|  |  |
| --- | --- |
|  | **CSC-150 – Object Oriented Programming** |
| **Semester II (Spring 2023)**  **Course Instructor(s): Dr. Fahim Akhtar, Engr. Asif Ali** |

|  |
| --- |
| **Lab 05**  **Static Methods & Recursion** |

**Objective(s):**

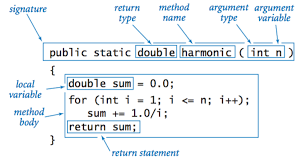
1. Static Methods
2. Types of Static Methods
3. Method Overloading
4. Recursion
5. Lab tasks
6. Post Lab questions

|  |
| --- |
| **1: Static Methods** |

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

**Static** methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class.  Static method belongs to the class rather than the 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 {}

//Java Program to demonstrate the use of a static method.

class Student{

int rollno;

String name;

static String college = "ITS";

//static method to change the value of static variable

static void change(){

college = "BBDIT";

}

//constructor to initialize the variable

Student(int r, String n){

rollno = r;

name = n;

}

//method to display values

void display(){System.out.println(rollno+" "+name+" "+college);}

}

//Test class to create and display the values of object

public class TestStaticMethod{

public static void main(String args[]){

Student.change();//calling change method

//creating objects

Student s1 = new Student(111,"Engr");

Student s2 = new Student(222,"Asif");

Student s3 = new Student(333,"Ali");

//calling display method

s1.display();

s2.display();

s3.display();

}

}

**Restrictions**

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

***Why is the Java main method static?***

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation

|  |
| --- |
| **2: Type of static Methods** |

**Methods without Return Type and No Parameter**



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 Lab05 {  public static void main(String [] args){  drawLine();  System.out.println("Object Oriented Programming");  drawLine();  System.out.println("Lab 05");  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**



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

|  |
| --- |
| public class Lab05 {  public static void main(String [] args){  drawLine(30);  System.out.println("Object Oriented Programming");  drawLine(10);  System.out.println("Lab 05");  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 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 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.
  1. **Pass by Reference (value of memory address location)**

|  |  |  |
| --- | --- | --- |
|  | public static void displayCars(Car car) |  |

* + - Objects and object variables are passed by reference or address.

|  |
| --- |
| **3:l 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) |

1. **Data type of parameters.**

For example:

|  |
| --- |
| add(int, int)  add(int, float) |

1. **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.

*A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.*

Recursive functions have two important components:

1. **Base case** (i.e., when to stop), 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** (i.e., call ourselves), where the function calls itself as part of the computation.

Perhaps the simplest example is calculating factorial: 𝑛! = 𝑛 ∙ (𝑛 − 1) ∙ ⋯∙ 2 ∙ 1. However, we can also see that 𝑛! = 𝑛 ∙ (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 |

*When a recursive call is made, new storage locations for variables are allocated on the stack*

**Lab Tasks**

Marks : 10, All Questions carry equal marks

Exercise 1 Bank.java

Write a program by creating Bank class having the following methods

getInfo() Which takes user name, type of account, and balance.

Cashdeposit( ) which takes the deposit amout, and date of deposit.

Withdraw( ) which takes the withdraw amount, and date of withdraw

Statement( ) that will display complete statement. Display final balance

Exercise 2 (Method Overloading) Methods.java

Write a program by creating class MethodsOverloading that contains following methods

AreaRectangle(int length, int width) It takes length and width and display Area of Rectangle.

Intro(String name, int age) It takes name, and age.Display name, and age.

Force(double mass, double accleration ) It takes mass and acceleartion and display force

Exercise 3 (Static Method) FindFirstLastDigit.java

Write a static method named lastDigit that returns the last digit of an integer. For example, FirstlastDigit(3852) should return 3 and 2.

Exercise 4 (Factorial) FactorialDetector.java

Write a program that takes input from the user. It tells the factorial number of the input.

Output

Enter any input: 24

24 is the factorial of : 4

Enter any input: 6

6 is the factorial of : 3

Enter any input: 10

10 has no any factorial!

Exercise 5 (Recursion) FibonacciSeries.java

Write a program that find fibonacci series using recursive function and also using loop. Also, takes input and check number at particular index of fibonacci series.

|  |
| --- |
| **Post lab questions to ponder** |

1. Do you know about access modifiers? Write down different types of access modifiers with suitable example.
2. What are non-access modifiers? Also describe all non-access modifiers in Java.
3. Can constructors be static in java? Try it out and justify.
4. Why use iterations when we have recursion and vice versa?
5. Can we overload by return type?

**END**