18. char to array: " +
java.util.Arrays.toString(char1));

//19
// boolean startsWith(String prefix)
// Returns true if the string starts with the specified prefix.
String str2 = "hihello";
String str3 = "hi";
// String str4 = str2.startsWith(str3); // returns true
System.out.println("19. char to array: " + str2.startsWith(str3));
}
}
```

```
//Below are combined outputs.
//Students are instructed to execute this code for each string
method independently

Output:

java
strings
example
kapil
1. Length of 'Hello': 5
2. Character at given position is': D
3. substring': lo
4. substring from index': ell
5. concat string': HelloHi
6. index of string': 8
7. index of string first occ':13
8. index of string last occ':14
9. boolean: true
9. boolean: false
9. boolean: true
10. boolean: true
10. boolean: true
11. compare: -28
12. compare: 4
13. convert to lowercase: hello
14. convert to UPPER Case: HELLO
15. Trim white space: Learn Share Learn
16. String Replace: hello
17. String substirng: true
18. char to array: [C@6e0be858
18. char to array: [h, e, l, l, o]
19. char to array: true
```

Part B. Problems for Implementation:

Aim: Implementation of a given problem statement/s using *classes* and *objects*.

1. Create a class called **Employee** that includes three pieces of information as instance variables: first name, last name, and monthly salary. Your class should have a constructor that initializes the three instance variables. Provide a setter and getter method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named **EmployeeTest** that demonstrates the **Employee** class's capabilities. Create two **Employee** objects and display each object's yearly salary. Then give each **Employee** a 10% raise and display each **Employee's** yearly salary again.

2. **Implement a Java program to print the area of a rectangle** by creating a class named '**Area**' that has two methods. The first method, named '**setDim**', takes the length and breadth of the rectangle as parameters. The second method, named '**getArea**', returns the area of the rectangle. The length and breadth of the rectangle are entered through the keyboard.

3. Write a Java program to demonstrate the use of static variables, static blocks, and static methods.

4. Write a Java program to implement a stack and a queue.

5. Write a Java program to arrange 10 names in alphabetical order**.**

Part C. Write-up Questions:

1. Explain briefly object oriented programming concepts.
2. Explain class and object with simple examples.
3. Explain constructor, its types and constructor overloading with simple java examples.
4. Explain static variable, static block and static method.
5. Explain 10 different methods used on strings

Conclusion:

Students should be able to write Java programs using classes and objects after completing this content.