Unit II

Objects and Methods

Tutor: Pema Galey

Learning Outcomes

In this session, you will learn about:

- *Java Class and Objects*
- *Methods*
- *Method Overloading*
- *Variable Length Arguments*
- *Constructor*
- **String** operations and methods
- Access Modifiers → Public, Private, Protected, Default (package)
- **this** and **final** keywords
- **static** keyword
- Recursion method
- Operator: **instanceof**

Additional Topics

- Call-by-Value
- Call-by-reference
- Command-line Arguments
- Math class - methods

Introduction to OOP

- Java is an object-oriented programming language. The core concept of the object-oriented approach is to break complex problems into smaller objects.

- An ***object*** is any entity that has a state and behavior. For example, a bicycle is an object. It has
  - **States**: idle, first gear, etc.
  - **Behaviors**: braking, accelerating, etc.

Java Class

- A **class** is a blueprint for the object. Before we create an object, we first need to define the class.
- We can think of the class as a sketch (*prototype*) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.
- Since many houses can be made from the same description, we can create many objects from a class.

Java Class

```
class ClassName {
    // fields
    // methods
}
```

- Here, *fields* (variables) and *methods* represent the state and behavior of the object respectively. Also, known as class components.
  - **Fields (data members or attributes)** are used to store data
  - **Methods** are used to perform some operations

Example: Class Components

```
class Bicycle {
        // state or field
        private int gear = 5;
        // behavior or method
        public void braking() {
                System.out.println("Working of Braking");
        }
}
```

Java Objects

- An object is called an instance of a class.
- For example, suppose Bicycle is a class then *MountainBicycle, SportsBicycle, TouringBicycle*, etc can be considered as objects of the class.

- **Creating an Object in Java:**

```
className object = new className();
// for Bicycle class
Bicycle sportsBicycle = new Bicycle();
Bicycle touringBicycle = new Bicycle();
```

Access Members of a Class

- We can use the name of objects along with the . (dot) operator to access members of a class.

- For example,

```
// create object
Bicycle sportsBicycle = new Bicycle();
// access field and method
sportsBicycle.gear;
sportsBicycle.braking();
```

- Any Questions?

Java Methods

- A method is a block of code that performs a specific task.

- Dividing a complex problem into smaller chunks makes your program easy to understand and reusable.

- In Java, there are two types of methods:
  - **User-defined Methods**: We can create our own method based on our requirements.
  - **Standard Library Methods**: These are built-in methods in Java that are available to use.

User-Defined Methods

Declaring a Java Method

- The syntax to declare a method is:

```
returnType methodName() {
        // method body
}
```

- **returnType** - It specifies what type of value a method returns. If the method does not return a value, its return type is void.
- **methodName** - It is an identifier
- **method body** - It includes the programming statements that are used to perform some tasks. The method body is enclosed inside the curly braces { }.

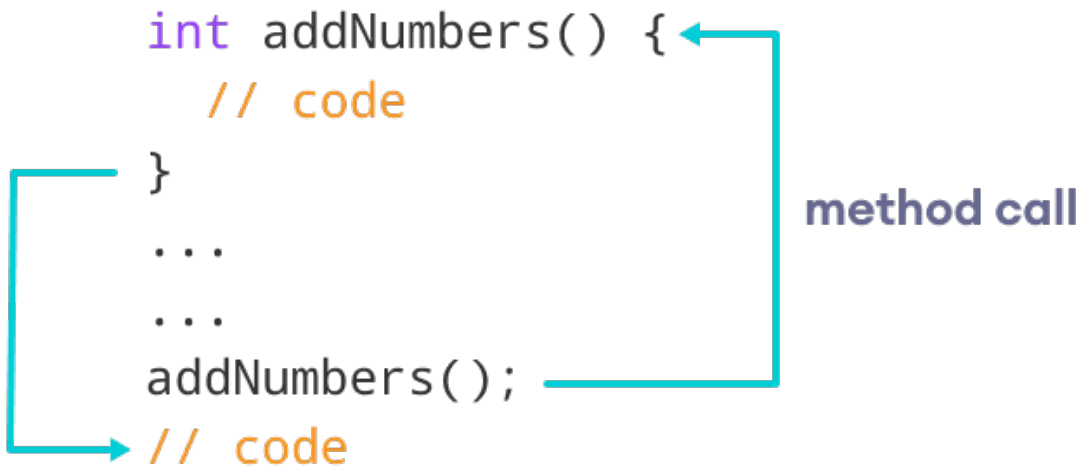
Declaring a Method

```
modifier static returnType nameOfMethod (parameter1, parameter2, ...) {
        // method body
}
```

- **modifier** - It defines access types whether the method is public, private, and so on.
- **static** - If we use the static keyword, it can be accessed without creating objects.
- **parameter1/parameter2** - These are values passed to a method. We can pass any number of arguments to a method.

Calling a Method in Java

- Calling a method
- Return type



Calling a Method using Single Class

```
class Main {
        // create a method
        public int addNumbers(int a, int b) {
                int sum = a + b;
                // return value
                return sum;
        }
        public static void main(String[] args) {
                int num1 = 25; int num2 = 15;
                // create an object of Main
                Main obj = new Main();
                // calling method
                int result = obj.addNumbers(num1, num2);
                System.out.println("Sum is: " + result);
        }
}
```

Output:
Sum is: 40

Calling a Method using more than one Class

```
class Main {
        // create a method
        public int addNumbers(int a, int b) {
                int sum = a + b;
                // return value
                return sum;
        }
}
class MainClass { // filename must be one containing the main method
        public static void main(String[] args) {
                int num1 = 25; int num2 = 15;
                // create an object of Main
                Main obj = new Main();
                // calling method
                int result = obj.addNumbers(num1, num2);
                System.out.println("Sum is: " + result);
        }
}
```

Output:
Sum is: 40

Standard Library Methods

- The standard library methods are built-in methods in Java that are readily available for use.
- These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.
- For example,
  - *print()* is a method of *java.io.PrintSteam*. The *print("...")* method prints the string inside quotation marks.
  - *sqrt()* is a method of *Math* class. It returns the square root of a number.

- Any Questions?

Method Overloading

- In Java, two or more methods may have the same name if they differ in signatures:
  - ✓Different number of parameters,
  - ✓Different types of parameters, or
  - ✓Different order of parameters.

- These methods are called overloaded methods and this feature is called method overloading.

Method Overloading

- **For example**:

  void func() { ... }
  void func(int a) { ... }
  float func(double a) { ... }
  float func(int a, float b) { ... }

- **Note**: The return types of the above methods are not the same. It is because method overloading is not associated with return types. Overloaded methods may have the same or different return types, but they must differ in parameters.

How to perform method overloading in Java?

- Here are different ways to perform method overloading:
  1. Overloading by changing the number of arguments
  2. By changing the data type of parameters

Method Overloading

```
class Main {
    String language;
    int lenght;
    // Method with no parameter
    public void getName() { this.language = "Java"; }
    // Method with a single parameter
    public void getName(String language) { this.language = language; }
    public void getName(int a) {
        length = a;
        System.out.println("Programming Langauage: " + this.language);
        System.out.println("Programming Lenght: " +length);
    }
    public static void main(String[] args) {
        // call constructor with no parameter
        Main obj = new Main();
        obj.getName();
        obj.getName("Java");
        obj.getName(20);
    } }
```

Java Constructor

What is a Constructor?

- A constructor in Java is similar to a method that is invoked when an object of the class is created.
- It is a special Method.
- Unlike Java methods, a constructor has the same name as that of the class and does not have any return type.
- For example,

```
class Test {
        Test() {
                // constructor body
        }
}
```

Example: Constructor

```
class Main {
        private String name;
        // constructor
        Main() {
                System.out.println("Constructor Called:");
                name = "Programiz";
        }
        public static void main(String[] args) {
                // constructor is invoked while
                // creating an object of the Main class
                Main obj = new Main();
                System.out.println("The name is " + obj.name);
        }
}
```

Output:

```
Constructor Called:
The name is Programiz
```

Type of Constructor

- In Java, constructors can be divided into 3 types:
  1. No-Argument Constructor
  2. Parameterized Constructor
  3. Default Constructor

1. No-Argument Constructor

- Similar to methods, a Java constructor may or may not have any parameters (arguments).
- If a constructor does not accept any parameters, it is known as a no-argument constructor.
- For example,

```
private Constructor() {
        // body of the constructor
}
```

2. Parameterized Constructor

- A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

```
class Main {
        String languages;
        // constructor accepting single value
        Main(String lang) {
                languages = lang;
                System.out.println(languages + " Programming Language");
        }
        public static void main(String[] args) {
                 // call constructor by passing a single value
                Main obj1 = new Main("Java");
                Main obj2 = new Main("Python");
                Main obj3 = new Main("C");
} }
```

3. Default Constructor

- If we do not create any constructor, the Java compiler automatically create a no-argument constructor during the execution of the program.
- This constructor is called default constructor.
- The default constructor initializes any uninitialized instance variables with default values.

Constructor Overloading

- Similar to method overloading, we can also create two or more constructors with different parameters. This is called constructors overloading.

```
class Main {
    String language;
    // constructor with no parameter
    Main() { this.language = "Java"; }
    // constructor with a single parameter
    Main(String language) { this.language = language; }
    public void getName() {
        System.out.println("Programming Langauage: " + this.language);
    }
    public static void main(String[] args) {
        // call constructor with no parameter
        Main obj1 = new Main();
        // call constructor with a single parameter
        Main obj2 = new Main("Python");
        obj1.getName();
        obj2.getName();
    } }
```

- Any Questions?

Java String

- **Java String Operations**
  1. Get Length of a String
     - To find the length of a string, we use the length() method of the String
  2. Join two Strings
     - We can join two strings in Java using the concat()
     - *String joinedString = first.concat(second);*
  3. Compare two Strings
     - we can make comparisons between two strings using the equals() method

String Methods

Methods	Description
substring()	returns the substring of the string
replace()	replaces the specified old character with the specified new character
charAt()	returns the character present in the specified location
getBytes()	converts the string to an array of bytes
indexOf()	returns the position of the specified character in the string
compareTo()	compares two strings in the dictionary order
trim()	removes any leading and trailing whitespaces
format()	returns a formatted string
split()	breaks the string into an array of strings
toLowerCase()	converts the string to lowercase
toUpperCase()	converts the string to uppercase
valueOf()	returns the string representation of the specified argument
toCharArray()	converts the string to a char array

Access Modifier

- In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods.
- For example,

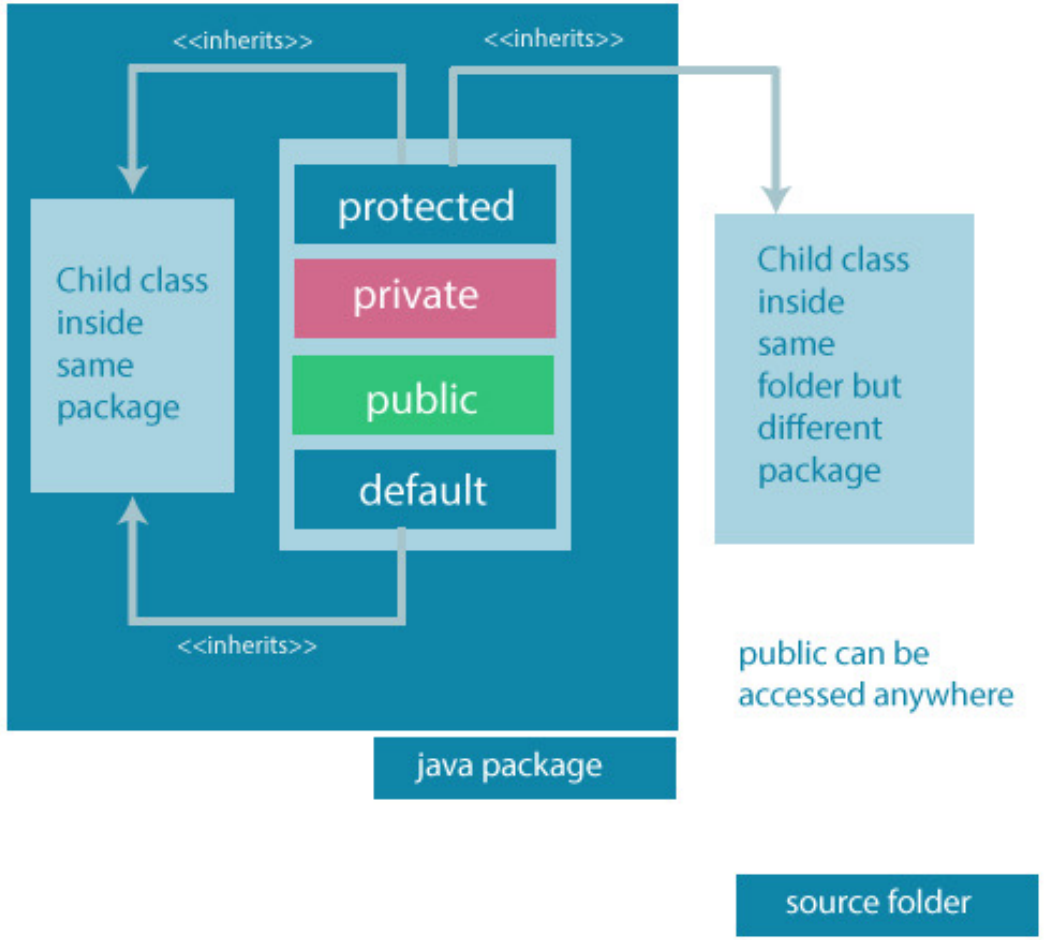
```
class Animal {
        public void method1() {...}
        private void method2() {...}
}
```

Types of Access Modifier

- There are four access modifiers keywords in Java and they are:

Modifier	Description
Default	declarations are visible only within the package (package private)
Private	declarations are visible within the class only
Protected	declarations are visible within the package or all subclasses
Public	declarations are visible everywhere

Access Modifiers



this Keyword

- In Java, this keyword is used to refer to the current object inside a method or a constructor.
- For example,

```
class Main {
        int instVar;
        Main(int instVar){
                this.instVar = instVar;
                System.out.println("this reference = " + this);
        }
        public static void main(String[] args) {
                Main obj = new Main(8);
                System.out.println("object reference = " + obj);
} }
```

Using this in Constructor Overloading

- While working with constructor overloading, we might have to invoke one constructor from another constructor. In such a case, we cannot call the constructor explicitly. Instead, we have to use this keyword.

```
class Complex {
        private int a, b;
        // constructor with 2 parameters
        private Complex( int i, int j ){
                this.a = i; this.b = j;
        }
        // constructor with single parameter
        private Complex(int i){
                // invokes the constructor with 2 parameters
                this(i, i);
        }
        public static void main( String[] args ) {
                Complex c1 = new Complex(3);
} }
```

final Keyword

- In Java, the final keyword is used to denote constants. It can be used with variables, methods, and classes.

- Once any entity (variable, method or class) is declared final, it can be assigned only once. That is,
  1. the **final variable** cannot be reinitialized with another value
  2. the **final method** cannot be overridden
  3. the **final class** cannot be extended

final Varialbe

- In Java, we cannot change the value of a final variable. For example,

```
class Main {
        public static void main(String[] args) {
                // create a final variable
                final int AGE = 32;
                // try to change the final variable
                AGE = 45;
                System.out.println("Age: " + AGE);
        }
}
```

static Keyword

- The **keyword static** indicates that the particular member belongs to a type itself, rather than to an instance of that type.
- The **static keyword** is mainly used for memory management. It can be used with variables, methods, blocks and nested classes. It is a keyword which is used to share the same variable or method of a given class.
- Basically, static is used for a constant variable or a method that is same for every instance of a class.

static Keyword

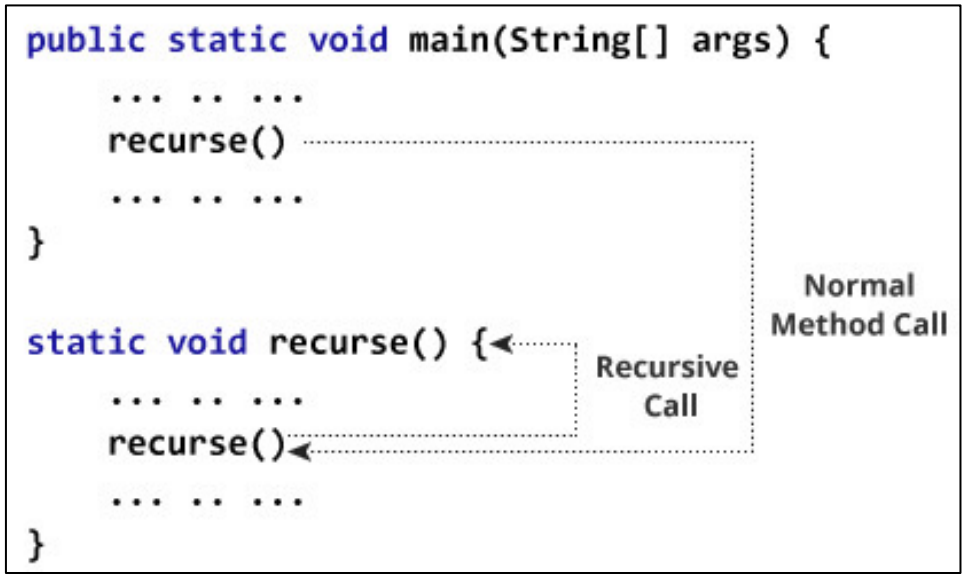
- In Java programming language, static keyword is a non-access modifier and can be used for the following:
  1. Static ***Block***
  2. Static ***Variable***
  3. Static ***Method***
  4. Static ***Classes***

- Example of static variable:

```
class Factorial {
        static int n;
        public static void main(String[] args) {
                n = 4;
                System.out.println(" N = " + n);
        }
}
```

Recursion Method

- In Java, a method that calls itself is known as a recursive method. And, this process is known as recursion.
- A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.



Example: Recursion

```
// Factorial of a number using Recursion
class Factorial {
        static int factorial( int n ) {
                if (n != 0) // termination condition
                        return n * factorial(n-1); // recursive call
                else
                        return 1;
        }
        public static void main(String[] args) {
                int number = 4, result;
                result = factorial(number);
                System.out.println(number + " factorial = " + result);
        }
}
```

Output:
4 factorial = 24

instanceof Operator

- The instanceof operator in Java is used to check whether an object is an instance of a particular class or not.

- Its syntax is

  objectName instanceOf className;

- Here, if objectName is an instance of className, the operator returns true. Otherwise, it returns false.

Example: instanceof

```
class Main {
        public static void main(String[] args) {
                // create a variable of string type
                String name = "Programiz";
                // checks if name is instance of String
                boolean result1 = name instanceof String;
                System.out.println("name is an instance of String: " + result1);
                // create an object of Main
                Main obj = new Main();
                // checks if obj is an instance of Main
                boolean result2 = obj instanceof Main;
                System.out.println("obj is an instance of Main: " + result2);
        }
}
```

Output:
name is an instance of String: true
obj is an instance of Main: true

Additional Topics

- Call-by-Value
- Call-by-reference
- Command-line Arguments
- Math class - methods

Call-by-Value

- Call by Value means calling a method with a parameter as value. Through this, the argument value is passed to the parameter.
- The modification done to the parameter passed does not reflect in the caller's scope

Call-by-Value

```
class Test{
     void meth(int i, int j){ // i=15, j=20 i.e. i and j are formal
                              //parameters
        i*=2; //i=i*2=30
        j/=2; //j=j/2=10
     }
}
class CallByValue{
     public static void main(String args[]){ //entry point
        Test ob=new Test();
        int a=15,b=20;
        System.out.println("a and b before call: "+a+" & "+b);
        ob.meth(a,b); // a & b are actual parameters
        System.out.println("a and b after call: "+a+" & "+b);
     }
}
```

Call-by-Reference

- Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.
- the modification done to the parameter passed are persistent and changes are reflected in the caller's scope.

Call-By-Reference

```
//call-by-reference
class Test{
    int a,b;
    Test(int i, int j){
        a=i;
        b=j;
    }
    void meth(Test t){//t=ob
        t.a*=2;//t.a=15*2=30
        t.b/=2;//t.b=20/2=10
    }
}
class CallByReference{
    public static void main(String args[]){
        Test ob=new Test(15,20);//ob.a=15, ob.b=20
        System.out.println("ob.a and ob.b before call: "+ob.a+" & "+ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: "+ob.a+" & "+ob.b);
    }
}
```

Math Class

- The Java Math class has many methods that allows you to perform mathematical tasks on numbers.
- Some basic methods are:
  - abs()
  - sqrt()
  - pow()
  - min()
  - max()
  - ceil()
  - floor()
  - round()
  - random()
  - etc..

Summary

Topics that we have learned:

- Java Class and Objects
- Methods
- Method Overloading
- Constructor
- String operations and methods
- Access Modifiers
- **this** and **final** keywords
- Recursion method
- Operator: **instanceof**

Thank you!