TECHNICAL REPORT WRITING

Java CA-2 Assignment

NAME: RUPAK SARKAR

ROLL NO.: 14271024036

STREAM: MASTER OF COMPUTER APPLICATION

SUBJECT NAME: OBJECT ORIENTED PROGRAMMING WITH JAVA

SUBJECT CODE: MCAN-203

COLLEGE NAME: MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

UNIVERSITY NAME: MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

ABSTRACT

Object-Oriented Programming (OOP) is a fundamental paradigm in software development that enhances code reusability, maintainability, and scalability. Java, being a fully object-oriented language, provides robust features that support OOP principles such as encapsulation, inheritance, polymorphism, and abstraction. This paper explores the key concepts of OOP in Java, detailing their implementation, benefits, and real-world applications. Through practical examples and case studies, we analyze how Java leverages OOP to develop modular, efficient, and secure software solutions. Additionally, we discuss best practices for applying OOP in Java, common pitfalls, and their mitigations. This study aims to provide a comprehensive understanding of OOP in Java, aiding developers in writing structured and maintainable code.

CONTENTS

INTRODUCTION

Object-Oriented Programming (OOP) is a fundamental programming paradigm that enhances software design by structuring code around objects, which encapsulate both data and behavior. By promoting modularity, reusability, and scalability, OOP simplifies complex software development. Java, being a widely used object-oriented language, fully supports OOP principles such as **encapsulation, inheritance, polymorphism, and abstraction**, enabling developers to create efficient and maintainable applications.

A **stack** is a linear data structure that follows the **Last In, First Out (LIFO)** principle, meaning the last element added is the first to be removed. Stacks are widely used in programming for tasks such as expression evaluation, function calls, and backtracking algorithms. In Java, stacks can be implemented using **classes and objects**, where a class defines the stack's properties and operations, and objects provide instances to perform stack-related functions like push, pop, and peek.

This report explores the core concepts of OOP in Java and demonstrates how to implement a stack using classes and objects. It covers the **advantages of OOP in data structure implementation**, real-world applications of stacks, and best practices for writing modular and efficient Java code. Understanding these principles allows developers to build structured and reusable solutions for complex problems.

TECHNICAL REPORT

Stack using Class and Object in Java.

```
class Stack
{
   int top;
   int stck[] = new int[10];
   Stack()
   {
      top = -1;
   }
   void push(int item)
   {
      if(top == (stck.length-1))
      {
         System.out.println("Stack is Full!");
      }
      else
      {
         stck[++top]=item;
      }
   }
   int pop()
   {
      if(top==-1)
      {
         System.out.println("Stack is Empty!");
         return 0;
      }
```

```
        else
        {
            return stck[top--];
        }
    }
}
class StackDemo
{
    public static void main(String args[])
    {
        Stack s1 = new Stack();
        int k=10;
        for (int i=0;i<10;i++)
        {
            s1.push(k);
            k++;
        }
        for (int i=0;i<10;i++)
        {
            System.out.print(s1.pop()+" ");
        }
    }
}
```

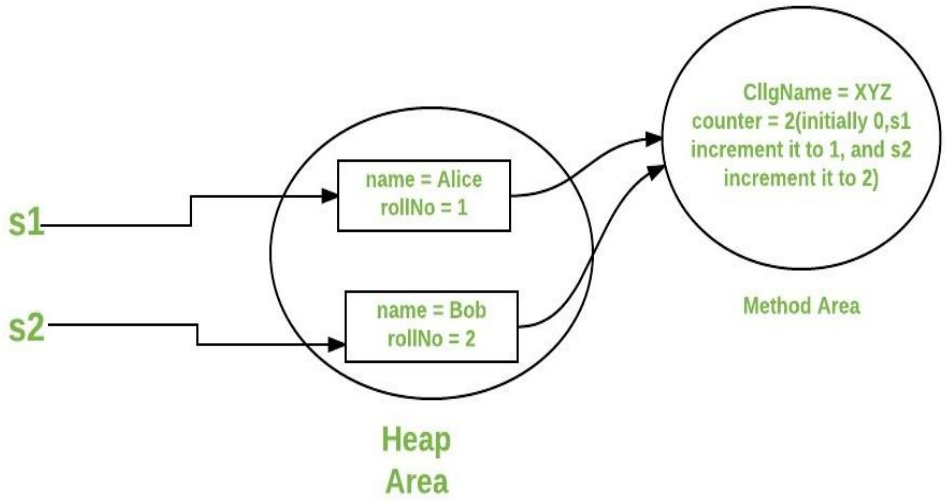
Output:

```
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> javac StackDemo.java
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> java StackDemo
19 18 17 16 15 14 13 12 11 10
```

Use of 'Static' keyword in Java –

The **static** keyword in Java is used for memory management and is applicable to variables, methods, blocks, and nested classes. When a member is declared as static, it belongs to the class rather than an instance, meaning it is shared among all objects of that class instead of being replicated for each object.

Static variable belongs to the class and not the object. When a member is declared as static it can be accessed before any of objects of its class are created and without reference to an object. We can declare both methods and variables to be static.



Instant variable declared as static are global variables. When objects of its class are created, no copy of a static variable is made. So, instead all instances of the class share the same static variable.

```
class StaticDemo
{
   static int a=3;
   static int b;
   static void meth(int x)
   {
      System.out.println("x: "+x);
      System.out.println("a: "+a);
      System.out.println("b: "+b);
   }
   static
   {
      System.out.println("Static block initialized");
      b=a*4;
   }
   public static void main(String args[])
   {
      meth(42);
   }}
```

Output:

```
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> javac StaticDemo.java
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> java StaticDemo
Static block initialized
x: 42
a: 3
b: 12
```

As soon as static demo class is loaded all static statements are run. First a is set to 3, then the static block is executes printing the message and finally b is initialized with the value a*4=>12.

After that, main is called, which calls **meth()**, passing 42 to x. Then all three print statements are executed.

Restrictions of Static

- They can call other static methods.
- They must only static data.
- They cannot refer to 'this' or 'super' in any way.

Use of 'this' keyword in Java –

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines this 'this' keyword. 'this' can be used inside any method to refer to the current object. That is 'this' is always a reference to the object on which the method was invoked.

Code:

```
class Box
{
   double height;
   double width;
   double depth;
   void setdata(double h, double w, double d)
   {
      this.height = h;
      this.width = w;
      this.depth = d;
   }
```

```
    double vol()
    {
        return height*width*depth;
    }
}
class BoxDemo {
    public static void main(String args[]) {
        Box B1 = new Box();
        B1.setdata(3, 4, 5);
        Box B2 = new Box();
        B2.setdata(6, 7, 8);
        System.out.println("Volume of Box B1: " + B1.vol());
        System.out.println("Volume of Box B2: " + B2.vol());
    }
}
```

Output:

```
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> javac BoxDemo.java
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> java BoxDemo
Volume of Box B1: 60.0
Volume of Box B2: 336.0
```

Implementation of Time Class with Addition Method Returning a Time Object

Code:

```
class Time
{
    int hr, min, sec;
    Time() {
        hr = 0;
        min = 0;
        sec = 0;
    }
    Time(int h, int m, int s)
    {
```

```
        hr = h;
        min = m;
        sec = s;
    }
    Time add(Time T1, Time T2)
    {
        this.sec = T1.sec + T2.sec;
        this.min = T1.min + T2.min + (this.sec / 60);
        this.sec %= 60;
        this.hr = T1.hr + T2.hr + (this.min / 60);
        this.min %= 60;
        return this;
    }
    void display()
    {
        System.out.println(hr + " hr " + min + " min " + sec + " sec");
    }
}
public class TimeDemo
{
    public static void main(String args[])
    {
        Time T1 = new Time(2, 45, 50);
        Time T2 = new Time(1, 20, 30);
        Time T3 = new Time();
        T3 = T3.add(T1, T2);
        System.out.print("Sum of T1 and T2: ");
        T3.display();
    }
}
```

Output:

```
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> javac TimeDemo.java
PS C:\Users\rupak\OneDrive\Desktop\OOPS-JAVA> java TimeDemo
Sum of T1 and T2: 4 hr 6 min 20 sec
```

CONCLUSION

In this report, we explored fundamental concepts of Java programming, including class implementation, constructors, static and 'this' keywords, and object manipulation. The Stack class was implemented with a user-defined size, demonstrating the use of constructors, push(), and pop() methods for efficient stack operations. The significance of the static keyword in memory management and shared class-level attributes was discussed, along with practical examples.

Additionally, we examined the 'this' keyword, which helps in distinguishing instance variables from method parameters and referring to the current object. Finally, we implemented a Time class, showcasing how to add two Time objects using an add() method that returns a new Time object, emphasizing object-oriented principles and method operations in Java.

Through these implementations, we gained a deeper understanding of object-oriented programming (OOP) concepts, memory management, and Java's core features, which are essential for developing efficient, structured, and maintainable applications.

ACKNOWLEDGEMENT

I would like to acknowledge all those without whom this report would not have been successful. Firstly, I would wish to thank my teacher Mr. Soumya Chakravarty who guided me throughout the report and gave his immense support. He made us understand how to successfully complete this report and without him, the report would not have been complete.

I would also like to thank my parents who have always been there whenever needed. Once again, thanks to everyone for making this report successful.