

Object Oriented Programming

Prepared by Dr. Yasser Abdelhamid

IT223
Object Oriented Programming

Lecture 04



OUTLINE

- Java program structure
- Packages
- Constructors
- Method declaration
- Access Modifiers
- Non-Access Modifiers

JAVA PROGRAM STRUCTURE

Documentation Section	Suggested
Package Statement	Optional
Import Statement	Optional
Interface Statement	Optional
Class Definitions	Optional
Main Method Class { Main Method Define }	Essential

JAVA PACKAGES

- ❖ In Java, a package is a way to organize and group related classes, interfaces, and sub-packages.
- ❖ It provides a mechanism for creating a namespace and helps in avoiding naming conflicts between classes or interfaces with the same name.
- ❖ A package is represented by a directory structure in the file system.
- ❖ The package name is used to create a hierarchical structure of directories.

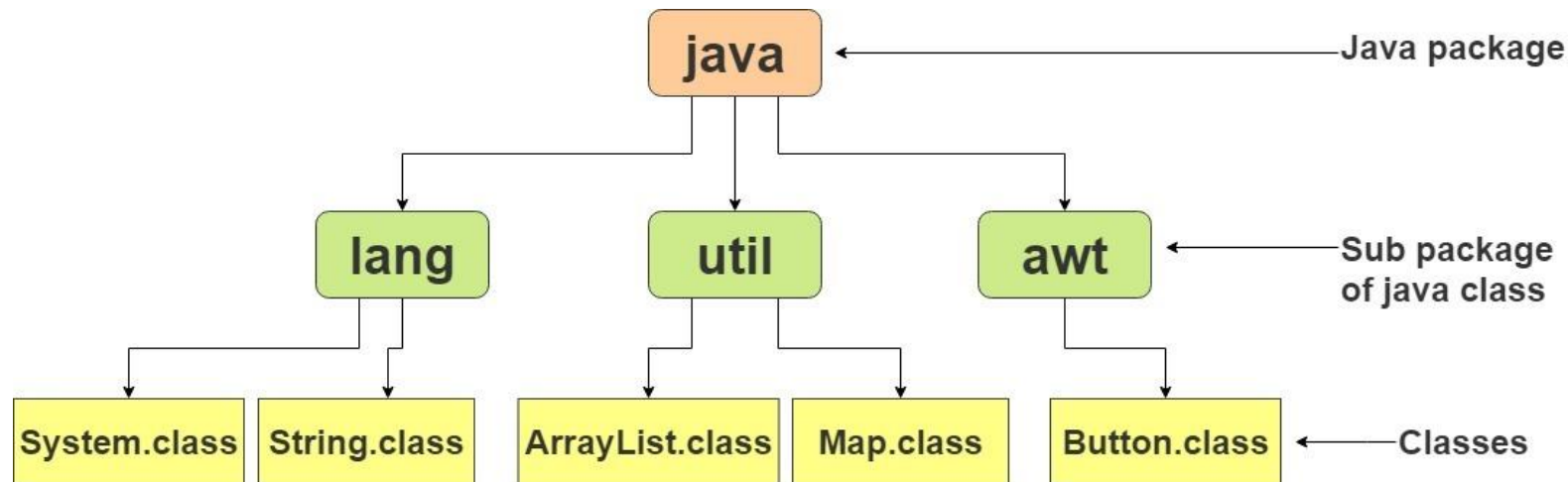
HOW TO CREATE A PACKAGE?

- ❖ To create a package in Java, you include a package declaration at the top of each Java source file that belongs to the package.
- ❖ For example, to declare a class named "MyClass" inside a package named "com.example.package", the package declaration would be:

```
package com.example.package;  
  
public class MyClass {  
    // Class members and methods  
}
```

JAVA API

- ❖ The Java API is a library of prewritten classes
- ❖ The library contains a large number of components
- ❖ To use a class or a whole package from the library, you need to use the **import** keyword



USING JAVA PACKAGES

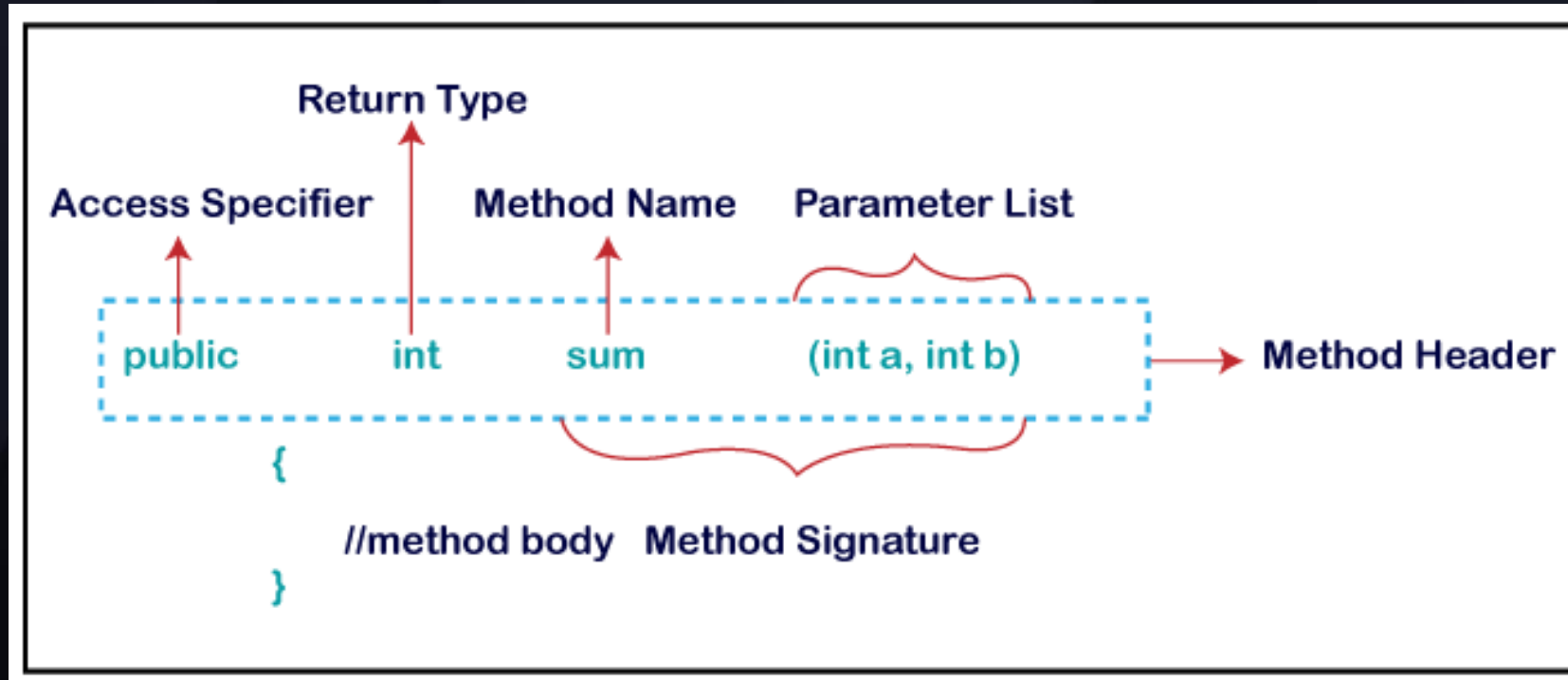
```
import java.util.Scanner;  
// import java.util.*; to import the whole package  
class MyClass {  
    public static void main(String[] args) {  
        Scanner myObj = new Scanner(System.in);  
        System.out.println("Enter username");  
  
        String userName = myObj.nextLine();  
        System.out.println("Username is: " + userName);  
    }  
}
```

USER-DEFINED PACKAGES

```
package mypack;  
class MyPackageClass {  
    public static void main(String[] args) {  
        System.out.println("This is my package!");  
    }  
}
```

```
javac -d . MyPackageClass.java  
java mypack.MyPackageClass
```

METHOD DECLARATION



METHOD SIGNATURE

In Java, a method signature refers to the unique combination of a method's name and its parameter types. It provides a concise summary of the method's characteristics, allowing the compiler and developers to differentiate between different methods with the same name but different parameters.

METHOD SIGNATURE

The method signature consists of the following elements:

- **Method Name:** It is the name used to identify the method within its class or interface.
- **Parameter Types:** It specifies the number, order, and data types of the method's parameters. The parameter types help distinguish overloaded methods, which are multiple methods with the same name but different parameter lists.

Methods cannot have the same name and the same parameter types but different return types.

EXAMPLE:

```
public class MyClass {  
    public void printMessage(String message) {  
        // Method implementation  
    }  
  
    public void printMessage(int number) {  
        // Method implementation  
    }  
  
    public int calculateSum(int a, int b) {  
        // Method implementation  
        return a + b;  
    }  
}
```

CONSTRUCTORS

- ❖ In Java, a constructor is a **special method that is used to initialize objects** of a class.
- ❖ It is called when an instance (object) of the class is created using the **new** keyword.
- ❖ The constructor **has the same name as the class** and **does not have a return type**, not even void.

CONSTRUCTORS

- ❖ Constructors are **primarily used to set initial values to the instance variables** of an object or perform any necessary setup tasks when creating an object.
- ❖ They **can take parameters to initialize the object** with specific values.
- ❖ Constructors can be **overloaded**, which means **having multiple constructors with different parameter lists**. This allows for object creation with different initialization options.

EXAMPLE

```
public class MyClass {  
    private int value;  
  
    // Constructor without parameters  
    public MyClass() {  
        value = 0; // Initialize the value to 0  
    }  
  
    // Constructor with a parameter  
    public MyClass(int initialValue) {  
        value = initialValue; // Initialize the value to the provided initial value  
    }  
  
    // Other methods and members of the class  
    // ...  
}
```

EXAMPLE

```
public class Class1{  
    int x;  
    public Class1(int y) {  
        x = y;  
    }  
  
    public static void main(String[] args) {  
        Class1 myObj = new Class1(5);  
        System.out.println(myObj.x);  
    }  
}
```

DEFAULT CONSTRUCTOR

❖ If you do not create a class constructor yourself, Java creates a **default constructor** that does not make any initializations.

```
MyClass obj1 = new MyClass();           // Calls the default constructor  
MyClass obj2 = new MyClass(10);         // Calls the constructor with int parameter
```

MODIFIERS

- ❖ In Java, modifiers are keywords that are used to specify the scope, accessibility, and behavior of classes, methods, variables, and other program elements.
- ❖ Modifiers alter the default behavior of program elements and provide additional control over their visibility, access, and usage.

MODIFIERS

❖ Modifiers are classified into:

- **Access Modifiers** - control the access level.
- **Non-Access Modifiers** - do not control access level, but provides other functionality.

CLASS ACCESS MODIFIERS

- ❖ In Java, class access modifiers are used to control the visibility and accessibility of classes.
- ❖ They determine which other classes can access and interact with a particular class.

PUBLIC CLASS MODIFIER

- ❖ A public class can be accessed from any other class or package.
- ❖ It has the widest visibility and is typically used for classes that form the API (Application Programming Interface) of a library or module.

```
public class PublicClass {  
    // Class members and methods  
}
```

EXAMPLE: PUBLIC CLASS

```
public class MyPublicClass{  
    public static void main(String[] args) {  
        System.out.println("I am accessible to any other class");  
    }  
}
```

DEFAULT (PACKAGE-PRIVATE) CLASS

- ❖ If no access modifier is specified, it is considered the default (also known as package-private) access level.
- ❖ A class with default access can be accessed **only within the same package**.

```
class DefaultClass {  
    // Class members and methods  
}
```

EXAMPLE: DEFAULT CLASS

```
class MyDefaultClass{  
    public static void main(String[] args) {  
        System.out.println("I am accessible only to classes in the same package");  
    }  
}
```

ATTRIBUTES ACCESS MODIFIERS

- ❖ In Java, attribute access modifiers are used to control the visibility and accessibility of class attributes (also known as fields or variables).
- ❖ They determine how other classes can access and modify the attributes of a particular class.

PUBLIC ATTRIBUTES

- ❖ A public attribute can be accessed and modified from any other class or package.
- ❖ It has the widest visibility and is typically used for attributes that form the API (Application Programming Interface) of a class.

```
public class MyClass {  
    public int publicAttribute;  
    // Other class members and methods  
}
```

PROTECTED ATTRIBUTES

- ❖ A protected attribute can be accessed and modified within the same package or in subclasses (even if they are in different packages).

```
public class MyClass {  
    protected int protectedAttribute;  
    // Other class members and methods  
}
```

DEFAULT (PACKAGE-PRIVATE) ATTRIBUTES

- ❖ If no access modifier is specified, it is considered the default (also known as package-private) access level.
- ❖ An attribute with default access can be accessed and modified only within the same package.

```
public class MyClass {  
    int defaultAttribute;  
    // Other class members and methods  
}
```

PRIVATE ATTRIBUTES

- ❖ A private attribute is the most restricted access level.
- ❖ It can only be accessed and modified within the same class and is not visible to any other class, even within the same package.
- ❖ Private attributes are typically used to encapsulate data and enforce information hiding.

```
public class MyClass {  
    private int privateAttribute;  
    // Other class members and methods  
}
```

METHOD ACCESS MODIFIERS

- ❖ In Java, method access modifiers are used to control the visibility and accessibility of methods.
- ❖ They determine how other classes can access and invoke methods of a particular class.
- ❖ Java provides four access modifiers for methods:
 - public
 - protected
 - default
 - private

PUBLIC METHODS

- ❖ A public method can be accessed and invoked from any other class or package.
- ❖ It has the widest visibility and is typically used for methods that form the API (Application Programming Interface) of a class.

```
public class MyClass {  
    public void publicMethod() {  
        // Method implementation  
    }  
    // Other class members and methods  
}
```

PROTECTED METHODS

- ❖ A protected method can be accessed and invoked within the same package or in subclasses (even if they are in different packages).

```
public class MyClass {  
    protected void protectedMethod() {  
        // Method implementation  
    }  
    // Other class members and methods  
}
```

DEFAULT (PACKAGE-PRIVATE) METHODS

- ❖ If no access modifier is specified, it is considered the default (also known as package-private) access level.
- ❖ A method with default access can be accessed and invoked only within the same package.
- ❖ It is used when no access control restriction is required for a method.

```
public class MyClass {  
    void defaultMethod() {  
        // Method implementation  
    }  
    // Other class members and methods  
}
```

PRIVATE METHODS

- ❖ A private method is the most restricted access level.
- ❖ It can only be accessed and invoked within the same class and is not visible to any other class, even within the same package.
- ❖ Private methods are typically used for internal implementation details that are not meant to be exposed or accessed from outside the class.

```
public class MyClass {  
    private void privateMethod() {  
        // Method implementation  
    }  
    // Other class members and methods  
}
```

CONSTRUCTORS ACCESS MODIFIERS

- ❖ In Java, constructors can also have access modifiers to control their visibility and accessibility.
- ❖ The access modifiers for constructors work similarly to those for methods, and they determine how other classes can create and initialize objects of a particular class.
- ❖ Java provides three access modifiers for constructors:
 - public
 - default
 - private

PUBLIC CONSTRUCTOR

- ❖ A public constructor can be accessed and used to create objects from any other class or package.
- ❖ It has the widest visibility and is typically used for constructors that form the public API (Application Programming Interface) of a class.

```
public class MyClass {  
    public MyClass() {  
        // Constructor implementation  
    }  
    // Other class members and methods  
}
```

DEFAULT (PACKAGE-PRIVATE) CONSTRUCTOR

- ❖ If no access modifier is specified for a constructor, it is considered the default (also known as package-private) access level.
- ❖ A constructor with default access can be accessed and used to create objects only within the same package.
- ❖ It is used when no access control restriction is required for the constructor.

```
public class MyClass {  
    MyClass() {  
        // Constructor implementation  
    }  
    // Other class members and methods  
}
```

PRIVATE CONSTRUCTOR

- ❖ A private constructor is the most restricted access level.
- ❖ It can only be accessed and used to create objects within the same class and is not visible to any other class, even within the same package.
- ❖ Private constructors are typically used for classes that should not be instantiated directly.

```
public class MyClass {  
    private MyClass() {  
        // Constructor implementation  
    }  
    // Other class members and methods  
}
```

ATTRIBUTES, METHODS AND CONSTRUCTORS MODIFIERS

- ❖ **public** : The code is accessible for all classes
- ❖ **private** : The code is only accessible within the declared class
- ❖ **default** : The code is only accessible in the same package. This is used when you don't specify a modifier.
- ❖ **protected** : The code is accessible in the same package and subclasses.

NON-ACCESS CLASS MODIFIERS

❖final:

- The class cannot be inherited by other classes

❖abstract

- The class cannot be used to create objects.
- To access an abstract class, it must be inherited from another class.

NON-ACCESS ATTRIBUTES AND METHODS MODIFIERS

❖ final

- Attributes and methods cannot be overridden/modified.

❖ static

- Attributes and methods belong to the class, rather than an object

❖ abstract

- Can only be used in an abstract class

EXAMPLE: STATIC AND PUBLIC METHODS

- ❖ **static** method, can be accessed without creating an object of the class.
- ❖ **public** methods, can only be accessed through creating objects.

```
public class Class01 {  
    static void myStaticMethod() {  
        System.out.println("I am a static method");  
    }  
  
    public void myPublicMethod() {  
        System.out.println("I am a public method");  
    }  
    public static void main(String[] args) {  
        Class01.myStaticMethod();  
        Class01 obj1 = new Class01();  
        obj1.myPublicMethod();  
    }  
}
```

EXERCISE

- ❖ Create a class named “Employee”, that has the attributes: name, address, department, id , phone number, and basic salary.
- ❖ Create the following methods:
 - netSal : to calculate net salary which is 80% of the basic salary.
 - dispEmp: to display all information of the employee on console.
- ❖ Create the class “TestClass “ to test the Employee class.

SUMMARY

- Java program structure
- Packages
- Constructors
- Method declaration
- Access Modifiers
- Non-Access Modifiers



Thank You

www.midocean.ae

