

## **Session 2- Facilitation Guide**

### **The Structure of Java Programs**

#### **Index**

- I. Identifiers**
- II. Legal Identifiers / Keywords**
- III. Java Code Conventions**
- IV. Declare Classes**
- V. Declare Class Members**
- VI. Data Types**
- VII. Variable Declarations**
- VIII. Declaring Enums**
- IX. Casting Primitives**
- X. Introduction to Packages**
- XI. Access Modifiers**

#### **For (2 hrs) ILT**

In our last introductory session we learned:

- The history of Java Programming language.
  - We learned that Java has evolved into one of the most popular programming languages, known for its portability, security, performance, and extensive libraries. ● Its versatility allows it to be used in a wide range of applications, from web development and enterprise software to mobile apps and embedded systems. ● Java continues to be a critical player in the software development landscape, supported by Oracle and a vibrant community of developers worldwide. ● We also learned how to install Java Development Kit on your computer and how to run your first Java program.
- We understood the advantage of object-oriented programming languages like Java over other procedural programming languages.

In this session we will introduce identifiers and keywords of Java and will teach you how to define a class and add class members. Java Class consists of Class Declaration, Class Members, Constructors and Methods. It also contains optional package declaration and import statements. We will discuss each of them in the upcoming sessions. In this class we will mainly discuss Java identifiers, data types, variables and access modifiers. We will also explain package declarations and naming conventions. Now let us understand some of the common terms used in Java programming.

## I. Identifiers

Identifiers in Java are names given to variables, classes, methods, and other programming elements in a Java program. An identifier can be composed of letters, **digits**, **underscores**, and **dollar signs**. However, the first character of an identifier must be a **letter**, **dollar sign(\$)** or an **underscore ('\_')**, and it cannot be a reserved keyword.

## II. Legal Identifiers / Keywords

In Java, a reserved keyword is a word that has been set aside by the Java programming language for a specific purpose. These keywords have predefined meanings and cannot be used as identifiers (variable names, class names, method names, etc.) within the Java code. Attempting to use a reserved keyword as an identifier will result in a compilation error. These are some of the reserved keywords in java.

**abstract, byte, double, default, final, float, import, static, switch, throw, try, volatile.**

(For complete list please refer to the page :

[https://docs.oracle.com/javase/tutorial/java/nutsandbolts/\\_keywords.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html))

Example of valid identifiers: age, Employee, weight, finalValue, default\_obj, update67, \$salary.

Example of invalid identifiers: 1st\_value, final, default, 2d, ~parameter, 67\_update.

## III. Java Code Conventions

Java coding conventions are a set of guidelines and best practices that developers follow to write code in a consistent and readable manner. These conventions are not enforced by the Java compiler, but they are widely adopted in the Java community to enhance code readability, maintainability, and collaboration among team members. Various organizations also follow their own coding standards in addition to standard coding conventions and they typically document it to ensure a uniform coding style across all of their projects.

As a part of naming convention, class names should start with uppercase. On the other hand, package and variable names should start with lowercase. However, constants and enum names should be in all UPPER\_CASE letters.

Here is the general rule:

- For class – CamelCase with the first letter capitalized.
- For methods/variables –camelCase with the first letter lowercase.
- For constants and enum – all UPPER\_CASE letters.

**Note:** CamelCase is a way to separate the words in a phrase by making the first letter of each word capitalized and not using spaces. Examples of camelCase are : MicrosoftWord, FedEx, mobileUser.

As such, there will be no error in compilation if we do not follow the coding convention. But strictly following coding guidelines will make our code more readable and introduce less bugs during development.

Programmers should also ensure that their source codes are indented properly. Integrated Development Environment (IDE) such as Eclipse or IntelliJ are useful in auto indenting and code reformatting.

**Note:** An integrated development environment is a software application that helps programmers develop software code easily and efficiently.

## Comment in Java Code

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

In Java, there are two types of comments that you can use to add explanatory or descriptive text within your code: single-line comments and multi-line comments.

### • Single-line comments:

Single-line comments start with `//` and continue until the end of the line. They are used for adding comments on a single line.

Example:

```
...  
int age = 25; // This is a single-line comment  
...
```

### • Multi-line comments:

Multi-line comments, also known as block comments, are used for adding comments that span multiple lines. They start with `/*` and end with `*/`.

Example:

Java

```
...
/* This is a multi-line comment
   It can span multiple lines
   This is useful for longer comments or explanations */
int salary = 50000;
...
```

It's worth noting that multi-line comments cannot be nested within each other. If you try to add a multi-line comment within another multi-line comment, it will result in a compilation error.

Comments in Java are ignored by the compiler, so they have no impact on the execution of the code. They serve as a helpful tool for documenting your code, making it more understandable and maintainable.

**Note:** Adding comments to your code is considered a good practice and can be especially helpful for yourself and other developers who may read and work with your code in the future.

#### IV. Declare Classes

In Java, every source file can start with a package declaration or an import statement, followed by a class or interface definition. If the package name is not declared in the beginning of the source file then it will be considered as the default package.

**Note:** Use of the default package is allowed but it is not encouraged as it is considered as bad programming practice.

Here are the rules for declaring a Java source file:

**Class or interface definition:** Every Java source file must have at least one class or interface definition. The syntax for a class or interface definition is as follows:

**Note:** We will discuss more about interfaces in the upcoming sessions.

For the time, just be familiar with the structure of the class. We will discuss all the terms listed here in our upcoming training session. So here is an example of how a java class

looks like.

Java

```
class <class_name> [extends super_class_name] [implements interface_name [,  
interface_name...]] {  
    // class members  
}
```

## V. Declare Class Members

In a Java program, class members refer to the fields (variables), methods, constructors, nested classes or interfaces, and static blocks.

**Note:** We will define all the terms mentioned here in the upcoming sessions.

Each member must be declared with an appropriate access modifier and should follow the syntax rules for the particular type of member.

Class members are used to define the structure and functionality of the class.

Here are the three main types of class members in Java:

### Fields (Variables):

Fields, also known as variables or class variables, represent the data associated with a class. They store the state or characteristics of objects created from the class. Fields can be of various data types such as integers, strings, booleans, and custom types. They can have different access modifiers (e.g., public, private, protected, or default) to control their visibility and accessibility.

Example:

Java

```
public class Vehicle {  
    private int id;  
    public String type;  
    protected double price;  
  
}
```

Here, Vehicle is a class that has characteristics such as id, type and price.

### Methods:

Methods define the behavior or actions that objects of a class can perform. They encapsulate the algorithms and operations associated with the class. Methods can have parameters (inputs) and a return type (output), which determines if and what value is returned from the method. Like fields, methods can have different access modifiers.

Example:

```
Java
public class Vehicle {
    public void accelerate() { // public method
        // method body
    }
}
```

Here, Vehicle is a class whose objects will have a behavior/action of acceleration. This is indicated by the method named accelerate().

### Constructors:

Constructors are special methods used to create and initialize objects of a class. It will be discussed in upcoming sessions.

## VI. Data Types

In Java, data types represent the kind of values that variables or expressions can hold. Java has two main categories of data types: primitive data types and reference data types.

### Primitive Data Types:

Java has eight primitive data types, which are used to store simple values. They are:

Data type	Description Range
<b>byte</b>	Represents a signed 8-bit integer -128 to 127
<b>short</b>	Represents a signed 16-bit integer -32,768 to 32,767

<b>int</b>	Represents a signed 32-bit integer -2,147,483,648 to 2,147,483,647
<b>long</b>	Represents a signed 64-bit integer -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<b>float</b>	Represents a single-precision 32-bit floating-point number
<b>double</b>	Represents a double-precision 64-bit floating-point number
<b>boolean</b>	Represents a boolean value true or false

**char** Represents a single Unicode character '\u0000' (0) to '\uffff' (65,535)

### Reference Data Types:

Reference data types refer to objects, which are instances of classes or arrays. Unlike primitive data types, reference data types store references to objects rather than the actual values.

**Note:** Understanding and appropriately using data types is essential in Java programming as they define the range of values that variables can hold and the operations that can be performed on them.

## VII. Variable Declarations

In Java, variables are declared by specifying the data type followed by the variable name. This informs the compiler about the type of data the variable will hold. Here's the general syntax for declaring variables in Java:

```
data_type variable_name;
```

Here are a few examples of variable declarations in Java:

```
Java
```

```
...
```

```
int age; // Declares an integer variable named 'age' double salary; //  
Declares a double variable named 'salary'  
boolean isStudent; // Declares a boolean variable named 'isStudent' char  
grade; // Declares a character variable named 'grade' String name; //  
Declares a String variable named 'name' ...
```

In the examples above, the data types include int (integer), double (floating-point number), boolean (boolean value), char (character), and String (sequence of characters).

Variables can also be assigned an initial value at the time of declaration using the assignment operator (=). Here are some examples:

```
Java  
...  
int age = 25; // Declares an integer variable named 'age' and assigns the  
value 25  
double salary = 50000.50; // Declares a double variable named 'salary' and  
assigns the value 50000.50  
boolean isStudent = true; // Declares a boolean variable named 'isStudent' and  
assigns the value true  
char grade = 'A'; // Declares a character variable named 'grade' and assigns  
the value 'A'  
String name = "John"; // Declares a String variable named 'name' and  
assigns the value "John"  
...
```

It's important to note that variables must be declared before they can be used. This means you need to declare a variable with its appropriate data type before you can assign a value to it or use it in any operations.

Additionally, variables in Java are case-sensitive, meaning that age, Age, and AGE are considered as different variable names.

**Note:** It's recommended to use meaningful names that describe the purpose of the variable to enhance code readability.

By declaring variables in Java, you can allocate memory for data storage and refer to the values they hold throughout your program.



### Example:

```
...
int age = 25;
double salary = 50000.50;
boolean isStudent = true;
char grade = 'A';
...
```

## VIII. Declaring Enums

An `enum` is a special "class" that represents a group of constants (unchangeable variables, like `final` variables).

To create an `enum`, use the `enum` keyword (instead of `class` or `interface`), and separate the constants with a comma. Note that they should be in uppercase letters:

Enum is used to define constants and it is a way to make the code more readable. Values are checked at compile time and thus avoids unexpected behavior due to invalid values being passed in.

Here's a simple example of an `enum` that will make the idea clear.

```
Java
public enum Color{
    GREEN,
    RED,
    BLUE;
}
```

The value `GREEN` can be accessed as `Color.GREEN`.

Java `enum` can be used as a class and it can contain methods, variables and constructors. Here is an example of java `enum` with methods.

```
Java
public enum Accounts {
    //Declaring the constants of the enum
}
```

```

CURRENT (10000.0),
SAVINGS (5000.0),
SALARY (2000.0),
NRI (20000.00);
//Instance variable of the enum
double minimumBalance;
//Constructor
Accounts(double balance) {
    this.minimumBalance=balance;
}
//Enum method, it can be static also
public static void enumMethod() {
    for (Accounts ac : Accounts.values())
        System.out.printf("Minimum Balance required for %s account
is %.2f %n",
                                ac, ac.minimumBalance);
}

public static void main(String[] args) {
    enumMethod();
}
}

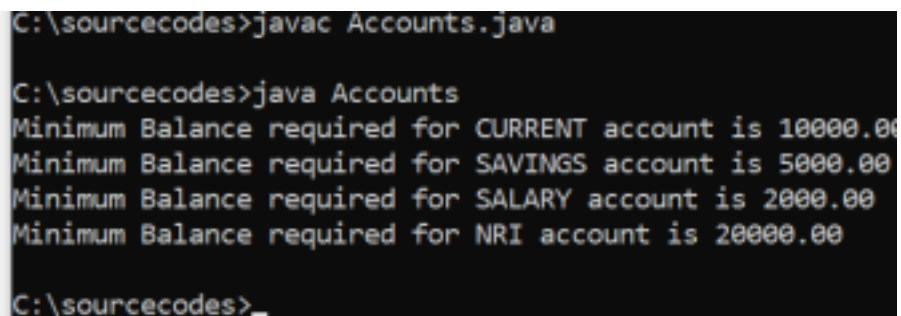
```

The output of the program will be :

```

Minimum Balance required for CURRENT account is 10000.00
Minimum Balance required for SAVINGS account is 5000.00
Minimum Balance required for SALARY account is 2000.00
Minimum Balance required for NRI account is 20000.00

```



```

C:\sourcecodes>javac Accounts.java

C:\sourcecodes>java Accounts
Minimum Balance required for CURRENT account is 10000.00
Minimum Balance required for SAVINGS account is 5000.00
Minimum Balance required for SALARY account is 2000.00
Minimum Balance required for NRI account is 20000.00

C:\sourcecodes>

```

Enum constants can be compared using == operator.

## IX. Casting Primitives

In Java, casting primitives refers to the process of converting one primitive data type to another. Since Java is a typed language, variables are assigned specific data types during their declaration, and the type of data a variable can hold is determined at compile-time.

Sometimes, it is necessary to convert a value of one primitive data type into another to perform certain operations or assignments. This is where casting comes into play. There are two types of casting for primitive data types in Java:

### **Widening (Implicit) Casting:**

Widening casting, also known as implicit casting, occurs when a value of a smaller data type is automatically converted to a value of a larger data type without the need for explicit type casting. Java performs this type of casting automatically because there is no risk of losing precision during the conversion. So if a data is converted in this order then it will be done automatically.

**byte -> short -> char -> int -> long -> float -> double**

For example, converting an int to a double:

```
...  
int numInt = 10;  
double numDouble = numInt; // Implicit casting (int to double)  
...
```

In the example above, the int value 10 is implicitly cast to the double value

### **10.0. Narrowing (Explicit) Casting:**

Narrowing casting, also known as explicit casting, is required when a value of a larger data type needs to be explicitly converted to a value of a smaller data type. This type of casting might lead to data loss or truncation because the target data type may not be able to represent all possible values of the source data type. This is not automatic.

For example, converting a double to an int:

```
double numDouble = 12.34;  
int numInt = (int) numDouble; // Explicit casting(double to int)
```

In the example above, the double value 12.34 is explicitly cast to the int value 12, and the fractional part is truncated. Here casting is not automatic and we will get a compilation error if casting is not done explicitly.

**Note:** It's important to be cautious when using narrowing casting to ensure that the converted value fits within the target data type's range, and no important data is lost during the conversion.

In general, Java automatically performs widening casting when converting between compatible data types, and explicit (narrowing) casting is required when converting between incompatible data types.

## **X. Introduction to Packages**

In Java, a package is a way to organize and structure classes and other Java elements into groups. It provides a means of managing and grouping related components together, helping to avoid naming conflicts and improve code organization.

A package is essentially a directory that holds the Java files (.java) that belong to it. It provides a namespace for the classes contained within it, allowing you to refer to them uniquely. By using packages, you can create a hierarchy of classes and organize them based on their functionality or purpose.

Java packages can broadly be divided into two categories - Built-in Packages and User-defined packages.

### **Java Built-in Packages:**

Java comes with a rich set of predefined packages that are included as part of the Java Development Kit (JDK). These packages provide essential functionalities and cover various aspects of programming, making it easier to perform common tasks. Examples of Java built-in packages include `java.lang`, `java.util`, `java.io`, `java.math`, `java.net`, `java.time`, etc.

The classes and interfaces in these built-in packages are automatically available for use in any Java program using **import** statements (will be explained next). However, the `java.lang` package is imported automatically and no explicit import statement is required.

### **User-Defined Packages:**

User-defined packages are packages created by Java developers to organize their own classes and interfaces. These packages are used to logically group related code together, making it easier to manage and maintain larger projects.

To create a user-defined package, developers use the package keyword followed by the package name at the top of each Java source file that belongs to that package. If the file belongs to a package, the first line of the file should be a package declaration. The syntax for a package declaration is as follows:

```
package <package_name>;
```

The package name should be a valid Java identifier and should correspond to the directory structure where the file is located.

User-defined packages often follow the reverse domain name convention (e.g., com.example.myproject) to avoid naming conflicts with packages created by other developers or libraries.

To use classes and interfaces from a user-defined package within another package, you need to explicitly import them using the import statement.

**Import statements:** After the package declaration (if any), import statements can be used to specify which classes or packages are being used in the file. The syntax for an import statement is as follows:

```
import <package_name.class_name>;
```

Multiple import statements can be used, separated by semicolons.

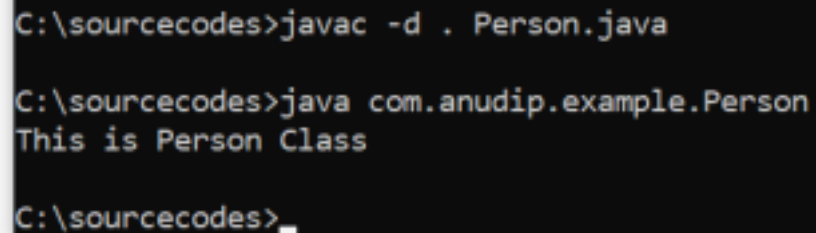
If we want to import all the classes and interfaces from a package then we use the wildcard (\*).

Here is an example of how typical Java class with package and import statements will look:

```
Java
package com.anudip.example;
import java.util.*;
public class Person {
    int age = 25;
    int salary = 50000;
    public static void main(String[] args) {
        System.out.println ("This is Person Class");
    }
}
```

The output of this program is

This is Person Class



```
C:\sourcecodes>javac -d . Person.java

C:\sourcecodes>java com.anudip.example.Person
This is Person Class

C:\sourcecodes>
```

**Note:** Could you guess why we compiled the above program with the “javac -d” option? Actually, if we use this option then the java compiler will create a directory structure according to the package name. The class file will be located at com/anudip/example/Person (Linux) or at com\anudip\example\Person (Windows). Then we can run the program with the fully qualified class name (along with package name).

## XI. Access Modifiers

In Java, access modifiers are used to define the visibility and accessibility of classes, variables, and methods in different parts of a program. There are four access modifiers available in Java:

**public:** Public members can be accessed from any part of the program. They have the highest level of accessibility.

**protected:** Protected members are accessible within the same package and by subclasses of the class in any package.

**private:** Private members are accessible only within the same class.

**Default (package-private):** Default members are accessible only within the same package.

The access modifiers can be used in the declaration of classes, variables, and methods. For example:

```
Java
public class ExampleClass {
    public int publicVariable;
```

```

protected int protectedVariable;
int defaultVariable;
private int privateVariable;
public void publicMethod() {
    // Code goes here
}
protected void protectedMethod() {
    // Code goes here
}
void defaultMethod() {
    // Code goes here
}
private void privateMethod() {
    // Code goes here
}
}

```

In this example, the class "ExampleClass" has four member variables and four member methods, each with a different access modifier.

The public members can be accessed from any part of the program, the protected members are accessible within the same package and by subclasses, the default members are accessible only within the same package, and the private members are accessible only within the same class.

Following table explains the variable accessibility based on the access modifiers.

Access  Modifier	Within the  same class	Within the  same package	Outside package  by subclass only	Outside  package
private	Y	N N		N

<b>Default</b>	Y	Y N	N
<b>protected</b>	Y	Y Y	N
<b>public</b>	Y	Y Y	Y

**Note:** Access modifiers help to control the level of access to different parts of a program and are important for encapsulation and data hiding in object-oriented programming. By using the appropriate access modifiers, we can ensure that our code is secure, maintainable, and easy to understand.

*After the 2 hr ILT, the student has to do a 1 hour live lab. Please refer to the Session 2 Lab doc in the LMS.*