Khulna Khan Bahadur Ahsanullah University

Object-oriented programming

CSE 1203

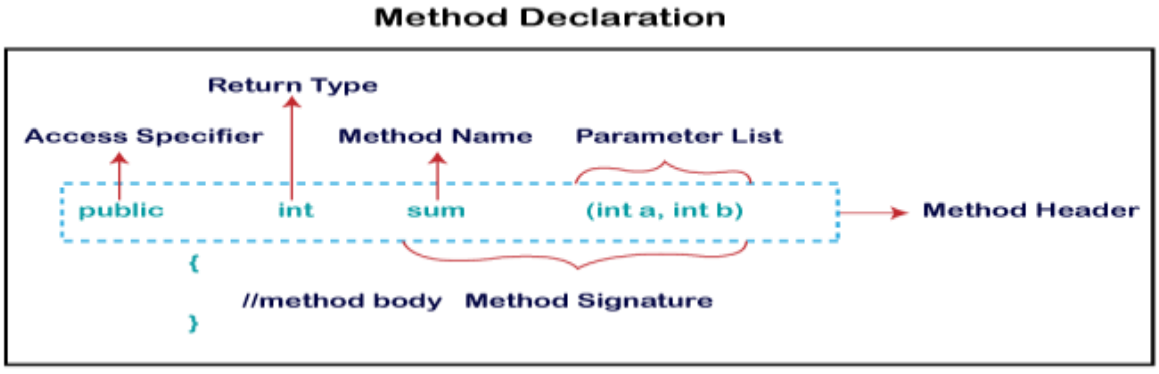
Lecture -10

1. What is method in Java?

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- It is used to achieve the **reusability** of code. We write a method once and use it many times.
- We do not require to write code again and again.
- It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.
- The most important method in Java is the **main()** method.

2. How to do declaration of method?

- The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.
- It has six components that are known as **method header**, as we have shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction ().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method: While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

Single-word method name: sum(), area()
Multi-word method name: areaOfCircle(), stringComparision().
It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

3. Discuss different types of Method.

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

- In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
- It is also known as the **standard library method** or **built-in method**.
- We can directly use these methods just by calling them in the program at any point.
- Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc.
- Each and every predefined method is defined inside a class.
- Such as **print()** method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method.
- For example, **print("Java")**, it prints Java on the console.

Demo.java

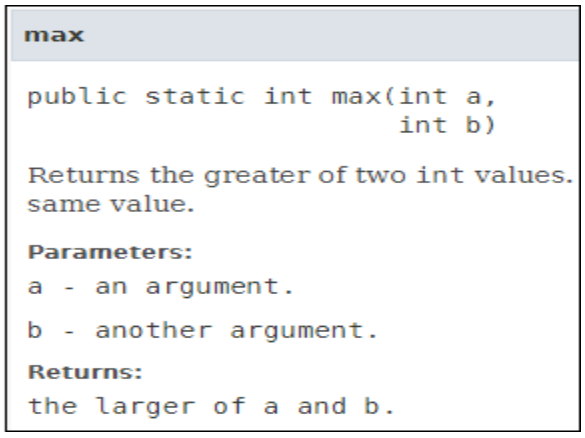
```
1. public class Demo {
2. public static void main(String[] args)  {
3. // using the max() method of Math class
4. System.out.print("The maximum number is: " + Math.max(9,7));
5. }  }
```

Output:

```
The maximum number is: 9
```

Explanation of the given code:

- In the above example, we have used three predefined methods **main(), print(),** and **max()**.
- We have used these methods directly without declaration because they are predefined.
- The print() method is a method of **PrintStream** class that prints the result on the console.
- The max() method is a method of the **Math** class that returns the greater of two numbers.
- We can also see the method signature of any predefined method by using the link https://docs.oracle.com/. When we go through the link and see the max() method signature, we find the following:



- In the above method signature, we see that the method signature has access specifier **public**, non-access modifier **static**, return type **int**, method name **max(),** parameter list **(int a, int b).** In the above example, instead of defining the method, we have just invoked the method. This is the advantage of a predefined method. It makes programming less complicated. Similarly, we can also see the method signature of the print() method.

User-defined Method

- The method written by the user or programmer is known as **a user-defined** method.
- These methods are modified according to the requirement.

How to Create a User-defined Method

- Let's create a user defined method that checks the number is even or odd.
- First, we will define the method.

```
//user defined method
public static void findEvenOdd(int num)
{
//method body
if(num%2==0)
System.out.println(num+" is even");
else
System.out.println(num+" is odd");
}
```

- We have defined the above method named findevenodd().
- It has a parameter **num** of type int.
- The method does not return any value that's why we have used void.
- The method body contains the steps to check the number is even or odd.
- If the number is even, it prints the number **is even**, else prints the number **is odd**.

How to Call or Invoke a User-defined Method

- Once we have defined a method, it should be called.
- The calling of a method in a program is simple.
- When we call or invoke a user-defined method, the program control transfer to the called method.

```
import java.util.Scanner;
public class EvenOdd
{
public static void main (String args[])
{
//creating Scanner class object
Scanner scan=new Scanner(System.in);
System.out.print("Enter the number: ");
//reading value from the user
int num=scan.nextInt();
//method calling
findEvenOdd(num);
}
```

- In the above code snippet, as soon as the compiler reaches at line **findEvenOdd(num),** the control transfer to the method and gives the output accordingly.

Let's combine both snippets of codes in a single program and execute it.

EvenOdd.java

```
import java.util.Scanner;
public class EvenOdd  {
public static void main (String args[])
{
Scanner scan=new Scanner(System.in);
System.out.print("Enter the number: ");
//reading value from user
int num=scan.nextInt();
//method calling
findEvenOdd(num);
}
//user defined method
public static void findEvenOdd(int num)
{
//method body
if(num%2==0)
System.out.println(num+" is even");
else
System.out.println(num+" is odd");
  }
}
```

Output 1:

```
Enter the number: 12
12 is even
```

Let's see another program that return a value to the calling method.

In the following program, we have defined a method named **add()** that sum up the two numbers. It has two parameters n1 and n2 of integer type. The values of n1 and n2 correspond to the value of a and b, respectively. Therefore, the method adds the value of a and b and store it in the variable s and returns the sum.

Addition.java

```
public class Addition
{
public static void main(String[] args)
{
int a = 19;
int b = 5;
//method calling
int c = add(a, b);   //a and b are actual parameters
System.out.println("The sum of a and b is= " + c);
}
//user defined method
public static int add(int n1, int n2)   //n1 and n2 are formal parameters
{
int s;
s=n1+n2;
return s; //returning the sum
}  }
```

Output: The sum of a and b is= 24

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Example of static method

Display.java

```
public class Display
{
public static void main(String[] args)
{
show();
}
static void show()
{
System.out.println("It is an example of static method.");
}  }
```

Output:

```
It is an example of a static method.
```

Instance Method

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

InstanceMethodExample.java

```
1. public class InstanceMethodExample
2. {
3. public static void main(String [] args)
4. {
5. //Creating an object of the class
6. InstanceMethodExample obj = new InstanceMethodExample();
7. //invoking instance method
8. System.out.println("The sum is: "+obj.add(12, 13));
9. }
10.int s;
11.//user-defined method because we have not used static keyword
12.public int add(int a, int b)
13.{
14.s = a+b;
15.//returning the sum
16.return s;
17.} }
```

Output:

```
The sum is: 25
```

Abstract Method

- The method that does not has method body is known as abstract method.
- In other words, without an implementation is known as abstract method.
- It always declares in the **abstract class**.
- It means the class itself must be abstract if it has abstract method.
- To create an abstract method, we use the keyword **abstract**.

Syntax

1. **abstract void** method_name();

Example of abstract method

Demo.java

```
1. abstract class Demo //abstract class
2. {
3. //abstract method declaration
4. abstract void display();
5. }
6. public class MyClass extends Demo
7. {
8. //method impelmentation
9. void display()
10.{
11.System.out.println("Abstract method?");
12.}
13.public static void main(String args[])
```

```
14.{
15.//creating object of abstract class
16.Demo obj = new MyClass();
17.//invoking abstract method
18.obj.display();
19.}
20.}
```

Output:

```
Abstract method...
```

Factory method

- It is a method that returns an object to the class to which it belongs.
- All static methods are factory methods.
- For example, **NumberFormat obj = NumberFormat.getNumberInstance();**

What is Constructors?

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new () keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
1. <class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
1. //Java Program to create and call a default constructor
2. class Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. public static void main(String args[]){
7. //calling a default constructor
8. Bike1 b=new Bike1();
9. } }
```

Output:

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
1. //Let us see another example of default constructor
2. class Student3{
3. int id;
4. String name;
5. void display(){System.out.println(id+" "+name);}
6. public static void main(String args[]){
7. //creating objects
8. Student3 s1=new Student3();
9. Student3 s2=new Student3();
10.//displaying values of the object
11.s1.display();
12.s2.display();
13.}  }
```

Output:

```
0 null
0 null
```

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of the parameterized constructor.
2. class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+" "+name);}
12.
13.    public static void main(String args[]){
14.    //creating objects and passing values
```

```
15.    Student4 s1 = new Student4(111,"Karan");
16.    Student4 s2 = new Student4(222,"Aryan");
17.    //calling method to display the values of object
18.    s1.display();
19.    s2.display();
20.   }
21.}
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
1.  //Java program to overload constructors
2.  class Student5{
3.      int id;
```

```
4.  String name;
5.  int age;
6.  //creating two arg constructor
7.  Student5(int i,String n){
8.  id = i;
9.  name = n;
10. }
11. //creating three arg constructor
12. Student5(int i,String n,int a){
13. id = i;
14. name = n;
15. age=a;
16. }
17. void display(){System.out.println(id+" "+name+" "+age);}
18.
19. public static void main(String args[]){
20. Student5 s1 = new Student5(111,"Karan");
21. Student5 s2 = new Student5(222,"Aryan",25);
22. s1.display();
23. s2.display();
24. }
25.}
```

Output:

```
111 Karan 0
222 Aryan 25
```

What is the difference between constructor and method in Java?

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor’s name must be same as the class name.	The method name may or may not be same as the class name.

Copy Constructor in java

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
- A copy constructor is used for creating objects by the use of another object of the same class in java.
- The copy constructor returns a duplicate copy of the existing object of the class.
- It is used only for the initialization and is not applicable when the assignment operator is used instead.

There are many ways to copy the values of one object into another in Java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using Java constructor.

```
1. //Java program to initialize the values from one object to another object.
2. class Student6{
3.     int id;
4.     String name;
5.     //constructor to initialize integer and string
6.     Student6(int i, String n){
7.     id = i;
8.     name = n;  }
```

```
9.  //constructor to initialize another object
10. Student6(Student6 s){
11. id = s.id;
12. name =s.name;
13. }
14. void display(){System.out.println(id+" "+name);}
15. public static void main(String args[]){
16. Student6 s1 = new Student6(111,"Karan");
17. Student6 s2 = new Student6(s1);
18. s1.display();
19. s2.display();
20. }  }
```

Output:

```
111 Karan
111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
1. class Student7{
2.     int id;
3.     String name;
4.     Student7(int i,String n){
5.     id = i;
6.     name = n;
7.     }
8.     Student7(){}
9.     void display(){System.out.println(id+" "+name);}
10.    public static void main(String args[]){
11.    Student7 s1 = new Student7(111,"Karan");
12.    Student7 s2 = new Student7();
13.    s2.id=s1.id;
14.    s2.name=s1.name;
15.    s1.display();
16.    s2.display();
17.   }
18.}
```

Output:

```
111 Karan
111 Karan
```

Does constructor return any value?

Yes, it is the current class instance (You cannot use return type yet it returns a value).

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Is there Constructor class in Java?

Yes.

What is the purpose of Constructor class?

Java provides a Constructor class which can be **used to get the internal information** of a constructor in the class. It is found in the java.lang.reflect package.

What is the difference between constructor and destructor?

Constructor	Destructor
A constructor is used to initialize an instance of a class	A destructor is used to delete or destroy the objects when they are no longer in use
Constructors are called when an instance of a class is created	Destructors are called when an object is destroyed or released
Memory allocation	Releases the memory
Overloading is possible	Overloading is not allowed
They are allowed to have arguments	No arguments can be passed in a destructor