Lab 05: Methods & Recursion

Objective(s):

1. Method Overloading
2. Pass by value & Pass by Reference
3. Using Object as a parameter
4. Returning objects from Methods
5. Recursion

1: Method Overloading

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different parameters.

OR

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.

Example:

```
// overloading in Java.
public class Sum {

    // Overloaded sum(). This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }

    // Overloaded sum(). This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }

    // Overloaded sum(). This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }

    // Driver code
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Three ways to overload a method

In order to overload a method, the parameters of the methods must differ in either of these:

1. Number of parameters.

   For example: This is a valid case of overloading

   ```
   add(int, int)
   add(int, int, int)
   ```

2. Data type of parameters.

   For example:

   ```
   add(int, int)
   add(int, float)
   ```

3. Sequence of Data type of parameters.

   For example:

```
add(int, float)
add(float, int)
```

Invalid case of method overloading:

If two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

```
int add(int, int)
float add(int, int)
```

2: Call by Value and Call by Reference

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }

 public static void main(String args[]){
   Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);

 }
}
```

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }


 public static void main(String args[]){
   Operation2 op=new Operation2();

   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

3: Passing object as a parameter

Objects in java are reference variables, so for objects a value which is the reference to the object is passed. Hence the whole object is not passed but its referenced gets passed. All modification to the object in the method would modify the object in the Heap.

```
class Add {
  int a;
  int b;

  Add(int x, int y) // parametrized constructor
  {
    a = x;
    b = y;
  }
  void sum(Add A1) // object  'A1' passed as parameter in function 'sum'
  {
    int sum1 = A1.a + A1.b;
    System.out.println("Sum of a and b :" + sum1);
  }
}

public class Main {
  public static void main(String arg[]) {
    Add A = new Add(5, 8);
    /* Calls  the parametrized constructor
    with set of parameters*/
    A.sum(A);
  }
}
```

4: Returning object as a parameter

Like any other data datatype, a method can returns object.

```
// Returning an object.
class Test {
  int a;

  Test(int i) {
    a = i;
  }

  Test incrByTen() {
    Test temp = new Test(a+10);
    return temp;
  }
}

class RetOb {
  public static void main(String args[]) {
    Test ob1 = new Test(2);
    Test ob2;

    ob2 = ob1.incrByTen();
    System.out.println("ob1.a: " + ob1.a);
    System.out.println("ob2.a: " + ob2.a);

    ob2 = ob2.incrByTen();
    System.out.println("ob2.a after second increase: "
                       + ob2.a);
  }
}
```

4: Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. It does this by making problem smaller (simpler) at each call.

Recursive functions have two important components:

1. **Base case(s)**, where the function directly computes an answer without calling itself. Usually the base case deals with the simplest possible form of the problem you're trying to solve. The base case returns a value without making any subsequent recursive calls. It does this for one or more special input values for which the function can be evaluated without recursion.
2. **Recursive case(s)**, where the function calls itself as part of the computation.

Perhaps the simplest example is calculating factorial: $n! = n \cdot (n-1) \cdot \cdots \cdot 2 \cdot 1$. However, we can also see that $n! = n \cdot (n-1)!$. Thus, factorial is defined in terms of itself. For example,

factorial(5) = 5 * factorial(4)
= 5 * (4 * factorial(3))
= 5 * (4 * (3 * factorial(2)))
= 5 * (4 * (3 * (2 * factorial(1))))
= 5 * (4 * (3 * (2 * (1 * factorial(0)))))
= 5 * (4 * (3 * (2 * (1 * 1))))
= 5 * 4 * 3 * 2 * 1 * 1 = 120

We can trace this computation in precisely the same way that we trace any sequence of function calls.

```
factorial(5)
   factorial(4)
      factorial(3)
         factorial(2)
            factorial(1)
               return 1
            return 2*1 = 2
         return 3*2 = 6
      return 4*6 = 24
   return 5*24 = 120
```

Lab Tasks:

Exercise1 — CalculateBMI.java

1. Define a class named **BMIAnalyzer** with the following instance variables:
   - **weight** (type: double) , **height** (type: double) To store the height in inches and weight of a person in pounds.
2. Implement a constructor in the **BMIAnalyzer** class that takes two parameters: weight and height, and initializes the instance variables accordingly.
3. Define an instance method named **calculateBMI()** inside the **BMIAnalyzer** class that calculates the BMI of a person based on the provided weight and height. Use the following formula:
   - BMI = weight (lb) / [height (in)]^2 * 703

4. Implement another instance method named **findStatus(double bmi)** inside the **BMIAnalyzer** class that categorizes the weight status of a person based on the provided BMI value. The method should return a String indicating one of the following weight statuses:

BMI	Weight Status
Below 18.5	Underweight
18.5 – 24.9	Normal
25.0-29.9	Overweight
30.0 and above	Obese

5. Create a **main** method to test your **BMIAnalyzer** class. Inside the **main** method:
   - Instantiate a **BMIAnalyzer** object with sample weight and height values.
   - Call the **calculateBMI()** method to calculate the BMI of the person.
   - Pass the calculated BMI to the **findStatus(double bmi)** method to determine the weight status.
   - Display the calculated BMI and weight status on the console.

Exercise2 — Sum.java

Write the following 2 methods:

```
public  int ComputeOddSum(int input)
public int ComputeEvenSum(int input)
```

The method **`ComputeOddSum`** find the sum of all odd numbers less than input.

The method **`ComputeEvenSum`** find the sum of all even numbers less than input.

Now, test these 2 methods by prompting the user to input a number each time until a negative number is entered.

Exercise 3 (Pass by Reference) — Account.java

1. Create a class called "BankAccount" with attributes such as account number, account holder name, and balance. Implement a method called "transferAmount" that takes two BankAccount objects as arguments, along with the amount to transfer, and transfers the specified amount from one account to another. Ensure that the balance is updated accordingly for both accounts. Test the method by creating two BankAccount objects, transferring amounts between them, and printing their updated balances.

Exercise 4 (Recursion) — Recursion.java

1. Write a recursive function to compute power of a number (X^n). Test and trace for 4^5? Hint: $4^5 = 4 * 4^4$; $4^0 = 1$.
2. Write a recursive function that calculates the factorial of a number.

Exercise 5 — GeometryUtil.java

Develop a class named **GeometryUtils** with overloaded methods to calculate properties of geometric shapes, such as area, perimeter, and volume. Implement methods like **calculateArea**, **calculatePerimeter**, and **calculateVolume** that accept different parameters for different shapes like (e.g., circle, rectangle, cube)

Supported Geometric Shapes:

1. **Circle:**
    - Parameters for area calculation: radius (double)
    - Parameters for perimeter calculation: radius (double)
2. **Rectangle:**
    - Parameters for area calculation: length (double), width (double)
    - Parameters for perimeter calculation: length (double), width (double)
3. **Cube:**
    - Parameters for volume calculation: side length (double)

END