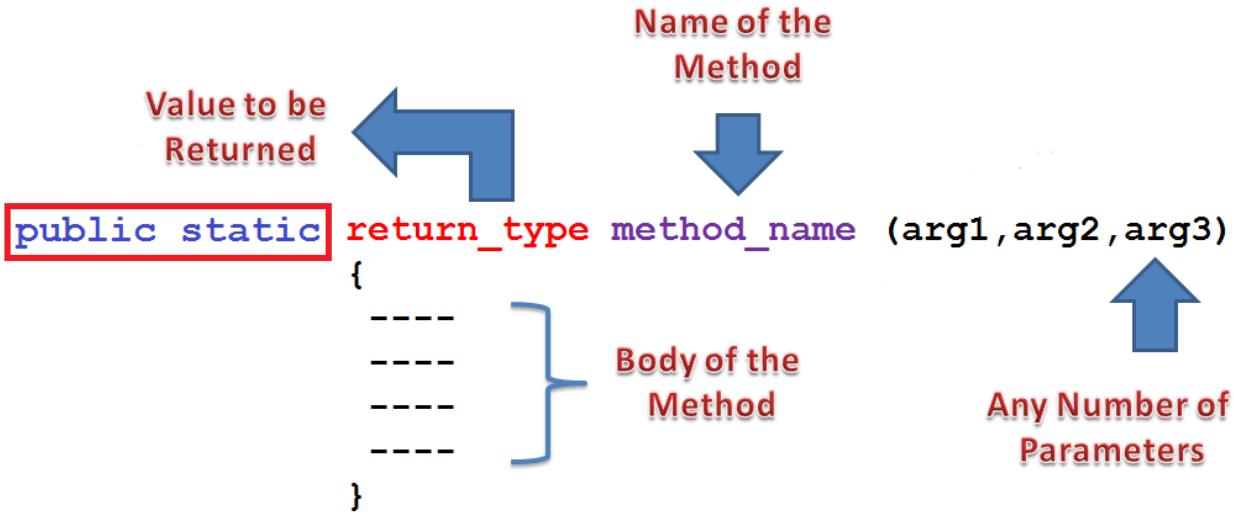
Lab03: Static Methods and Recursion

Designing and implementing Java programs that deal with:

1. **Static Methods**
2. **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 {}

Consider the following code and examine the repeated code with same functionality

```
public class Lab02 {
    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 02");
```

```
        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();
    }
}
```

1.1 Methods Without Return Type and No Argument

```
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 loctations.

```
public class Lab02 {
    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();
    }
}
```

1.2 Methods Without Return Type and with arguements

```
public static void method_name (arg1,arg2,arg3)
```

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

```
public class Lab02 {
    public static void main(String [] args){
        drawLine(30);
        System.out.println("Object Oriented Programming");
        drawLine(10);
        System.out.println("Lab 02");
        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();
    }
}
```

1.3 Methods With Return Type but no arguement

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 "Bahria University";
     }
}//class ends
```

1.4 Methods With Return Type and Arguments

public static return_type method_name (arg1,arg2,arg3)

```
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 valu) 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.

2. Method Overloading

When a class has two or more methods by same name but different parameters, it is known as method overloading.

```
public class MethodOverloading {
  public static void main(String[] args){
     add(11.5, 22.5);
     add(4,7,9);
     add("Life at ", "?");
     add(1,2);
  }//main ends
  public static void add(int a, int b){
     int sum = a + b;
     System.out.println("Sum of a+b is "+sum);
  }
  public static void add(int a, int b, int c){
     int sum = a + b + c;
     System.out.println("Sum of a+b+c is "+sum);
  }
  public static void add(double a, double b){
     double sum = a + b;
     System.out.println("Sum of a+b is "+sum);
  }
  public static void add(String s1, String s2){
     String s = s1+s2;
     System.out.println(s);
  }
}//class ends
```

3. Recursion

"A *recursive method* is a method that either directly or indirectly makes a call to itself." [Weiss]. It does this by making problem smaller (simpler) at each call.

Recursive functions have two important components:

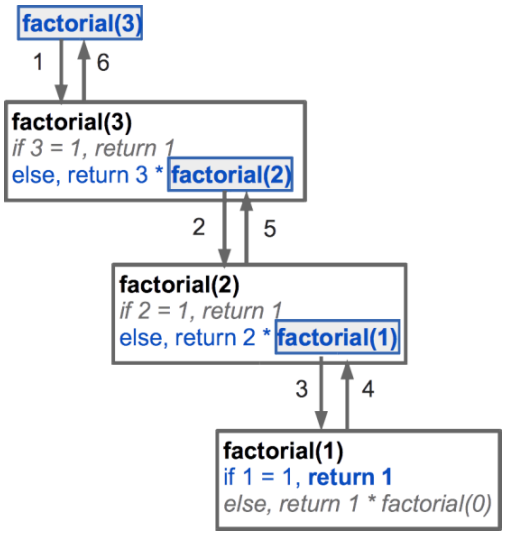
1. **Base case(s)**, where the function directly computes an answer without calling itself. Usually the base case deals with the simplest possible form of the problem you're trying to solve.
2. **Recursive case(s)**, where the function calls itself as part of the computation.

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
```

Iterative Solution	Recursive Solution

`public class FactRecursive {`
`pubic static viod main(String [] args){`
`System.out.println("Factorial of 3 is " +factorial(3));`
`}`
`static double factorial( double n ){`
`  double sum = 1.0;`
`  for( int i=1; i<=n; i++ )`
`  sum *= i;`
`  return sum;`
`}`
	`public class FactRecursive {`
	`pubic static viod main(String [] args){`
	`System.out.println("Factorial of 3 is " +factorial(3));`
	`}`
	`static double factorial( double n ){`
	`  if( n <= 1 )`
	`  return 1;`
	`  else`
	`  return n * factorial( n-1 );`
	`}`

`sum =1*1=1`
`sum =1*2=2`
`sum =2*3=6`
`return sum =6`
	

Exercises

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 to invoke the previous methods with the following two matrices:

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

Write a recursive method to get sum of all number from 1 up to given number. E.g. Number = 5 Result must be sum (1+2+3+4+5)

Exercise 5 (Recursion) *(Fibonacci.java)*

Write a recursive function to compute Ntn Fibonacci number. Test and trace for N = 6 is 8. We remember that a Fibonacci number can be recursively defined as:
$F(n) = F(n-1) + F(n-2)$ for $n \geq 2$, where $F(0) = 0, F(1) = 1$.

Exercise 6 (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 7 (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)
```