Lab 04: Static Methods & Recursion

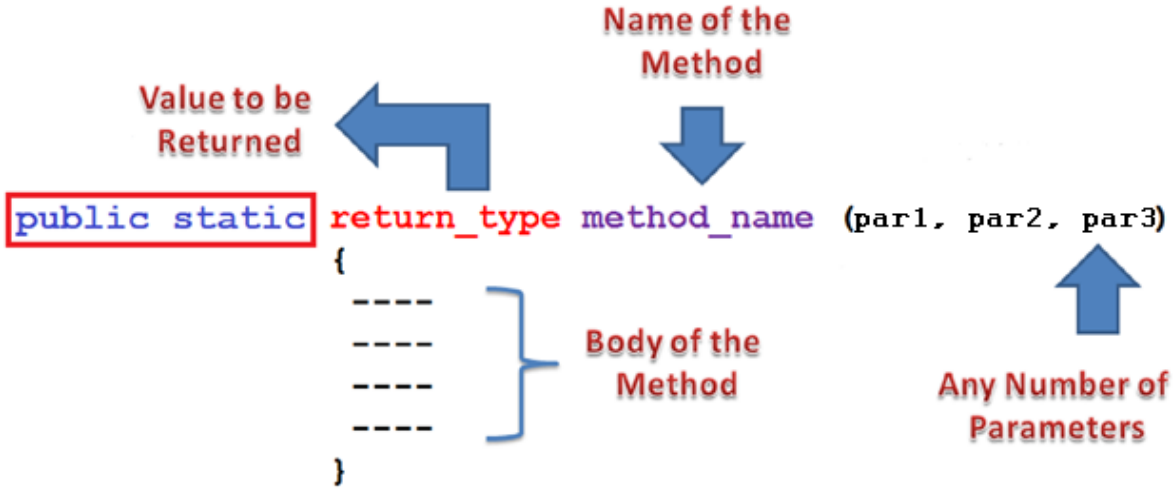
Objective(s):

1. Static Methods
2. Types of Static Methods
3. Method Overloading
4. Recursion

1: Static Methods

A method (function) is a group of statements that is executed when it is called from some point of the program.

A **static method** can be invoked without creating an object of a class. The following is its format:



Where:

- ***return_type*** is the data type specifier of the data returned by the function.
- ***method_name*** is the identifier by which it will be possible to call the function.
- ***parameters*** (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.

- ***statements*** is the function's body. It is a block of statements surrounded by {}

```
public class Lab04 {
    public static void main(String [] args){
        for(int i=1;i<30;i++)
        System.out.print("*");

        System.out.println();
        System.out.println("Object Oriented Programming");

        for(int i=1;i<30;i++)
        System.out.print("*");

        System.out.println();
        System.out.println("Lab 04");

        for(int i=1;i<30;i++)
        System.out.print("*");

        System.out.println();
        System.out.println("Department of Computer Science");

        for(int i=1;i<30;i++)
        System.out.print("*");
        System.out.println();
    }
}
```

2: Type of static Methods

Methods without Return Type and No Parameter

```
public static void method_name ( )
```

What we will do in this case for similar functionality code, create a function named drawLine() and call that function at repeated code locations.

```
public class Lab04 {
    public static void main(String [] args){
    drawLine();
    System.out.println("Object Oriented Programming");
    drawLine();
    System.out.println("Lab 02");
    drawLine();
    System.out.println("Department of Computer Sciences");
```

```
    drawLine();
    }
   public static void drawLine(){
      for(int i=1;i<30;i++)
        System.out.print("*");
        System.out.println();
    }
}
```

Methods without Return Type and with Parameters

```
public static void method_name ( par1, par2, par3 )
```

drawLine method with parameter to draw line according to the size provided in an argument.

```
public class Lab04 {
public static void main(String [] args){
        drawLine(30);
        System.out.println("Object Oriented Programming");
        drawLine(10);
        System.out.println("Lab 04");
        drawLine(10);
        System.out.println("Department of Computer Sciences");
        drawLine(30);
    }
public static void drawLine(int n){
   for(int i=1;i<n;i++)
        System.out.print("*");
        System.out.println();
    }
}
```

Methods With Return Type but no Parameter

```
public static return_type method_name ( )
```

```
public class GetNameExample {
  public static void main(String [] args) {
    System.out.println("The  University Name is "+getUniversityName());
  }//main ends
  public static String getUniversityName(){
    return "Sukkur IBA University";
  }
}//class ends
```

Methods With Return Type and Parameters

```
public static return_type method_name (par1, par2, par3)
```

```
public class ArraySum {
   public static void main(String [] args) {
      int [] a = {2,3,5,8,4,9,7,6,7,8};
      int sum = findSum(a);//invoking method with array argument
      System.out.println("The result is "+sum);
   }//main ends
   public static int findSum(int[] b){
      int total=0;
      for(int i=0; i<b.length;i++){
          total+=b[i]; return total;
      }
   }
}//class ends
```

There are two types of parameter passing:

1. **Pass by Value**

```
public static void sum(int a, int b)
```

   - Call By value (copy of value) is when primitive data types are passed in the method call.

2. **Pass by Reference (value of memory address location)**

```
public static void displayCars(Car car)
public static void sort(int [] arrayB)
```

- Objects and object variables are passed by reference or address.
- **Notice that** in java when an array is passed as an argument, its memory address location (its "reference") is used.

3: 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)
```

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:

Exercise 1 — CalculateBMI.java

Write a Java application with the following prototypes that returns the user's body mass index (BMI)

public static double calcluateBMI(double weight, double height)

To calculate BMI based on weight in pounds (lb) and height in inches (in), use this formula:

$$\text{BMI} = \frac{\text{mass(lb)}}{(\text{height(in)})^2} \times 703$$

and

public static String findStatus(double bmi)

Categorizes it as underweight, normal, overweight, or obese, based on the table from the United States Centers for Disease Control:

BMI	Weight Status
Below 18.5	Underweight
18.5 – 24.9	Normal

25.0-29.9	Overweight
30.0 and above	Obese

Prompt the user to enter weight in pounds and height in inches.

Exercise 2 — Sum.java

Write the following 2 static methods:

```
public static int ComputeOddSum(int input)
public static 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 — MatrixTest.java

Create a class MatrixTest with the following two methods having two matrices:

a. **Sum** that accepts two two-dimensional arrays (matrices) as parameters and returns a two-dimensional array representing their sum.
b. **Product** that accepts two two-dimensional arrays (matrices) as parameters and returns a two-dimensional array representing their product.

$$M1=\begin{pmatrix}1 & 2 & 3\\ 3 & 0 & 1\\ 2 & 1 & 4\end{pmatrix} \text{ and } M2=\begin{pmatrix}5 & -10 & 6\\ 8 & 7 & -1\\ 0 & 3 & 2\end{pmatrix} \text{ Note that: } M1+M2=\begin{pmatrix}6 & -8 & 9\\ 11 & 7 & 0\\ 2 & 4 & 6\end{pmatrix} \text{ and } M1\times M2=\begin{pmatrix}21 & 13 & 10\\ 15 & -27 & 20\\ 18 & -1 & 19\end{pmatrix}$$

Exercise 4 (Recursion) — Power.java

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

Exercise 5 (Recursion) — Palindrome.java

Write a recursive method isPalindrome that takes a string and returns true if it is read forwards or backwards. For example, isPalindrome("mom") → true isPalindrome("cat") → false isPalindrome("level") → true

The prototype for the method should be as follows:

```
public static boolean isPalindrome(String str)
```

Exercise 6 Even Fibonacci Sum (Recursion) — EvenFibSum.java

Write a method that returns the sum of all even Fibonacci numbers using **recursion**. Consider all Fibonacci numbers that are less than or equal to n.

Each new element in the Fibonacci sequence is generated by adding the previous two elements.

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Method should be:

```
public Integer evenFibonacciSum(Integer n)
```

END