College of Business Education

Java SE (Standard Edition) for Application Software Development

Dr. Ahmed Kijazi

Ahmed.kijazi@cbe.ac.tz

0712307240

1. The History of Java

Java is a computer language that James Gosling created in the early 1990s. It is known for its object-oriented design. The project was begun by a small team of sun microsystem engineers, **James Gosling (a researchers team leader), Mike Sheridan**, and **Patrick Naughton,** aiming to develop a language specifically designed for digital devices, including set-top boxes and televisions. They considered using C++ for the project; therefore, Gosling tried to modify and enhance the C++ programming language. However, the project was eventually abandoned due to several deficiencies in C++ programming, such as requiring more memory and being platform-dependent.

To overcome the challenge of C++ programming, in 1991, the team initiated a " **Green** " project to develop a "**Green talk"** programming language with the file extension (.gt). Gosling later updated the name "**Greentalk**" to "**OAK**" which is the name of the tree that remained outside his office (Fig. 1). The "**OAK**" tree can live over 1,000 years**.** The symbolism associated with oak trees has been held in high regard by various cultures and civilizations throughout history, signifying profound and influential meanings. The oak tree, characterized by its robust trunk and expansive branches, symbolizes strength, longevity, and endurance in many countries like the U.S.A., France, Germany, Romania etc.

Figure 1: OAK Tree

Since the name "**OAK**" it was already a trademark by **Oak Technologies**, the team brainstormed to find another name, and several were proposed, such as **JAVA, DNA, SILK, RUBY, etc. Finally,** "**OAK**" was renamed to Java In 1995, a type of delicious coffee from Java Island in Indonesia that Gosling was drinking at that time. That is why the logo of Java is a cup of coffee with steam on top of it (Fig. 2).

Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar.

Figure 2: A Logo of Java Programming

Oracle bought Sun Microsystems in April 2009, and the deal was finalized in January 2010. The acquisition was done intentionally to compete with rival companies in technology since Sun Microsystems was well-established in hardware and popular software technologies such as Java, Solaris, and MySQL. Java is a popular programming language widely used to develop enterprise applications. By acquiring Sun, Oracle would have control over the future development of Java, which would give the company a significant advantage in the enterprise software market. Sun's Solaris operating system and MySQL database were also seen as valuable assets for Oracle.

Therefore, Oracle currently owns Java, which over 3 billion devices use. Java operates on several platforms, including Windows, Mac OS, and UNIX. Java is a general-purpose programming language that allows programmers to code once and execute everywhere. Java is used to create mobile applications, web apps, desktop apps, games, and many other things.

2. Why do we use Java?

1. Robust – Java has built-in features to avoid unnecessary errors during execution, making it safer, stronger, healthier, and faster. e.g., The try-and-catch blocks and other built-in features exposing errors as soon as possible
2. Java is an object-oriented programming language that offers programs a clear structure and enables code to be reused, saving development costs.
3. It is open-source and completely free.
4. Java performs Automatic Garbage collection, freeing up memory when it is no longer used.
5. Platform independent that can run on any platform, e.g., Linux, Windows, Mac OS, etc
6. It is in high demand in today's work environment.
7. It is simple to learn and utilize.

3. Java Architecture

Java architecture defines the overall process of creating, compiling, and running Java Programs. Java architecture is mainly composed of the following: Java Development Kit(JDK), Java Runtime Environment(JRE), and Java Virtual Machine(JVM). The JDK encloses the JRE and other tools such as Java compiler (Javac), Java debugger (Jdb), Java console (JConsole), etc. Also, the JRE comprises JVM and other tools such as Deployment solutions, Integration libraries, language and utility libraries, etc. Similarly, the JVM is composed of different parts which ensure its operations. The explanations of each component of the Java architecture have been provided as follows;

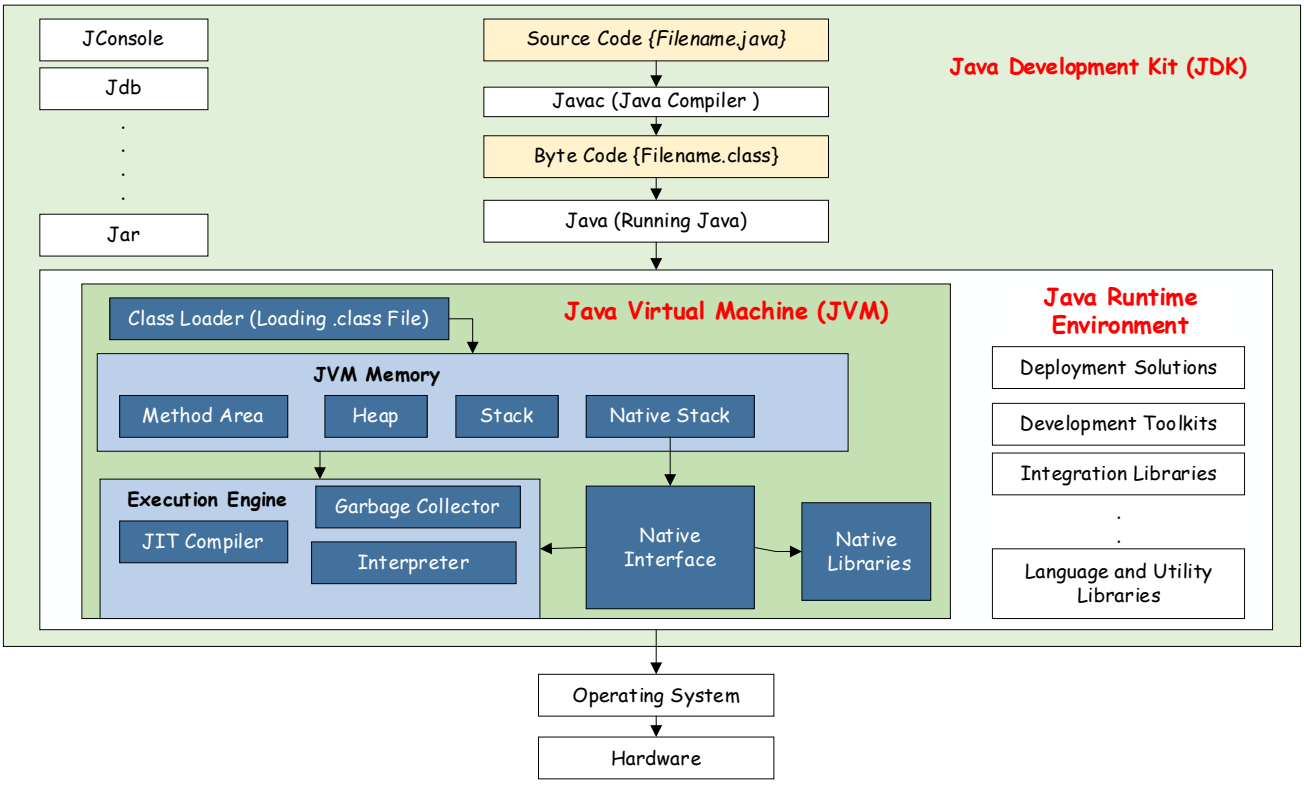


Figure 3: Java architecture

3.1 Java Development Kit

The Java Development Kit (JDK) is a collection of tools for developing and executing Java Programs. The JDK is composed of but is not limited to the following: JRE, Java compiler (Javac), Java debugger (Jdb), Java Console (JConsole), and Java, etc.

A. Java console (JConsole)

JConsole, also known as the Java Monitoring and Management Console, is a graphical tool that enables users to control and monitor Java application activities. When JConsole connects to a Java application, it provides application information such as running threads and loaded classes. This information assists you in monitoring the behavior of your application and the JVM. For example, the data may help you understand performance difficulties, memory use concerns, hang, and deadlocks.

B. Java compiler (Javac)

It defines the Java compiler responsible for converting source code into Java bytecode.

C. Java archive (Jar)

The defines the archiver that bundles relevant class libraries into a single JAR file.This utility helps handle JAR files as well.

D. Java debugger (Jdb)

It is a tool used to find and fix bugs in Java programs

E. Java

It is used to run/launch Java programs (Running Java class file, which has already been converted to byte code.

F. Java Runtime Environment

The Java Runtime Environment (JRE) software creates a runtime environment where Java applications may be executed. The JRE system on disk takes your Java code, integrates it with the necessary libraries, and launches the JVM to execute it. The JRE includes libraries and software required for Java applications to execute. JRE is included with the JDK but may also be downloaded independently.

1. Java Virtual Machine

Because of their flexibility in executing code on any platform, Java programs are referred to as WORA (Write once Run Anywhere). This is only possible because of the JVM. The JVM is a Java platform component that offers an environment for Java application execution. JVM translates bytecode into machine code, executed on the System where the Java application runs. JVM is composed of the following;

i. **Stack:** A data region in JVM memory reserved for a single execution thread. A thread's JVM stack holds numerous items like local variables, partial results, and data for method calls and returns during thread execution. It stores a collection of objects based on the Last-In-First-Out principle (LIFO).

ii. **Class method area:** It is one of the JVM Data Areas where Class data will be stored. This region stores static variables, static blocks, and static and instance methods.

iii. **Class loader:** It is responsible for loading class files to the virtual machine when the Java program is executed/launched

iv. **Heap:** is a memory location in which objects that are instantiated by programs executing on the JVM are stored. Heap uses the Last-In-First-Out principle (LIFO) to store objects.

v. **Native stack:** is a memory location that stores all methods written in languages other than Java.

vi. **Native interface:** JNI is an interface that enables Java to communicate with the code of other languages. It translates codes written in other languages to be compatible with the Java engine.

vii. Execution Engine

- **Interpreter:** This software converts byte codes line by line into native machine code. The conversion process lowers program execution speed.
- **Just-In-Time compiler:** This software aims to improve the performance of Java programs by examining the most frequently used codes by the interpreter and converting them to machine language so that the interpreter will no longer repeat their translation. The JIT compiler is activated by default. When the JIT is deactivated, the execution engine will only use an Interpreter for translation which in turn lowers the execution speed;
- **Garbage Collector:** As the name suggests, Garbage Collector means to gather up things that are no longer needed. The Garbage collection does this work in JVM. It keeps track of every object in the JVM heap space and removes the ones that aren't required.

Note that Having an Interpreter and Just-In-Time compiler makes the Java program both compiled and Interpreted language.

2. Deployment solutions

A collection of deployment technologies like Java Web Start and Java Plugin make activating apps easier and provide advanced support for future Java updates.

3. Development toolkits

The JRE also includes toolkits to assist developers in improving Java programs' user interface. Some of these toolkits are as follows:

- Abstract Window Toolkit (AWT)
- **Swing:** A graphical user interface (GUI) tool for creating objects, buttons, scroll bars, and windows.

4. Integration libraries

The Java Runtime Environment offers a variety of integration libraries to help developers build smooth data connections between their applications and services. These libraries include, among others:

- **Java Database Connectivity (JDBC) API:** Developers may use these technologies to create apps that can access remote relational databases, flat files, and spreadsheets.
- **Java IDL (CORBA):** Supports Java's distributed objects through the Common Object Request Architecture.
- **Java Naming and Directory Interface (JNDI):** A client-side programming interface and directory service that enables users to construct portable apps that can get information from remote databases using naming conventions.

4. Components of Java Program

If you are familiar with Java's building blocks, you have the foundational knowledge necessary to become an expert Java programmer. In addition, the more components you are familiar with, the quicker and easier you will pick up Java. Java Program components are the Module, Package, Class, Variable, Statement, Method, Constructor, and Inner Class.

1. Module

In Java, a module is just a collection of packages. The Java files for creating an application are kept within a package. These packages come together to form modules. There must be at least one module in every Java real-time project.

2. Package

These are groups or classes. Within a single Java package, you may store numerous interconnected class files.

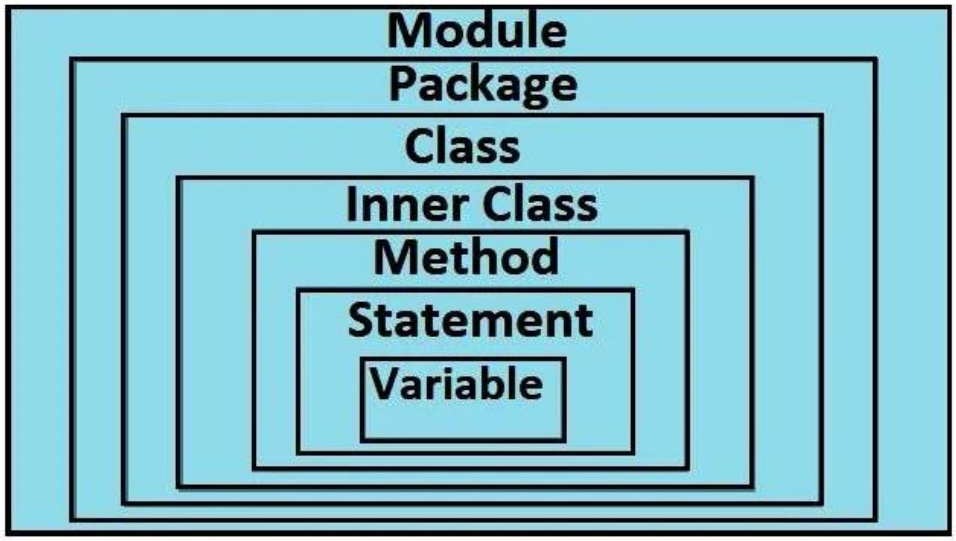


Figure 4: Components of Java program

3. Class

A class is a collection of variables, constructors, methods, blocks, and inner classes. You cannot create a single functional Java application without a class. Therefore, class is a Java program's most crucial and necessary component. Additionally, from the perspective of an object-oriented language, classes are the only tool you can use to construct objects.

4. Variable

The variable is a name given to a computer memory location for storing values. The values stored in this location can change during program execution.

5. Statement

Java statements are commands that instruct the programming language on what to do. For example, a statement to assign value to a variable, looping statements, and control floor statements such as *switch case, while loop, do while loop*, *if* and *if else* statements

6. Method

A method is a chunk of code that only executes when invoked(called). Functions are usually used to perform specific tasks, such as deposit withdrawal.

7. Constructors

A constructor is a method that has the same name as a class. Usually, constructors are automatically called when the class objects are created.

5. Java Platforms / Editions

There are four (4) platforms or editions of Java:

1. Java SE (Java Standard Edition)

The SE stands for **Java Standard Edition**, a computing platform in which we can execute software, and it can be used to develop and deploy portable code for desktop and server environments. It uses the Java programming language. It is part of the Java software platform family.

2. Java EE (Java Enterprise Edition)

Java EE stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE. It is a set of specifications wrapping around Java SE (Standard Edition). The Java EE provides a platform for developers with enterprise features such as distributed computing and web services. Examples of some contexts where Java EE is used are e-commerce, accounting, and banking information systems.

3. Java ME (Java Micro Edition)

The Java ME stands for **Java Micro Edition**. It is a development and deployment platform of portable code for embedded and mobile devices (sensors, gateways, mobile phones, printers, TV set-top boxes). It is based on **object-oriented Java**. The Java ME has a robust user interface, great security, built-in network protocols, and support for applications that can be downloaded dynamically. Applications developed on Java ME are portable, can run across various devices, and can also leverage the native capabilities of the device.

4. JavaFX

JavaFX is a Java library used to develop Desktop applications and Rich Internet Applications (RIA). It uses a lightweight user interface API.

6. Creating a Java program

Question 1: Write a Java program with the following criteria

1. Module name SpecialCourse
2. Package name ***AllPrograms***
3. ***Double-line comment*** explaining the purpose of the program
4. ***Single-line comment*** showing the beginning of the program
5. Class name ***message***
6. Display on the screen the text ***Welcome to Java Programming***
7. ***Single-line comment*** showing the end of the program

Solution

1. **Procedure 1:** Open the Eclipse IDE

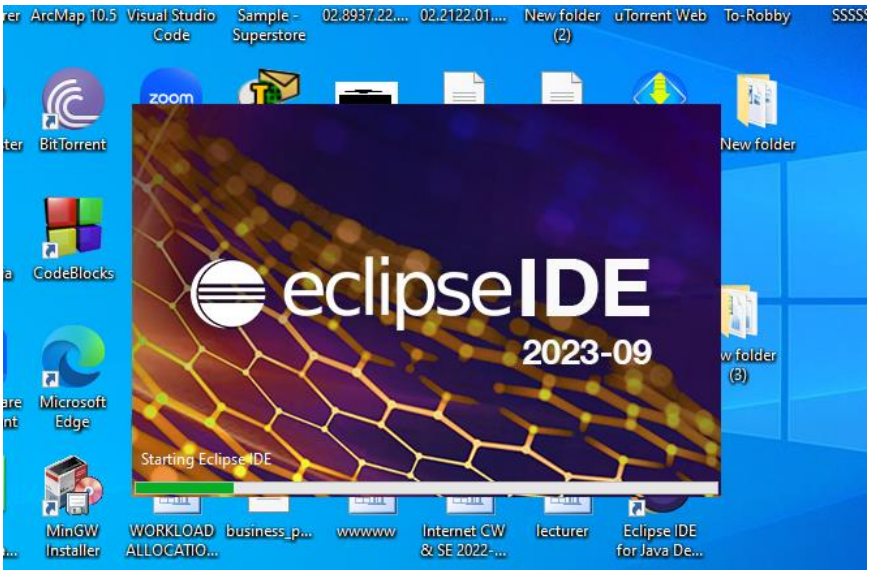


Figure 5: Eclipse IDE initializing

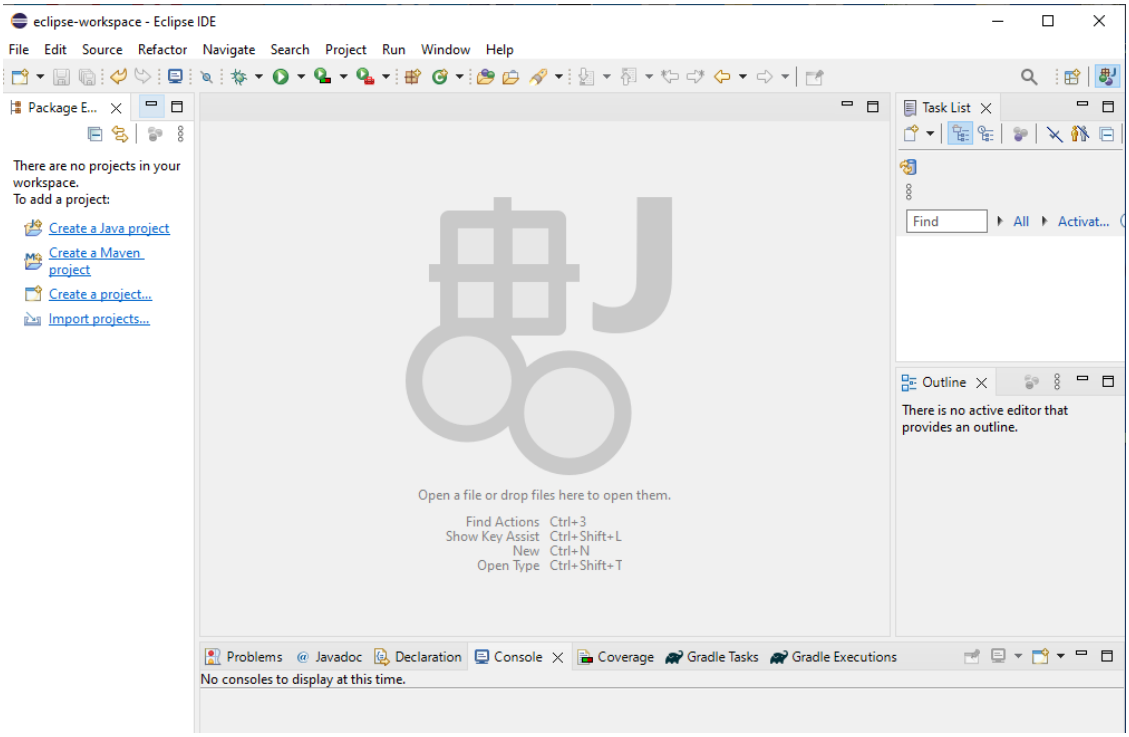


Figure 6: Eclipse IDE Workspace

2. **Procedure 2:** Creating a Java Project and Module

i. **Sub-procedure 1**: Go to **File** menu - > **New** menu- > **Java Project** menu. The new window will appear, then fill in the Project Name as **JavaProgramming** and Module Name as **SpecialCourse.**

ii. ***Sub-procedure 2:*** *Click the* ***Next*** *button.* The default project source path will be shown. This is the default folder where the project files will be stored (***JavaProgramming/Bin***). Note that the Programmer can change it to another folder. *Click the* ***Finish*** *button.*

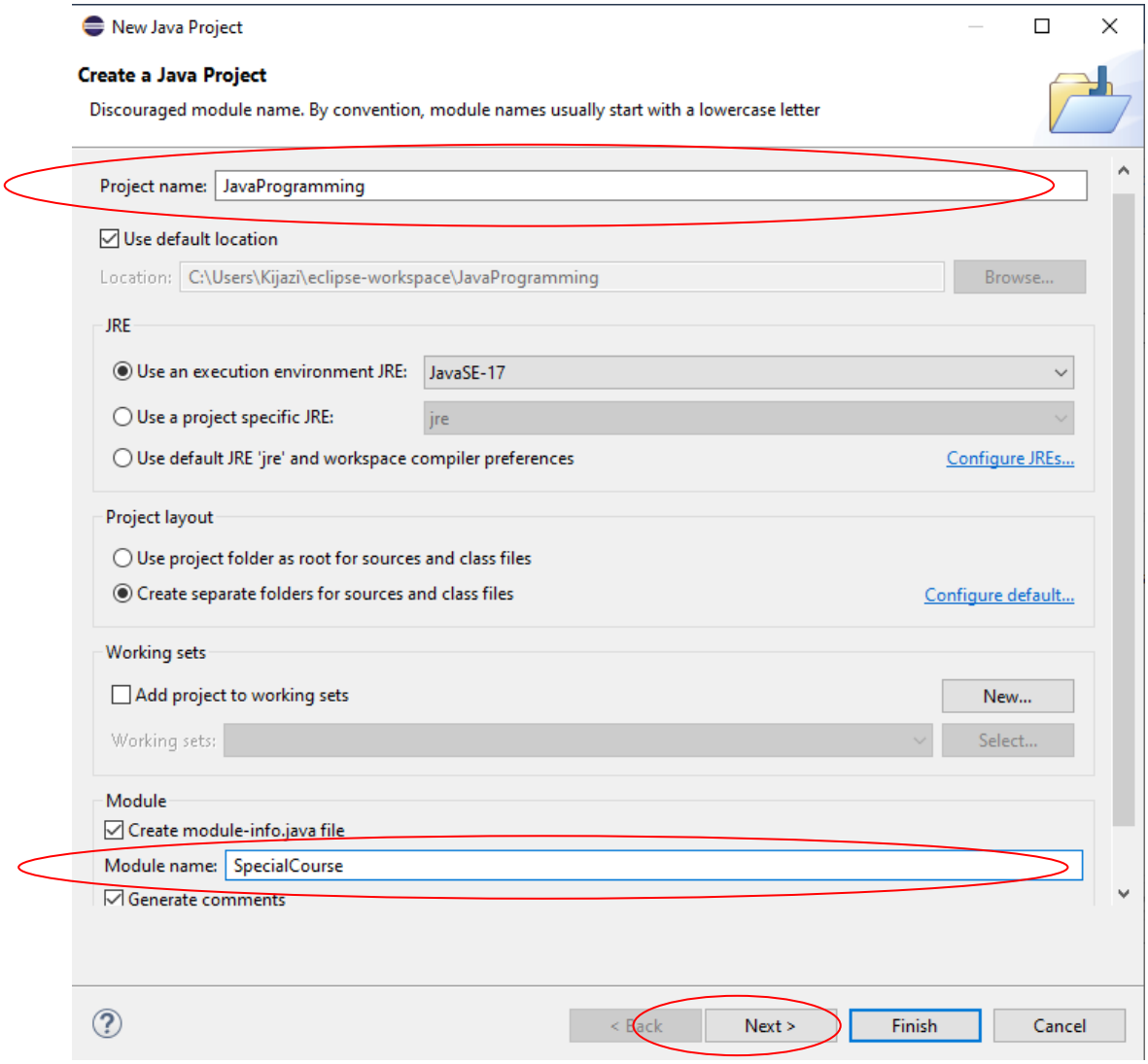


Figure 7: Creating and Java Project and Module

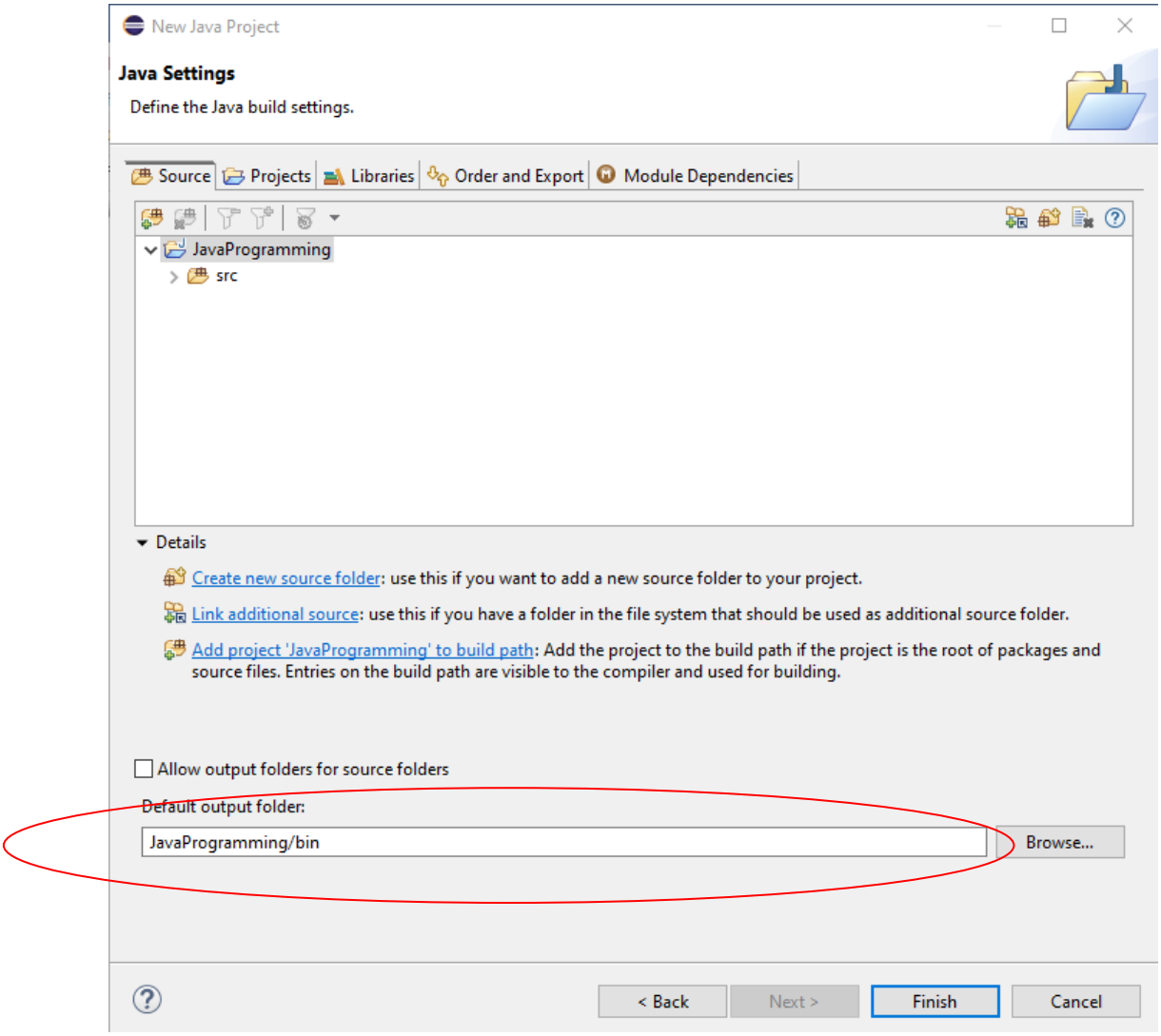


Figure 8: Java Project default output folder

3. **Procedure 3:** Creating a Package
    i. **Sub-procedure 1**: Right-click the **src** folder -> **New** menu -> **Package** menu.
    ii. **Sub-procedure 2**: Fill in the package name as **AllPrograms** and Click the **Finish** button

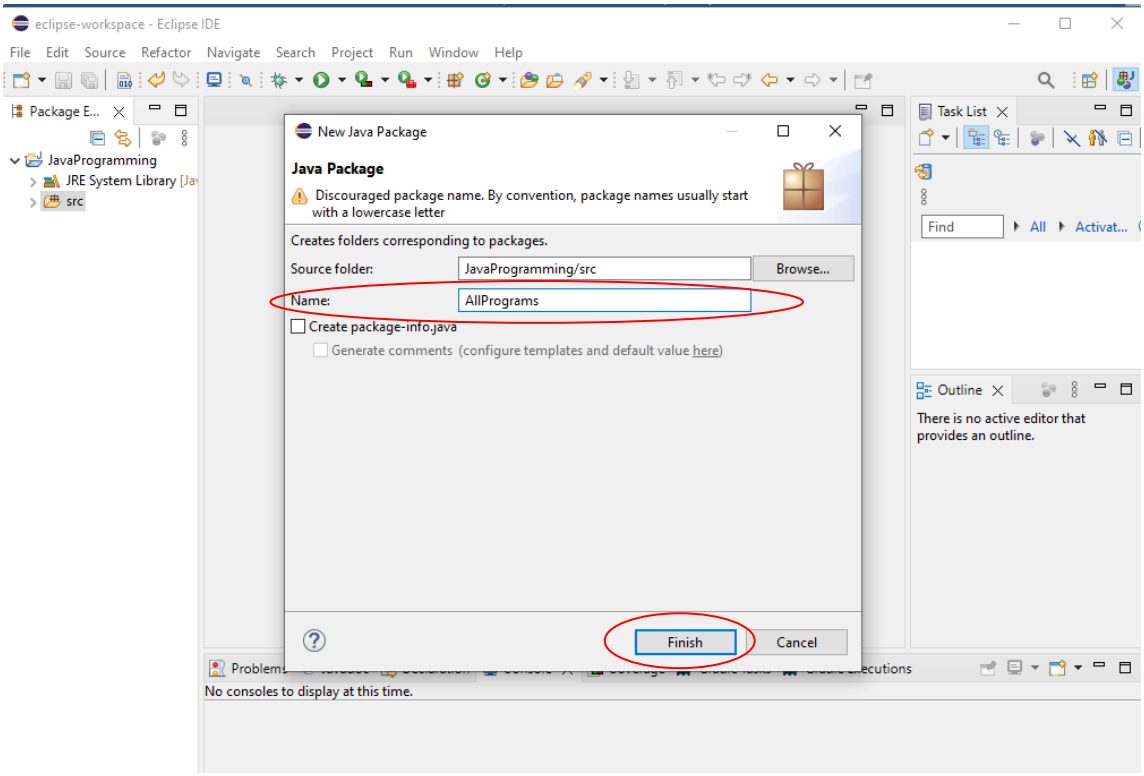


Figure 9: Naming the Java Package

4. **Procedure 4:** Creating a Class
   i. Sub-procedure 1: Right-click the Package (AllPrograms) folder -> New menu -> Class menu.
   ii. ***Sub-procedure 2:*** *Select the* ***public static void main (String args[])*** *checkbox.* The Main function will be automatically written in the program by selecting this checkbox. Otherwise, the Programmer should write it manually.
   iii. ***Sub-procedure 3:*** *Fill in the Class name as a* ***message*** *and Click the* ***Finish*** *button.* The incomplete Java Program will appear with the **Package** name, **Class** name, and **Main** function.

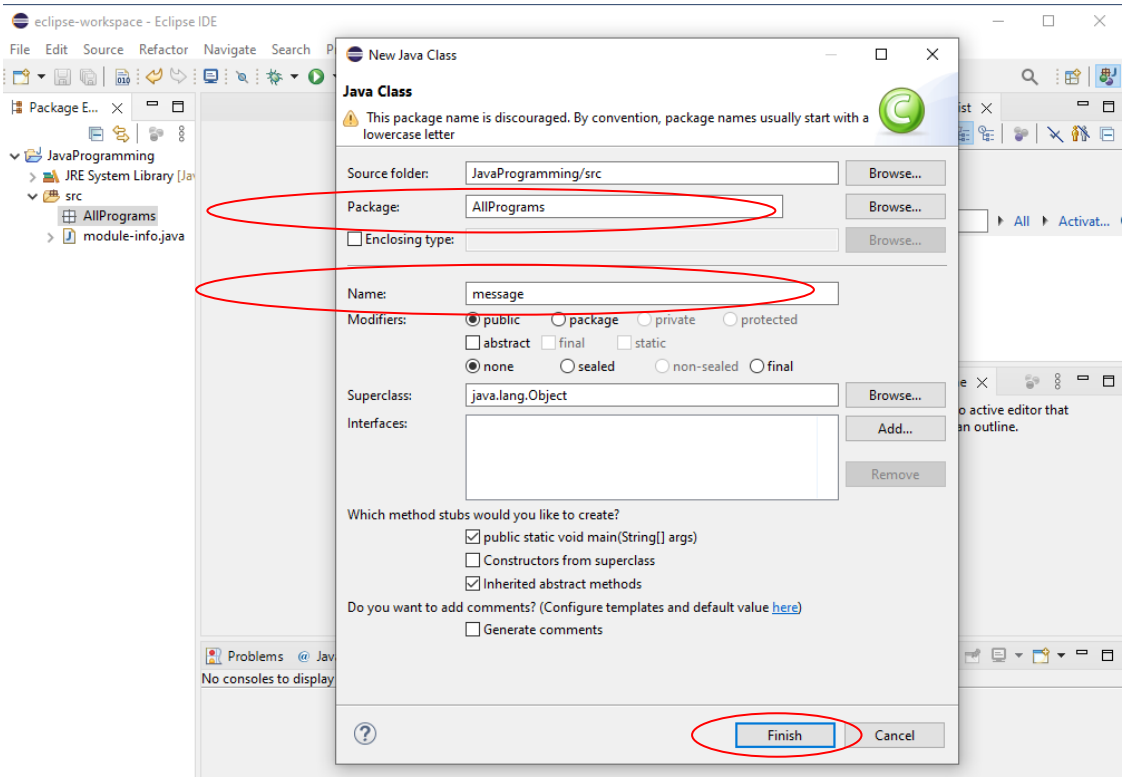


Figure 10: Naming the Java Class

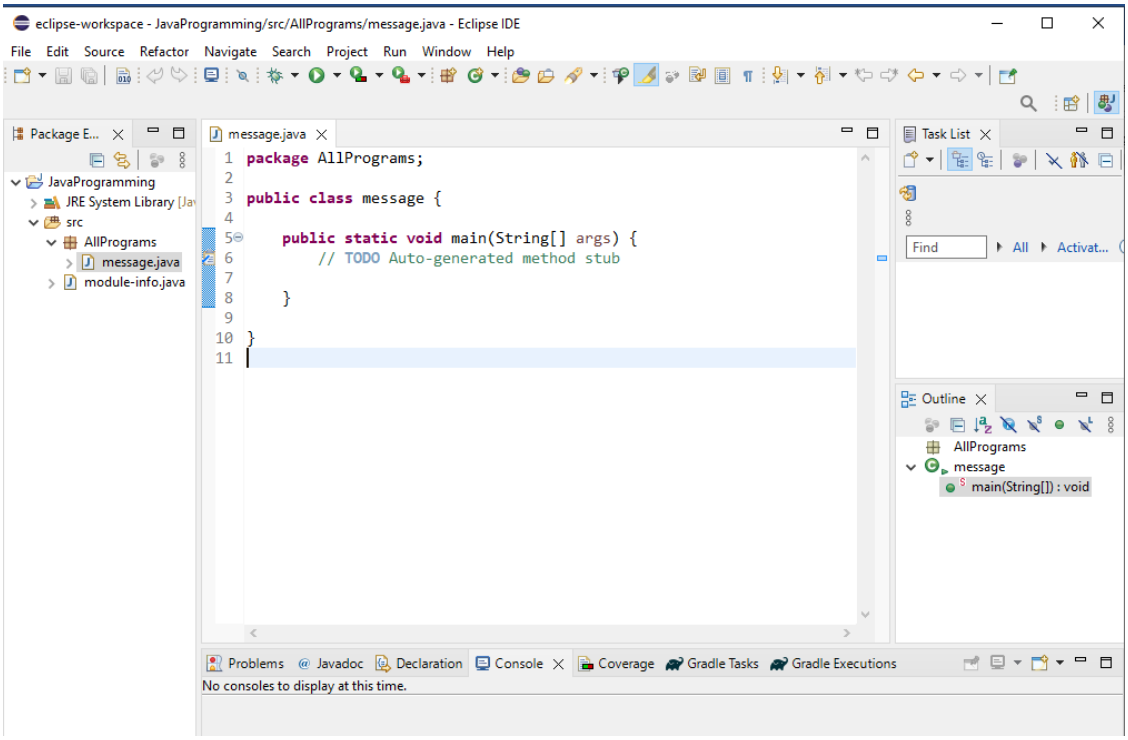


Figure 11: Incomplete Java Program

5. **Procedure 5:** Completing a Java Program by adding comments and the text that has to be displayed on the screen. Click the **run** button to run the program. The text "**Welcome to Java Programming"** will display on the screen.

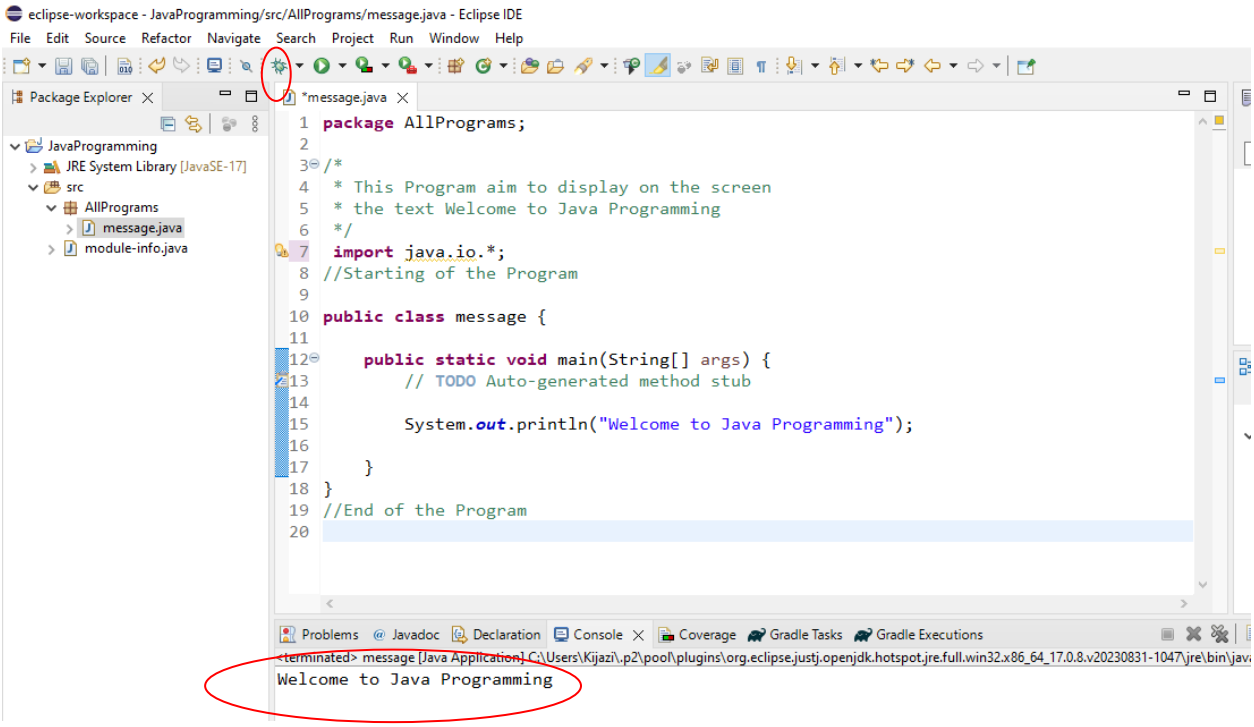


Figure 12: Complete Java Program

6.1 Contents of Java program

From the above program, the following information can be obtained

1. Commenting on Your Programs

We insert **comments** to **document programs** and improve their readability. However, the Java compiler Ignores comments, so they do not cause the computer to perform any action when the program is run.

There are different types of Comments, including;

```
// Single Line comments


/*
    Double Line comments
*/
```

e.g.

```
/*
 * This Program aim to display on the screen
 * the text Welcome to Java Programming
 */


//Starting of the Program
//End of the Program
```

Forgetting one of the delimiters of a comment is a syntax error. A syntax error occurs when the compiler encounters code that violates Java's language rules (i.e., its syntax). Syntax errors are also called compiler, compile-time, or compilation errors because the compiler detects them when compiling the program. When a syntax error is encountered, the compiler issues an error message. This applies not only to comments but also to other components of the program.

2. Declaring a Class

Every Java program consists of at least one class that you (the Programmer) define. The **class keyword** introduces a class declaration and is immediately followed by the **class name** (message). **Keywords** (sometimes called **reserved words**) are reserved for use by Java and

are always spelled with all lowercase letters. The keyword public means that **objects outside the current package can use the class**. By default, a class can only be used by other classes in the same package in which it is declared.

Note that. A **class name** is an identifier—a series of characters consisting of letters, digits, underscores (_), and dollar signs ($) that do not begin with a digit or special symbols rather than (_) and ($)and do not contain spaces. Some valid identifiers are Welcome1, $value, _value, m_inputField1 and button7. The name 7button is not a valid identifier because it begins with a digit, and the name input field is not a valid identifier because it contains a space

Note that: The same rule applies when declaring **variables**, **constants, and packages**

Syntax for declaring a class

public class *classname {*

//Class Body,

}

e.g.

public class message {

}

3. Declaring package

These are groups or classes. Within a single Java package, you may store numerous interconnected class files

The syntax for declaring package

package package_name;

e.g.

```
package AllPrograms;
```

4. Declaring the main method

```
public static void main (String[] args) {


}
```

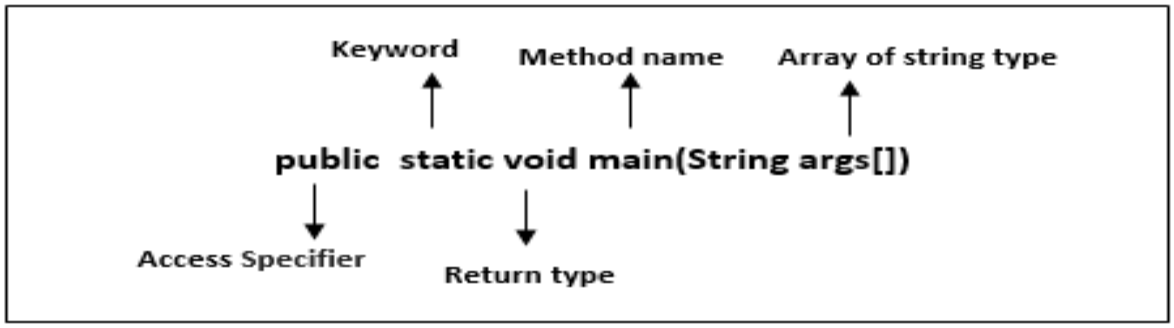


Figure 13: Parts of the main function

The main method begins the execution of the Java application (Which is the starting point of every Java application). Java class declarations normally contain one or more methods. For a Java application, one of the methods must be called main and defined as shown in line 9; otherwise, the Java Virtual Machine (JVM) will not execute the application. See the following keyword and their application.

public: It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use the private, protected, and default methods before the main() method, they will not be visible to JVM.

static: You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods invoke without creating the objects, so we do not need any object to call the main() method.

void: In Java, every method has the return type. The void keyword acknowledges the compiler that the main() method does not return any value.

main: It is the name of the main function

String args[]: The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in string values.

5. Importing java libraries

A Java library is just a collection of classes already written by somebody else. You download those classes, import them into your programs, and use them in your code
e.g.

```
import java.io.*;
```

7. Datatypes

A data type specifies the type of data that a variable can store, such as integer, floating, character, etc.

Data types are divided into two groups:

1. Primitive data types - include byte, short, int, long, float, double, boolean, and char. These are basic data types that are built-in by Java.
2. Non-primitive data types - These are data types that are derived from the Primitive ones such as String, Arrays, and Classes.

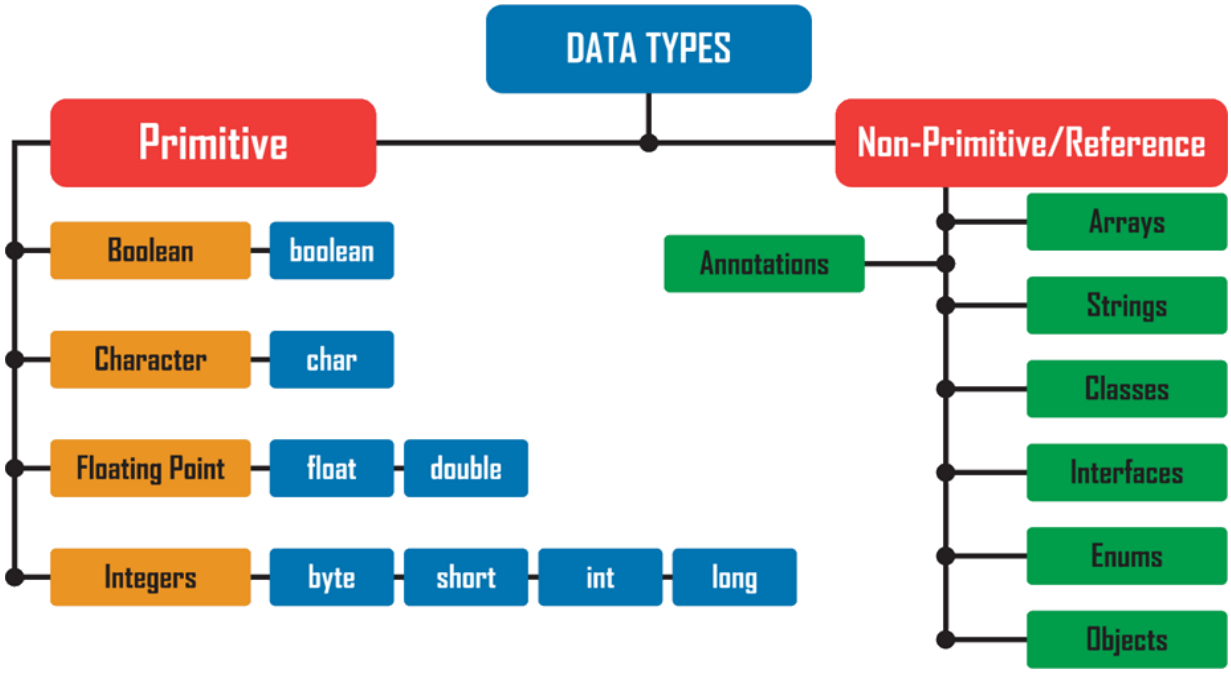


Table1: Primitive datatypes Size, default value and ranges

Data types	Default value	Size	Range
byte	0	1 byte	Stores whole numbers from -128 to 127
short	0	2 bytes	Stores whole numbers from -32,768 to 32,767
int	0	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	0L	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	0.0d	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	false	1 bit	Stores true or false values
char	'\u0000' or Empty	2 bytes	Stores a single character/letter or ASCII values

Program 1: Displaying the size of variables

```
01 package AllPrograms;
02
03 public class AllProgramsClass {
04
05     public static void main (String [] args) {
06
07         System.out.println("int is : " + (Integer.SIZE/8) + " bytes.");
08         System.out.println("long is: " + (Long.SIZE/8) + " bytes.");
09         System.out.println("char is: " + (Character.SIZE/8) + " bytes.");
10         System.out.println("float is: " + (Float.SIZE/8) + " bytes.");
11         System.out.println("double is: " + (Double.SIZE/8) + " bytes.");
12     }
13 }
```

Program 1 output

```
int is: 4 bytes.

long is: 8 bytes.

char is: 2 bytes.

float is: 4 bytes.

double is: 8 bytes.
```

8. Variables

This is a name given to a computer memory location whose values can be changed/altered during the program execution. Therefore, these are containers for storing data.

In Java, there are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers with decimals, such as 19.99 or -19.99

- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

The syntax for declaring (Creating) Variables

Data_type variable_name = value;

e.g.;

int myNum=5;
float myFloatNum=5.99f;
char myLetter='D';
Boolean myBool=true;
String myText="Hello";

8.1 Types of Variables

There are three (3) types of variables, including;

1. Local variable

The **"static" keyword** cannot be used to declare a local variable. Local variables are variables that are declared inside the method, Constructor, or blocks. Only that method, Constructor, or block may utilize this variable, and the other method, Constructor, or block in the class is entirely unaware of its existence.

- Characteristics of local variables
  i. These variables can only be accessed within the function in which they are declared. The lifetime of the local variable is within its function only, which means the variable exists till the function executes.
  ii. Because there is no default value for local variables, they must be declared, and an initial value must be given before usage.
  iii. Access modifiers cannot be used for local variables.

2. Instance variable

An instance variable is declared within the class but outside the body of a method. It has not been declared static. The instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

- **Characteristics of instance variables**

i. The instance variable can be accessed directly by any non-static method, not otherwise when they are in the same class. Similarly, when the child class extends the parent class, the variable can be accessed by the method in the child class without an object. Alternatively, the child class must create a new object using the main class to access the variable.

i. A static method cannot access instance variables directly. However, it can access the instance variables using an object.

ii. Access modifiers can be used for 'instance' variables.

iii. Instance variables have default values. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the Constructor.

3. Static variable

- **Characteristics of static variables**

ii. Static variables have default values

iii. Static variables can be accessed directly by any method, whether static or not, without an object within the class where it was declared. Similarly, when the child class extends the parent class, the variable can be accessed by the method in the child class without an object. Alternatively, the child class must create a new object using the main class to access the variable.

Program 2: Print variables' default values

```
01 package AllPrograms;
02 import java.util.*;
```

```
03 public class AllProgramsClass {
04 int a;
05 float b;
06 double c;
07 char d;
08 long e;
09 String f;
10 public void PrintDefaultValue() {
11         System.out.println("The Default Values are:");
12         System.out.println("Integer "+ a);
13         System.out.println("Float "+ b);
14         System.out.println("Double "+ c);
15         System.out.println("Char "+ d);
16         System.out.println("Long "+ e);
17         System.out.println("String "+ f);
18 }
19 public static void main(String[] args) {
20
21         DisplayingVariableSize obj=new DisplayingVariableSize();
22         obj.PrintDefaultValue();
23 }
24
25 }
```

Program 2 output

```
The Default Values are:
Integer 0
Float 0.0
Double 0.0
Char
String null
```

8.2 Constant (Final Variables)

These are names given to computer memory locations whose value cannot be modified during the program execution. This variable can be either a local, instance, or static variable and obeys the appropriate properties, as explained in the types of variables section (Section 7.1).

Syntax for declaring a *final variable*

final Data_type variable_name*= value;*

e.g.

final int myNum=15;

final static int myNum=15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

9. Operands and Operators

9.1 Operands

Operands are variables or values that operators can manipulate

9.2 Operators

Operators are symbols that perform operations on variables and values. Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups.

1. Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators. Assume integer variable A holds 10 and variable B holds 20, then.

Table2: Arithmetic Operators

Operator	Description	Example	Descriptions
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200	It is evaluated first. If there are

/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2	several operators of this type, they're evaluated from left to right.
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0	
+ (Addition)	Adds values on either side of the operator.	A + B will give 30	It is evaluated next. If there are several operators of this type, they're evaluated from left to right.
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10	
=(Equal/Assignment)	Assign a value to a variable		Evaluated last.

2. Relational Operators

The following relational operators are supported by Java language. Assume variable A holds 10 and variable B holds 20, then.

Table3: Relational Operators

Operator	Description	Example
== (equal to)	Check if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition	(A > B) is not true.

	becomes true.	
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

3. Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types long, int, short, char, and byte. The bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now, in the binary format, they will be as follows;

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators;

Assume integer variable A holds 60 and variable B holds 13 then;

Table 4: Bitwise Operator

Operator	Description	Example
& (bitwise and)	Binary AND operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
| (bitwise or)	Binary OR operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operand value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operand value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 1111
>>> (zero-fill right shift)	Shift right zero fill operator. The left operand value is moved right by the number of bits specified by the right operand, and shifted values are filled up with zeros.	A >>>2 will give 15, which is 0000 1111

4. Logical Operators

The following table lists the logical operators. Assume Boolean variables A holds true and variable B holds false, then.

Table 5: Bitwise Operators

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. The condition becomes true if any of the two operands are non-zero.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make false.	!(A && B) is true

5. Assignment Operators

Following are the assignment operators supported by the Java language

Table 6: Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right-side operands to left-side operands.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It	C *= A is

	multiplies the right operand with the left operand and assigns the result to the the left operand.	equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

10. Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth program flow.

Java provides three types of control flow statements.

1. Decision-Making statements
    i. if statements
        a) Simple if
        b) if…. else….
        c) if…. else if…. else ….
    ii. switch statement
2. Loop statements
    i. do while loop
    ii. while loop
    iii. for loop
    iv. for-each loop
3. Jump statements

i. break statement

ii. continue statement

10.1 Simple if Statements:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a code block if the condition is true.

Syntax

```
if(condition)
{
    Statement 1; //executes when the condition is true
}
```

Question: Given two numbers a=10 and b=20. Write a Java Program to find out which is greater and display it on the screen

Program 3: Adding two given numbers

```
01    package AllPrograms;
02
03    public class AllProgramsClass
04    {
05       public static void main (String[] args)
06       {
07          int a=10;
08          int b=20;
09          if(a>b)
10          {
11             System.out.println("a Is Greater");
12          }
13          if(b>a)
14          {
15             System.out.println("b Is Greater");
16          }
17          if(a==b)
18          {
19             System.out.println("a and b are equal");
20          }
21       }
22    }
```

```
23
```

Program 3 output

```
b Is Greater
```

10.2 if-else- statement

The *if--else--* statement provides an alternative block (the else block) that executes when the condition is false. Apart from the *if--* statement that provides no alternative block when the condition is false.

Syntax

```
if(condition)
{
  Statement 1; //executes when the condition is true.
}
else
{
   Statement 2; //executes when the condition is false
 }
```

Question: Write a Java program to find the largest number between two input numbers.

Note that your program should receive numbers as input from the use

Program 4: Find the largest between two input numbers

```
01    package AllPrograms;
02    import java.util.*;
03    public class AllProgramsClass
04    {
05       public static void main (String[] args)
06       {
07          float a;
08          float b;
09          Scanner numbers=new Scanner (System.in);
10          System.out.println("Enter First Number");
11          a=numbers. nextFloat();
12          System.out.println("Enter Second Number");
13          b=numbers. nextFloat();
14          if(a>b)
15          {
16             System.out.println("First Number is Greater");
```

```
17          }
18          else if(b>a)
19          {
20             System.out.println("Second Number is Greater");
21          }
22          else
23          {
24             System.out.println("a and b are equal");
25          }
26       }
27    }
```

Program 4 output

```
Enter First Number
1000
Enter Second Number
2000
Second Number is Greater
```

10.3 The if-else-if- else- statement

The *if-else-if-else-* statement contains the if-statement followed by multiple *else-if* statements. This statement allows the program to test multiple conditions before executing an alternative block (*else block*). When the particular condition is true, its corresponding block is executed. Similarly, the *else* block is executed when neither of the conditions is true.

Syntax

```
if (condition 1)
{
   Statement 1; //executes when condition 1 is true
}
else if (condition 2)
{
   Statement 2; //executes when condition 2 is true
}
```

```
else
{
    Statement 3; //executes when all the conditions are false
}
```

Question: Assume that CBE always assigns Diploma students' to different Bachelor's Degree Courses based on their GPA qualification, as shown in Table 6.CBE implemented a simple program to automate this task to avoid paperwork and save time. Assume that you are working as a programmer at CBE. Write a simple Java program that can perform this task.

Table 6: GPA ranges vs academic Departments

GPA Range	Department
4.5 - 5.0	ICT
4.5 - 5.0	Legal Metrology
3.5 - 4.4	Accounts
2.1 - 3.4	Procurement
2.1 - 3.4	Marketing

Program 5: Displaying the department

```
01    package AllPrograms;
02    import java.util.*;
03    public class  AllProgramsClass {
04
05       public static void main (String[] args)
06       {
07          float GPA;
08          Scanner input=new Scanner(System.in);
09          System.out.println("Enter Student GPA");
10          GPA=input.nextFloat();
11
12          if(GPA>=4.5 && GPA<=5.0)
13          {
14             System.out.println("Students can be allocated to the ICT
```

```
15                or Legal Metrology Department");
16            }
17            else if(GPA>=3.5 && GPA<=4.4)
18            {
19               System.out.println("Student can be allocated in Accounts Department");
20            }
21            else if(GPA>=2.1 && GPA<=3.4)
22            {
23              System.out.println("Student can be allocated
24              to the Procurement or Marketing Department");
25            }
26            else
27            {
28              System.out.println("Student does not qualify for any Department");
29            }
30        }
31    }
```

Program 5 output

```
Enter Student GPA
4.7
Students can be allocated to the ICT or Legal Metrology Department
```

Question: Write a Java program to display the student's examination results. Your program should perform the following: Accept the Student name, English Marks, Geography Marks, and Biology Marks as inputs from the user; also calculate the average and grade based on the given average marks range as shown in Table 7. Finally, the program should display student names, marks obtained from each subject, average marks, and grades obtained.

Table 7: Average Score vs Grades

Average score	Grade
80 - 100	A
70 - 79	B+

60 - 69	B
50 - 59	C
0 - 49	F

Program 6: Displaying student scores

```
01    package AllPrograms;
02    import java.util.*;
03    public class AllProgramsClass {
04
05       public static void main (String[] args)
06       {
07          String student_name;
08          float English;
09          float Geography;
10          float Biology;
11          float average;
12          Scanner input=new Scanner(System.in);
13          System.out.print("Enter Student Name");
14          student_name=input.nextLine();
15          System.out.print("Enter English Marks");
16          English=input.nextFloat();
17          System.out.print("Enter Geography Marks");
18          Geography=input.nextFloat();
19          System.out.print("Enter Biology Marks");
20          Biology=input.nextFloat();
21          average=(English+Geography+Biology)/3;
22          System.out.println(".... Student Results.....");
23          System.out.println("Student Name is "+ student_name);
24          if(average>=80 && average<=100)
25             System.out.println("Student Grade is A");
26          else if(average>=70 && average<=79)
27             System.out.println("Student Grade is B+");
28          else if(average>=60 && average<=69)
29             System.out.println("Student Grade is B");
30          else if(average>=50 && average<=59)
31             System.out.println("Student Grade is C");
32          else if(average>=0 && average<=49)
```

```
33              System.out.println("Student Grade is F");
34          else
35              System.out.println("Enter the correct score");
36      }
37  }
```

Program 6 Output

```
Enter Student Name
AHMED ATHUMAN
Enter English Marks
70
Enter Geography Marks
80
Enter Biology Marks
50
.....Student Results.....
Student Name is AHMED ATHUMAN
Student Grade is B
```

10.4 Switch Case Statement

The Java *switch statement* executes one statement from multiple conditions. It is like the if-else-if ladder statement; it tests the expression value against each case value. It executes the case value body if the expression value matches the case value. However, it executes the default body if no match is found. Each case statement can have an optional break statement. When control reaches the break statement, it exits the switch statement. If a break statement is not found, it executes the next case.

Syntax

```
Switch (expression)
{
case value1:
  //code to be executed;
 break;  //optional
```

```
case value2:
 //code to be executed;
 break;  //optional
.....
default:
  Code to be executed if all cases are not matched;
}
```

Question: Using a switch case statement, write a Java program to display the module name based on the given module code considering Table 3. Note that your program should accept module code as input from the User (*Also observe the output without using the break statements*).

Table 3: Module code against the department

Module code	Department
ITU07313	Data Structure and Algorithm
ITU07312	Programming in Java
ITU07212	Programming in C++

Program: Displaying course name

```
01    package AllPrograms;
02    import java. util.Scanner;
03    public class AllProgramsClass {
04          public static void main(String[] args)
05       {
06          Scanner input=new Scanner(System.in);
07          System.out.println("Enter Module Code");
08          String modulecode=input.nextLine();
09          switch(modulecode)
10          {
11             case "ITU07313":
12                System.out.println("Data Structure and Algorithm");
13                break;
```

```
14              case "ITU07312":
15                  System.out.println("Programming in Java");
16                  break;
17              case "ITU07212":
18                  System.out.println("Programming in C++");
19                  break;
20              default:
21                  System.out.println("Unknown Module");
22          }
23      }
24  }
```

10.5 Loop statements

Looping statements allow particular instructions to be executed repeatedly until a condition is met. Consider a circumstance where you must print numbers ranging from 1 to 1000. How would you react? Will you type **System.out.print()** thousands of times or try to copy/paste it? Hopefully, it will be a tedious job. By using looping statements, this process can be completed efficiently. However, this is the most fundamental use of looping statements; they can also be deployed in various circumstances.

10.5.1 for loop

Like a while loop, the for loop executes its internal part only if the condition is valid in each execution. Therefore, before performing any iteration, the for loop tests the condition to see if it is true.

Syntax

```
for(init; condition, incr/decr)
{
  //Statements to be executed or body
}
```

In the above syntax:

- init: The init expression is used for initializing a variable and is executed only once.
- Condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false.
- incr/decr: The increment or decrement statement is applied to the variable to update the initial expression.

Note that: *The "init, condition, and incr/decr" parts are enclosed inside the brackets in the for loop syntax*

Question: Write a Java program to display 1 to 10 using a for-loop

Program: Displaying 1 to 5 using a for-loop

```
01      package AllPrograms;
02     public class AllProgramsClass {
03             public static void main(String[] args)
04         {
05            for(int i=1;i<=5;i++)
06            {
07              System.out.println(i);
08            }
09         }
10     }
```

Output

```
1
2
3
4
5
```

Question: Write a Java program to display 2, 4, 6, 8, 10 using a for-loop

Program: Displaying arithmetic progression series using a for-loop

```
01     package AllPrograms;
02     public class AllProgramsClass {
03         public static void main(String[] args)
```

```
04        {
05           for (int i=1; i<=5; i++)
06           {
07            int n=2*i;
08            System.out.println(n);
09           }
10        }
11     }
```

Output

```
2
4
6
8
10
```

Question: Write a Java program to display the following shape using a for-loop

```
*
**
***
****
*****
******
```

Program: Displaying Christmas tree using a for-loop

```
01     package AllPrograms;
02     public class AllProgramsClass {
03       public static void main(String[] args)
04        {
05           for(int i=1; i<=5;i++)
06           {
07              for(int j=1;j<=i;j++)
08              {
09                 System.out.print("*");
10              }
```

```
11              System.out.print("\n");
12          }
13      }
14  }
```

Output

```
*
**
***
****
*****
```

Question: Write a Java program to display the following shape using a for-loop

```
********
*******
******
*****
****
***
**
*
```

Program: Displaying an inverted Christmas tree using a for-loop

```
01  package AllPrograms;
02  public class AllProgramsClass {
03          public static void main(String[] args)
04      {
05          for(int i=0; i<=8;i++)
06              {
07                  for(int j=1;j<=8-i;j++)
08                  {
09                    System.out.print("*");
10                  }
11                    System.out.print("\n");
12              }
13      }
14  }
```

Output

```
********
*******
******
*****
****
***
**
*
```

Question: Write a Java program to display the following shape using a for-loop

```
2
2 4
2 4 6
2 4 6 8
```

Program: Displaying triangular-shaped arithmetic progression series using a for-loop

```
01    package AllPrograms;
02    public class AllProgramsClass {
03
04        public static void main(String[] args)
05        {
06                int n;
07                for(int i=1; i<=4;i++)
08                {
09                    for(int j=1;j<=i;j++)
10                    {
11                      n=2*j;
12                      System.out.print(" "+n);
13                    }
14                    System.out.print("\n");
15                }
16        }
17    }
```

Output

```
2
2 4
2 4 6
2 4 6 8
```

Question: Write a Java program to display the following shape using a for-loop

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Program: Displaying numerical inverted Christmas tree using a for-loop

```
01    package AllPrograms;
02    public class AllProgramsClass {
03       public static void main(String[] args)
04       {
05              for(int i=0; i<=5;i++)
06              {
07                 for(int j=1;j<=5-i;j++)
08                 {
09                   System.out.print(j);
10                 }
11                   System.out.print("\n");
12              }
13       }
14    }
```

Output

```
12345
1234
123
12
1
```

10.5.2 while loop

Like a for loop, the while loop executes its internal part only if the condition is valid in each execution. Therefore, before performing any iteration, the while loop tests the condition to see if it is true. The while loop is similar to the for loop; however, the init, condition, and the incr/decr part are separate (not enclosed inside the bracket)

Syntax

```
init;
while (condition)
{
  //Statements to be executed or body
  incr/decr;
}
```

In the above syntax:

- init: The init expression is used to initialize a variable and is executed only once.
- condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false
- ncr/decr: The increment or decrement statement is applied to the variable to update the initial expression.

Question: Write a Java program to display 1 to 5 using a while loop

Program: Displaying 1 to 5 using a while loop

```
01    package AllPrograms;
02    public class AllProgramsClass {
03
04        public static void main(String[] args)
05        {
06              int n=1;
07              while(n<=5)
08              {
09                    System.out.println(n);
10                    n++;
11              }
12        }
13    }
```

Output

```
1
2
3
```

```
4
5
```

Exercise: Attempt all programs completed using the for-loop statement above using a while loop

10.5.3 do-while loop

Unlike a for loop and while loop, the do-while executes its internal part only if the condition is valid in each execution proceeding the first one. This means the first execution is a must for the do-while loop. Like while loop, the init, condition, and the incr/decr part are separate (not enclosed inside the bracket)

Syntax

```
init;
do
{
  //Statements to be executed or body
   incr/decr;
}
while(condition);
```

In the above syntax:

- init: The init expression is used to initialize a variable and is executed only once.
- condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false.

- incr/decr: The increment or decrement statement is applied to the variable to update the initial expression.

Question: Write a Java program to display 1 to 5 using a do-while loop

Program: Displaying 1 to 5 using a do-while loop

```
01    package AllPrograms;
02    public class AllProgramsClass {
03
04        public static void main(String[] args)
05        {
06                int i=1;
07                do
08                {
09                        System.out.println(i);
10                        i++;
11                }
12                while(i<=5);
13        }
14    }
```

Output

```
1
2
3
4
```

Exercise: Attempt all programs completed using the for-loop statement above using a do-while loop

10.6 Jump statements

10.7 Break

The Break Statement is a loop control statement used to end the loop. When the execution meets the breaks statement, it stops.

The syntax for the "break" statement

```
break;
```

The syntax when the "**break**" statement is used in the for-loop statement

```
for (init; condition, incr/decr)
{
        if(condition)
        {
            break;
        }
}
```

Note that: The loop will end its execution at the instant the condition is true

Question: Write a Java program which will terminate the for-loop execution in the 3rd iteration

Program: To terminate the for-loop execution after the 3rd iteration

```
01  package AllPrograms;
02  import java.util.Scanner;
03  public class AllProgramsClass {
04
05      public static void main (String [] args)
06      {
07          for (int i=1; i<=8; i++)
08          {
09                  System.out.println("Iteration No "+i);
10                  if(i==3)
11                  {
12                          break;
13                  }
14          }
15      }
16  }
```

Output

```
Iteration No 1
Iteration No 2
Iteration No 3
```

10.8 Continue

The continue statement is used inside loop control structures in computer languages to control the execution flow. When a continue statement is met within the loop, the control skips the entire code following it after a certain condition has been met for the current iteration and continues to the subsequent iterations. The Java continue statement can be used in any form of loop, but it is most commonly found in for, while, and do-while

The syntax for the "continue" statement

```
continue;
```

The syntax when the "**continue**" statement is used in the for-loop statement

```
for(init; condition, incr/decr)
{
        //This statement will always be executed
        if(condition)
        {
            Continue;
        }
        // This statement will not be executed when the condition is true
}
```

Question: Write a Java program to display 1, 2, 3,6,7,8 using a for loop

Program: Displaying 1, 2, 3, 6, 7, 8 using a for loop

```
01  package AllPrograms;
02  import java. util.Scanner;
03  public class AllProgramsClass {
04
05    public static void main(String[] args)
06     {
07          for(int i=1;i<=8;i++)
08          {
09                if(i==4||i==5)
10                {
11                      continue;
12                }
13                System.out.println(i);
14          }
15     }
16  }
```

Output

```
1
2
3
6
7
8
```

11. Java Arrays

Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java Array depending on its size. There are two types of arrays, namely.:

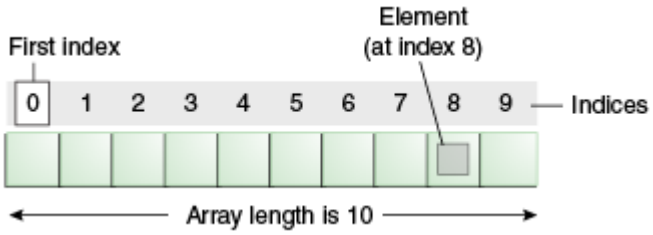
i Single-dimensional Array

ii Multi-dimensional Array

11.1.1 Single Dimensional Array

Single-dimension Array stores Elements in linear order (rows only). It stores the 1st Element at the 0 index, the 2nd at the 1 index, and so on.

Consider a description of an array named data[10] below;



There are three (3) syntaxes for declaring a single-dimension array

i Data_type arr[ ]; or

ii Data_type [ ]arr; or

iii Data_type [ ] arr

The **data_type** can be primitive data types like **int, char, double, byte, etc., or Java objects.** The **arr** is an identifier/Array name. An array name can be any name that follows the rules for naming variables.

But how many elements can this Array hold?

That is an excellent question! In Java, we must allocate memory for the Array to define the number of elements it can hold. The process of allocating the memory size of a Java Array is also known as **instantiation**. For example, an example,

arr=new Data_type [Array size];

For example, Separately **declaring** and **instantiates** an Array named "data" as an integer of size 10 using the first syntax

```
int data[ ];            //Array declaration
data=new int[10];  //Array initialization
```

Also, Arrays can be declared and initialized at the same time.

i Data_type arr[ ]=new Data_type [Array size] ;

ii Data_type [ ]arr=new Data_type [Array size] ;

iii Datatype[ ] arr=new Data_type [Array size] ;

For example, parallel **declares and instantiates** an Array named "data" as an integer of size 10 using all syntaxes.

```
i    int data[ ]=new int [10] ;
ii   int  [ ]data =new int[10] ;
iii  int[ ]  data =new int [10] ;
```

How do you initialize Single-dimensional arrays in Java?

Java Array initialization is also known as storing Elements in Java Array. Consider storing the Elements 100, 234,100,4,19,100,345 as integers to an Array named as **data.** Various mechanisms can be used as follows;

First Mechanism

```
int data[ ];            // Declaration
data=new int[7];   // instantiation
data[0]=100;        // Initializations
data[1]=234;
```

data[2]=100;

data[3]=4;

data[4]=19;

data[5]=100;

data[6]=345;

Second Mechanism

int data[ ]=new int[7]; // Declaration and instantiation.

data[0]=100; // Initializations

data[1]=234;

data[2]=100;

data[3]=4;

data[4]=19;

data[5]=100;

data[6]=345;

Third Mechanism

int data[ ]=new int[ ]{ 100, 234,100,4,19,100,345 } // Declaration, instantiation and Initialization

Fourth Mechanism

int data[ ]={ 100, 234,100,4,19,100,345 } // Declaration, instantiation and Initialization

Question: Write the program to perform the following to an Array named "data" below

An array "int data[]=new int[7]"

Elements	100	234	100	4	19	100	345

Index	0	1	2	3	4	5	6

i Store the given Elements in an array "A".
ii Display the 3rd element.
iii Display the Elements at index 2.
iv Update the Element 234 to be 5000
v Display all Elements Using a for-loop statement

Program: Storing and displaying Array Elements in Single-dimension Array

```
01    package AllPrograms;
02    import java.util.Scanner;
03    public class AllProgramsClass {
04
05      public static void main(String[] args)
06       {
07                  int data[]=new int[] {100, 234,100,4,19,100,345};
08                  //Displaying the third element
09                  System.out.println("The third Element is "+data[2]);
10
11                  //Displaying the Element at Index 2
12                  System.out.println("The Element at Index 2 is "+data[2]);
13
14                  //Updating the Element 234 to 5000
15                  data[1]=5000;
16
17                  //Displaying all Elements Using for-loop statement
18                  System.out.println("The following are the Elements:");
19                  for(int i=0;i<=6;i++)
20                  {
21                        System.out.println(data[i]);
22                  }
23       }
24    }
25
```

Output

```
The third element is 100
The Element at Index 2 is 100
The following are the Elements:
100
5000
100
4
19
100
345
```

11.1.2 Multi-Dimensional Array

Single-dimension Array stores Elements in Matrix form (rows and columns). It stores the 1st Element at the 00 index, the 2nd at the 01 index, the 3rd at the 02 index, and so on. Consider a description of an array named data[3][5] that stores fifteen elements as below. The [3] of the "data" Array indicates the number of rows, while [5] indicates the number of rows. Therefore the array data[3][5] has 3 rows and 5 columns

int data[ ][ ]=new int [3][5]

Elements	First	Second	Third	Fourth	Fith
Indices	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
Elements	Sixth	Seventh	Eighth	Ninth	Tenth
Indices	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
Elements	Eleventh	Twelves	Thirteenth	Fourteenth	Fifteenth
Indices	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

There are three (3) syntaxes for declaring a Multi-dimension array

i Data_type arr[ ][ ]; or

ii Data_type [ ][ ]arr; or

iii Datatype[ ][ ] arr

The **data_type** can be primitive data types like **int, char, double, byte, etc., or Java objects.** The **arr** is an identifier/Array name. Array name can be any name that follows the rules of naming variables.

But how many elements can this Array hold?

That is an excellent question! In Java, we must allocate memory for the Array to define the number of elements it can hold. The process of allocating the memory size of a Java Array is also known as **instantiation**. For example an example,

arr=new Datatype[Array size] [Array size] ;

For example, Separately **declaring** and **instantiates** an Array named "data" as an integer of 3 rows and five columns using the first syntax

```
int data[ ][ ];              //Array declaration
data=new int[3][5];          //Array initialization
```

Also, Arrays can be declared and initialized at the same time.

i Data_type arr[ ][ ]=new Data_type [Array size] [Array size];

ii Data_type [ ][ ]arr=new Data_type [Array size] [Array size];

iii Datatype[ ][ ] arr=new Data_type [Array size] [Array size];

For example, parallel **declares and instantiates** an Array named "data" as an integer of the size 3 rows and 5 columns using all syntaxes.

i int data[ ][ ]=new int [3][5] ;

ii int [ ][ ]data =new int [3][5] ;

iii int[ ][ ] data =new int [3][5] ;

How do you initialize Multi-dimension Arrays in Java?

Java Array initialization is also known as storing Elements in Java Array. Consider storing the Elements 100, 234,100,4,19,345,45,67,90,1000,45,67,89,69,35 as integers to an Array named as **data** as an integer of 3 rows and 4 columns. Various mechanisms can be used as follows; {100, 234,100,4,19},{100,345,45,67,90},{1000,45,67,89,69}}

First Mechanism

```
int data[ ][ ];            // Declaration
data=new int[3][5];   // instantiation
data[0][0]=100;        // Initializations
data[0][1]=234;
data[0][2]=100;
data[0][3]=4;
data[0][4]=19;
data[1][0]=345;
data[1][1]=45;
data[1][2]=67;
data[1][3]=90;
data[1][4]=1000;
data[2][0]=45;
data[2][1]=67;
data[2][2]=89;
data[2][3]=69;
data[2][4]=35;
```

Second Mechanism

```
int data[ ][ ]=new int[3][5];   // Declaration and instantiation.
data[0][0]=100;          // Initializations
data[0][1]=234;
data[0][2]=100;
data[0][3]=4;
data[0][4]=19;
data[1][0]=345;
data[1][1]=45;
data[1][2]=67;
data[1][3]=90;
data[1][4]=1000;
data[2][0]=45;
data[2][1]=67;
data[2][2]=89;
data[2][3]=69;
data[2][4]=35;
```

Third Mechanism

```
int data[ ][ ]=new int[ ][ ]{{100, 234,100,4,19},{345,45,67,90,1000},{45,67,89,69,35}};
 // Declaration, instantiation, and Initialization
```

Fourth Mechanism

```
int data[ ]={{100, 234,100,4,19},{345,45,67,90,1000},{45,67,89,69,35}};
// Declaration, instantiation, and Initialization
```

Question: Write the program to perform the following to an Array named "data" below;

```
int data[ ][ ]=new int [3][5]
```

Elements	100	234	100	4	19
Indices	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
Elements	345	45	67	90	1000
Indices	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
Elements	45	67	89	69	35
Indices	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

i Store the given Elements in an array "A".

ii Display the 3rd element.

iii Display the Elements at index [2][0].

iv Update the Element 1000 to be 5000

v Display all Elements Using a for-loop statement

Program: Storing and displaying Array Elements in Multi-dimension Array

```
01 package AllPrograms;
02 public class AllProgramsClass {
03    public static void main(String[] args)
04    {
05          int data[ ][ ]=new int[ ][ ]{{100,234,100,4,19},{345,45,67,90,1000},
06                                       {45,67,89,69,35}};
07          //Displaying the third element
08          System.out.println("The third Element is "+data[0][2]);
09
10          //Displaying the Element at Index [2][0]
11          System.out.println("The Element at Index 2 is "+data[2][0]);
12
13          //Updating the Element 1000 to 5000
14          data[1][4]=5000;
15
16          //Displaying all Elements Using for-loop statement
17          System.out.println("The following are the Elements:");
18          for(int i=0;i<=2;i++)
19          {
20                System.out.println("Row Number "+(i+1));
21               for(int j=0;j<=4;j++)
22                {
```

```
23                  System.out.print("   "+data[i][j]);
24              }
25              System.out.println();
26          }
27      }
28  }
```

Output

```
The third element is 100
The Element at Index 2 is 45
The following are the Elements:

Row Number 1
  100   234   100   4   19

Row Number 2
  345   45   67   90   5000

Row Number 3
  45   67   89   69   35
```

12. Access Modifiers

In Java, access modifiers determine the accessibility or scope of a field, method, Constructor, or class. The access level of fields, constructors, methods, and classes can be modified by applying an access modifier to them. There are two types of modifiers in Java:

i. **Access modifiers**
ii. **Non-access modifiers**.

12.1 Access modifiers

These control the access level

12.1.1 Access modifiers for classes

Classes can use either *public* or *default* access modifiers

- **Public:** The class can be accessed by any other class.
- **Default:** The class can only be accessed by other classes within the same package. This is employed when a modifier is not specified.

12.1.2 Access modifiers for functions, constructors, blocks, and variables

Access modifiers restrict access to different elements of a class, including functions, constructors, blocks, and variables. Java supports four different forms of access specifiers:

- **Public:** The element can be accessed by all classes in our application when the public specifier is used. The public is a function access specifier, according to our given example.
- **Protected:** A protected modifier's access level is within and outside the package through the child class. If you do not make the child class, it cannot be accessed outside the package.
- **Default:** Java uses the default access specifier when no access specifier is specified in the method declaration. It is only visible from the same package.
- **Private:** The element becomes only accessible within the explicitly declared classes using a private access specifier.

Packages	Classes	Public	Protected	Default	Private
Same Package	Main class	Yes	Yes	Yes	Yes
	Sub class	Yes	Yes	Yes	No
	Non sub class	Yes	Yes	Yes	No
Different Package	Sub class of the Main class	Yes	Yes	No	No
	Non sub class	Yes	No	No	No

12.2 Non-access modifiers

They do not control access levels but provide other functionality.

12.2.1 Non-access modifiers for classes

Classes can use either *final* or *static* non-access modifiers

- **Final:** This class cannot be inherited by other classes.
- **Abstract:** This class cannot be used to create objects

12.2.2 Non-access modifiers for classes

- **Final:** It is impossible to modify or override attributes and methods.
- **Static:** The element belongs to the class, not an object.
- **Abstract:** It is used to declare methods in sub classes. The abstract indicates that the function has no body part e.g. *abstract void calculate ().* The body part is defined in the sub class derived from the main class.
- **Transient:** Attributes and methods are skipped when serializing the object containing them
- **Synchronized:** Methods can only be accessed by one thread at a time
- **Volatile:** The value of an attribute is not cached thread-locally, and is always read from the "main memory"

13. Functions/Methods

A Java method is a collection of statements grouped to perform a specific operation when called. For example, a section of a bank information system dealing with customers' deposits may contain several lines of code that communicate with the database, which transfers and

stores the desired amount of funds in the customer's account. All program statements that perform deposits can be grouped into a function. We usually write a method once and use it many times at different parts of our program by calling it rather than re-writing it. This mechanism allows the code to be reused in a computer program. It also provides an easy mechanism for modifying code because any modification to the function will affect all the blocks that utilize it.

13.1 Types of Method

There are two types of Methods in Java.

i. Predefined Methods
ii. User-defined Methods

13.1.1 Predefined Methods

These are methods already defined in Java class libraries; therefore, they are known as standard library methods. They are also called built-in methods because they are built in a programming language. These methods can be directly used at any time in the application by just calling them. Examples of Predefined Methods include max(), length(), equals(), compareTo(), sqrt(), print() and println() etc. The print() function is a **Print Stream** class method that prints the result to the console. The max() method in the **Math** class returns the greater of two values.

13.1.2 User-defined Method/Function

The term "user-defined method" refers to the Method the User or the Programmer writes in a program. Normally, the User-defined Method's name reflects the function's operations. For example, in a banking system, the function of depositing funds into customers' accounts is called **cashDeposite()**.

Syntax for declaring a Method/Function

```
access_specifier data_type method_name (list of arguments)
{
```

```
    //Function Body
}
```

Example:

```
public float average (float a, int b)
{
   //Codes for calculating the average goes here
}
```

Definition of terms from the function above

i **Access Specifier:** The access specifier or modifier specifies the method's access type. It determines the method's visibility.

ii **Return Type:** Define the type of value that the function will return. If the function returns a value, it is declared with a primitive data type such as int, float, or char; otherwise, it is declared void. The value returned by the function is the type of the declared datatype. ***float*** is the function return type according to our given example.

iii **Method Name:** It is the unique name of the function. The method name must relate to the task the function intends to perform. The ***average*** is the function name according to our given example.

iv **Parameter List (list of arguments):** Specifies the list and type of values that can be passed to the function. ***float a*** and ***int b*** are parameter declarations according to our given example. However, **a** and **b** are the list parameters (variables) that can store the value passed to the function.

v **Function Body:** It is where the codes to be executed when the function is called are written. This place may contain many lines of code based on the function operations.

13.1.2.1 Types of User-defined method

The Java user-defined methods can also be classified into the following types:

i. Static Method
ii. Instance Method

iii. Constructors
iv. Abstract Method
v. Factory Method

13.1.2.1.1 Static Methods

Static Methods are the ones that can be invoked/called without creating an object. The static methods can be called within the program without creating an object. The keyword **static** should be used when declaring a static **Method.**

The syntax for creating a static Method

```
access_specifier static data_type method_name (list of arguments)
{
    //Function Body
}
```

Example:

```
public static float average (float marks)
{
   //Codes for calculating the average goes here
}
```

static - Indicates that the **function** can be called without creating an **object.** Alternatively, we can say that no **object** owns this function.

- **Characteristics of static Method**
  i. It can access other static Methods and Variables without creating an object.
  ii. It cannot access an Instance Method or Variable directly. However, it can access the Instance Method or Variable using an object.

  Note: Since static methods are members of their class, these rules only apply to the class where the method was declared. This is because we don't need objects to reach them. It works like an instance function when called outside the class where it

was declared. To use it directly without an object, you must inherit the class where it was declared.

Question: Write a Java program that implements a static function concept that calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average input Marks using a static function

```
01 package AllPrograms;
02 import java.util.Scanner;
03 public class AllProgramsClass {
04
05 public static void average ()
06 {
07
08          // variable declaration
09          float math;
10          float geog;
11          float engl;
12          float avg;
13          //Creating a Scanner Object named as input
14          Scanner input=new Scanner(System.in);
15
16          //Getting Marks as Inputs from the Users
17          System.out.println("Enter Mathematics Marks ");
18          math=input.nextFloat();
19
20          System.out.println("Enter Geography Marks ");
21          geog=input.nextFloat();
22
23          System.out.println("Enter English Marks ");
24          engl=input.nextFloat();
25
26                  //Closing the scanner
27                   input.close();
28
29         //Calculate the average
30          avg=(math+geog+engl)/3;
31
32         //Displaying the output
33          System.out.println("The Average is "+avg);
34
```

```
35 }
36
37     public static void main(String[] args)
38     {
39         //Calling the static function
40         average ();
41     }
42 }
43
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
20
Enter English Marks
40
The Average is 23.333334
```

Question: Write a Java program that implements a static function concept with parameter passing that calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average input Marks using a static function with parameter passing

```
01  package AllPrograms;
02 import java.util.Scanner;
03 public class AllProgramsClass {
04
05       public static void average (float m, float g, float e)
06       {
07
08         float avg;
09         //Calculate the average
10         avg=(m+g+e)/3;
11
12         //Displaying the output
```

```
13          System.out.println("The Average is "+avg);
14
15        }
16
17        public static void main(String[] args)
18        {
19          // variable declaration
20          float math;
21          float geog;
22          float engl;
23
24          //Creating a Scanner Object named as input
25          Scanner input=new Scanner (System.in);
26
27          //Getting Marks as Inputs from the Users
28          System.out.println("Enter Mathematics Marks ");
29          math=input.nextFloat();
30
31          System.out.println("Enter Geography Marks ");
32          geog=input.nextFloat();
33
34          System.out.println("Enter English Marks ");
35          engl=input.nextFloat();
36
37                  //Closing the Scanner
38
39          input.close();
40         //Calling the static function
41          average(math,geog,engl);
42     }
43 }
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
10
Enter English Marks
10
The Average is 10.0
```

Note that: You will learn other types of Methods in the coming chapters

14. Java Classes and Objects

What is a class in Java?

A Class is an object constructor or a "blueprint" for creating objects. Or class is a group of variables and methods of different data types.

What is an object in Java?

An object is any real-world entity that can be converted into a computer program. The object can be physical or logical (tangible and intangible). An example of an intangible object is the banking system. Examples of tangible objects include a chair, bike, marker, pen, table, and car. We can also define an o**bject** as an instance of a **class.**

An object has three characteristics:

State: These are properties of an object. For example, a car's object state will be color, current speed, number of wheels etc. In programming, object states are the variables.
Behaviour: These are all actions that an object can perform. For example, for a car, object behaviour will be slow down, speed up and turn around. In programming, objects' behaviours are the functions.
Identity: It is a unique name of an object which differentiates it from the rest
The syntax for declaring an object
Classname objectname=new Classname();

Example of Class Student

```
Class Student
{
        //Variables declaration
        static/non static Variable 1;
        static/non static Variable 2;
```

```
    //Functions declaration
    static/non function 1(parameter 1, parameter 2,….. parameter n)
    {
      //Local variables
       Variable 1;
       Variable 2
    }
    static/non function 2(parameter 1, parameter 2,…..parameter n)
    {
      //Local variables
       Variable 1;
       Variable 2
    }

    public static void main (String args[ ] )
    {
        //The declarations of object "c"

         Student c=new Student ();

    }
}
```

Note the following:

1) Declaring Objects is also called instantiating a class.
2) Refers to properties of static and instance variables and static functions in the previous chapters

14.1 Instance Method

This method is declared in the class without using the static keyword

- **Characteristics of Instance Method**
  i. It can access static and non-static methods and variables without creating an object.
  ii. It can access other instance methods directly without creating an object.

iii. It can be accessed by static methods using an object.

The syntax for creating an instance Method

access_specifier data_typemethod name (list of arguments)

{

//Function Body

}

Example:

public float average (float marks)

{

//Codes for calculating the average goes here

}

Example: Write a Java program that implements calling instance function concept using object without parameter passing that calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average by calling an instance function using object without parameter passing

```
01  package AllPrograms;
02  import java.util.Scanner;
03  public class AllProgramsClass {
04
05          public void average ()
06          {
07
08           // variable declaration
09                  float math;
10                  float geog;
11                  float engl;
12                  float avg;
13
```

```
14                //Creating a Scanner Object named as input
15                Scanner input=new Scanner (System.in);
16
17                //Getting Marks as Inputs from the Users
18                System.out.println("Enter Mathematics Marks ");
19                math=input.nextFloat();
20
21                System.out.println("Enter Geography Marks ");
22                geog=input.nextFloat();
23
24                System.out.println("Enter English Marks ");
25                engl=input.nextFloat();
26
27              //Closing the Scanner
28
29                input.close();
30
31               //Calculate the average
32                avg=(math+geog+engl)/3;
33
34               //Displaying the output
35                System.out.println("The Average is "+avg);
36
37        }
38
39        public static void main(String[] args)
40        {
41
42
43                //Creating an Object named callavg
44                AllProgramsClass callavg=new AllProgramsClass();
45
46                callavg.average();
47        }
48  }
49
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
```

```
10
Enter English Marks
10
The Average is 10.0
```

Question: Write a Java program that implements the calling instance function concept using an object with parameter passing that calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average by calling an instance function using an object with parameter passing

```
01  package AllPrograms;
02  import java.util.Scanner;
03  public class AllProgramsClass {
04
05      public void average (float m, float g, float e)
06      {
07
08              float avg;
09              //Calculate the average
10              avg=(m+g+e)/3;
11
12              //Displaying the output
13              System.out.println("The Average is "+avg);
14
15       }
16
17       public static void main(String[] args)
18       {
19              // variable declaration
20              float math;
21              float geog;
22              float engl;
23
24              //Creating a Scanner Object named as input
25              Scanner input=new Scanner (System.in);
26
```

```
27                //Getting Marks as Inputs from the Users
28                System.out.println("Enter Mathematics Marks ");
29                math=input.nextFloat();
30
31                System.out.println("Enter Geography Marks ");
32                geog=input.nextFloat();
33
34                System.out.println("Enter English Marks ");
35                engl=input.nextFloat();
36
37               //Closing the Scanner
38
39                input.close();
40               //Calling the static function
41
42                //Creating an Object named callavg
43                AllProgramsClass callavg=new AllProgramsClass();
44
45                //Creating an Instance function
46                callavg.average(math,geog,engl);
47     }
48  }
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
10
Enter English Marks
10
The Average is 10.0
```

In actual time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class. We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

Example: A program implementing the main function in another class

```
01 package AllPrograms;
02 import java.util.Scanner;
03 class studentResults{
04     public void average ()
05     {
06
07       // variable declaration
08               float math;
09               float geog;
10               float engl;
11               float avg;
12
13               //Creating a Scanner Object named as input
14               Scanner input=new Scanner (System.in);
15
16               //Getting Marks as Inputs from the Users
17               System.out.println("Enter Mathematics Marks ");
18               math=input.nextFloat();
19
20               System.out.println("Enter Geography Marks ");
21               geog=input.nextFloat();
22
23               System.out.println("Enter English Marks ");
24               engl=input.nextFloat();
25
26                //Closing the Scanner
27
28               input.close();
29
30               //Calculate the average
31               avg=(math+geog+engl)/3;
32
33                //Displaying the output
34               System.out.println("The Average is "+avg);
35
36      }
37
38 }
39 public class AllProgramsClass {
40
41       public static void main (String [] args)
42       {
```

```
43
44                  //Creating an Object named callavg
45          studentResults callavg=new studentResults();
46
47                  callavg.average();
48          }
49  }
50
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
10
Enter English Marks
10
The Average is 10.0
```

14.2 Constructors Methods

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. When calling Constructor, memory for the object is allocated in the memory.

Rules for creating Java constructor

There are two rules defined for the Constructor.

1. The 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

Question: Write a Java program that implements Constructor without parameter passing. It calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average by using Constructor without parameter passing

```
01 package AllPrograms;
02 import java.util.Scanner;
03 public class AllProgramsClass {
04
05       public AllProgramsClass()
06       {
07
08                // variable declaration
09                float math;
10                float geog;
11                float engl;
12                float avg;
13
14                //Creating a Scanner Object named as input
15                Scanner input=new Scanner (System.in);
16
17                //Getting Marks as Inputs from the Users
18                System.out.println("Enter Mathematics Marks ");
19                math=input.nextFloat();
20
21                System.out.println("Enter Geography Marks ");
22                geog=input.nextFloat();
23
24                System.out.println("Enter English Marks ");
25                engl=input.nextFloat();
26
27                //Closing the Scanner
28
29                input.close();
30
31                //Calculate the average
32                avg=(math+geog+engl)/3;
33
34                //Displaying the output
35                System.out.println("The Average is "+avg);
36
37       }
38
39       public static void main(String[] args)
```

```
40      {
41
42              //Creating an Object named
43              AllProgramsClass callavg=new AllProgramsClass();
44
45      }
46 }
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
10
Enter English Marks
10
The Average is 10.0
```

Question: Write a Java program that implements Constructor with parameter passing that calculates the average of three input Marks, such as Mathematics, Geography, and English, and outputs the results on the screen. Note that your program should receive Marks as input from the user.

Program: To calculate the average by using a Constructor with parameter passing

```
01  package AllPrograms;
02  import java.util.Scanner;
03  public class AllProgramsClass {
04
05      public AllProgramsClass(float a,float b, float c)
06      {
07
08              float avg;
09              //Calculate the average
10              avg=(a+b+c)/3;
11
12              //Displaying the output
13              System.out.println("The Average is "+avg);
14
15      }
```

```
16
17          public static void main(String[] args)
18          {
19
20                  // variable declaration
21                          float math;
22                          float geog;
23                          float engl;
24
25                          //Creating a Scanner Object named as input
26                          Scanner input=new Scanner (System.in);
27
28                          //Getting Marks as Inputs from the Users
29                          System.out.println("Enter Mathematics Marks ");
30                          math=input.nextFloat();
31
32                          System.out.println("Enter Geography Marks ");
33                          geog=input.nextFloat();
34
35                          System.out.println("Enter English Marks ");
36                          engl=input.nextFloat();
37
38                          //Closing the Scanner
39
40                          input.close();
41
42                         //Creating an Object named
43                         AllProgramsClass callavg=new AllProgramsClass(math,geog,engl);
44
45          }
46  }
47
```

Output

```
Enter Mathematics Marks
10
Enter Geography Marks
10
Enter English Marks
10
The Average is 10.0
```

15. Encapsulation

The process of integrating data (variables) and the code that manipulates them (methods) into a single unit is called encapsulation in Java. Variables within a class are encapsulated so that only the class's methods can access them and others cannot. Java enhances data security and code maintainability by encapsulating data. Encapsulation is achieved when the variables and methods are declared **private.**

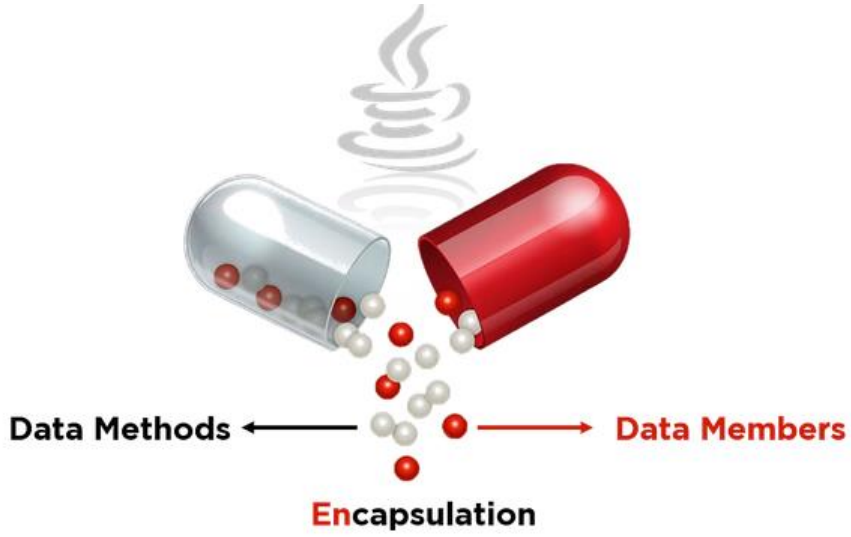


Figure: .....

Syntax for declaring encapsulated variables and functions

```
private data_type variable_name;
```

```
private access_specifier data_type method_name (list of arguments)
{
    //Function Body
}
```

However, external classes may need to update encapsulated variables in some situations. This can be achieved by defining the public methods for updating and displaying data.

Question: Given the declaration "*private String studentName="Abraham*";" of a class "studentDetails" . Write a Java program to update "*Abraham*" to "*Juma*" using another class known "*AllProgramsClass*".

Program: Demonstrating updating encapsulated variable

```
01  package AllPrograms;
02  class studentDetails
03  {
04     private String studentName="Abraham";
05
06     ublic void settingName(String newName)
07     {
08        this.studentName=newName;
09     }
10
11     public String displayingName( )
12     {
13        return this.studentName;
14     }
15  }
16  public class AllProgramsClass
17  {
18     public static void main(String[] args)
19      {
20
21        studentDetails obj=new studentDetails ();
22
23        //Displaying Student Name before Update
24        System.out.println("Student Name before Update is "+obj.displayingName());
25
26        //Update the Student Name to Juma
27        obj.settingName("Juma");
```

```
28
29          //Displaying Student Name after Update
30          System.out.println("Student Name After Update is "+obj.displayingName());
31      }
32  }
```

16. Java Abstraction

In Java, abstraction keeps a code's implementation details hidden and provides the user with only the information they need to know. Abstraction in Java provides the user with essential information while preventing the user from observing complex code implementations. Java offers many built-in abstractions and limited tools for creating custom abstractions. An example of abstraction in the real world is withdrawing currency from an ATM; it is sufficient to interact with the screen without being concerned with the underlying process.

Abstraction in Java can be achieved in two ways.

i. Using abstract classes: Abstraction utilizing abstract classes does not achieve 100% abstraction because it may also have none abstract methods and constructors apart from the abstract methods.

ii. Using interfaces: Abstraction utilizing interface achieves 100% abstraction because it contains only abstract methods.

16.1 Using Abstract Class

An abstract class is a class that has been declared abstract. It might include both abstract and non-abstract methods. For its methods to be implemented first, they have to be extended. The abstract class cannot be used to create an object.

16.1.1 Abstract Method

An abstract class in Java is a class that is declared with the abstract keyword. It can contain both abstract and non-abstract methods. Non-abstract methods possess a body, whereas abstract methods do not.

- **Characteristics of Abstract Class**

  i. An abstract class must be declared using the abstract keyword.

ii. It might include both abstract and non-abstract Methods.
iii. we cannot create objects of abstract classes
iv. It can include constructors and static methods.
v. It can have final methods that prevent the subclass from changing the method's body.
vi. Abstract methods cannot declare static
vii. Abstract methods can only set visible modifiers public, protected and default
viii. Non-abstract methods may employ public, protected, private, and default modifiers. However, private modifiers restrict access to the function outside of the class.
ix. Variables can use public, protected, private and default. Also, it can have instance and static methods

Syntax for declaring an abstract class

```
abstract class class_name {

  abstract datatype method_name ();

}
```

Question: The College of Business Education (CBE) has four campuses in various parts of Tanzania. Write a Java program that describes CBE using an abstract class.

```
01  package AllPrograms;
02
03  abstract class CBE{
04
05      abstract void aboutCBE();
06
07      void rectorMessage()
08      {
09          System.out.println("Welcome to CBE");
10      }
11  }
12  public class testVariable extends CBE{
13      void aboutCBE()
```

```
14        {
15                System.out.println("CBE has Campuses in Dar es salaam, Dodoma, Mbeya
16                 and Mwanza");
17        }
18
19        public static void main(String args[])
20        {
21                testVariable a=new testVariable();
22                a.aboutCBE();
23               a.rectorMessage();
24        }
25    }
26
```

Output

```
CBE has Campuses in Dar es salaam, Dodoma, Mbeya and Mwanza
Welcome to CBE
```

16.1.2 Using Interface

An interface is a completely abstract class. It contains only a collection of abstract methods (methods without a body). Non-abstract methods and constructors are not allowed in an interface, so they provide 100% abstraction. Similarly to the abstract class, we cannot create objects of the interface class. Apart from being extended as an abstract class, interfaces are implemented for them to be used. Therefore, the keyword 'implements' is used to implement interfaces.

- **Characteristics of interface Class**
  i. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
  ii. All abstract, default, and static methods in an interface are implicitly public, so you can omit the public modifier. In addition, an interface can contain constant declarations.
  iii. A class may implement more than one interface at a time.

iv. An interface can extend another interface in the same manner that a class can.

16.1.2.1 Implementation of the Method

By default, all methods in an interface are abstract and must be implemented by any class that implements the interface. Starting from Java 8, interfaces are capable of containing default and static methods that have concrete implementations. Starting from Java 9, interfaces are now capable of containing private methods.

16.1.2.2 Variables

By default, variables declared in an interface are public, static, and final, which means they are constants.

Syntax for declaring an interface class

```
interface class_name {

 datatype method_name1 ();

datatype method_name2 ();

}
```

Question: The College of Business Education (CBE) has four campuses in various parts of Tanzania. Write a Java program that describes CBE using an inteface class.

```
01  package GuiPackage;
02
03  interface CBE{
04
05      abstract void aboutCBE();
06      abstract void rectorMessage();
07
08  }
```

```
09
10  public class testVariable implements CBE{
11
12      public void rectorMessage()
13      {
14              System.out.println("Welcome to CBE");
15      }
16      public void aboutCBE()
17      {
18              System.out.println("CBE has Campuses in Dar es salaam, Dodoma,
19              Mbeya and Mwanza");
20      }
21
22      public static void main(String args[])
23      {
24              testVariable a=new testVariable();
25              a.aboutCBE();
26              a.rectorMessage();
27      }
28  }
```

17. Nested Classes

It's possible to define a class inside another class in Java. This type of class is known as a nested class and is demonstrated here.

Nested classes are classified into two types: **Syntax for creating nested classes**

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
    static class StaticNestedClass {
        ...
    }
}
```

Note: A nested class may be declared **private, public, protected, or package private**. However, the outer classes can only be set to **public or package private**. A nested class is a member of its parent class. Non-static nested classes (inner classes) can access other members of the enclosing class, even if declared private. Static nested classes cannot access other members of the enclosing class.

Why use nested classes?

1. **It is a way of logically grouping classes that are only used in one place**: If a class is only beneficial to one other class, it makes sense to embed it in that class and keep the two together. Nesting such "helper classes" streamlines the package.
2. **It increases encapsulation**: Consider two top-level classes, A and B, in which B requires access to members of A that would otherwise be considered private. By hiding Class B within Class A, it is possible to declare the members of Class A to be private, allowing Class B to access them. Additionally, B itself can be hidden from the rest of the world.
3. **It can lead to more readable and maintainable code**: When minor classes are nested within top-level classes, the code is brought closer to the location where it is being utilized.

17.1 Inner Classes

The Inner Class is a non-static class enclosed by the Outer Class

Syntax for Creating Inner Classes

```
class OuterClass {
   ...
   class InnerClass {
      ...
   }
}
```

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();

Properties of the Inner Class

1. The inner class can directly access the methods and variables of its enclosing class regardless of whether they are declared static.
2. Inner-class objects can only access instances of their classes
3. Instances of the Outer Class cannot access instances of their Inner Classes
4. The inner class can access instances of other inner Classes through inheritance. Also, we can say that one inner class can inherit the properties of another.

Example: A program implementing Inner Classes

```
01  package AllPrograms;
02  import java.util.Scanner;
03
04  public class AllProgramsClass {
05
06   int instancevar1=10;
07
08   class instanceclass1
09   {
10          int variableclass1=10;
11          void fnclass1()
12          {
13                  //Accessing Instance Variable 1
14                  System.out.println("Instance Variable 1 is "+instancevar1);
15          }
16   }
17   class instanceclass2 extends instanceclass1{
18
19          void fnclass2()
20          {
21                  //Calling Instance variable of the Inner Class instanceclass1
22                  fnclass1();
```

```
23            }
24    }
25      public static void main(String[] args) {
26
27            //Creating Object of the Inner Class 2
28            AllProgramsClass obj1=new AllProgramsClass();
29            AllProgramsClass.instanceclass2 obj2=obj1.new instanceclass2();
30            obj2.fnclass2();
31      }
32  }
33
```

Output

```
Instance Variable 1 is 10
```

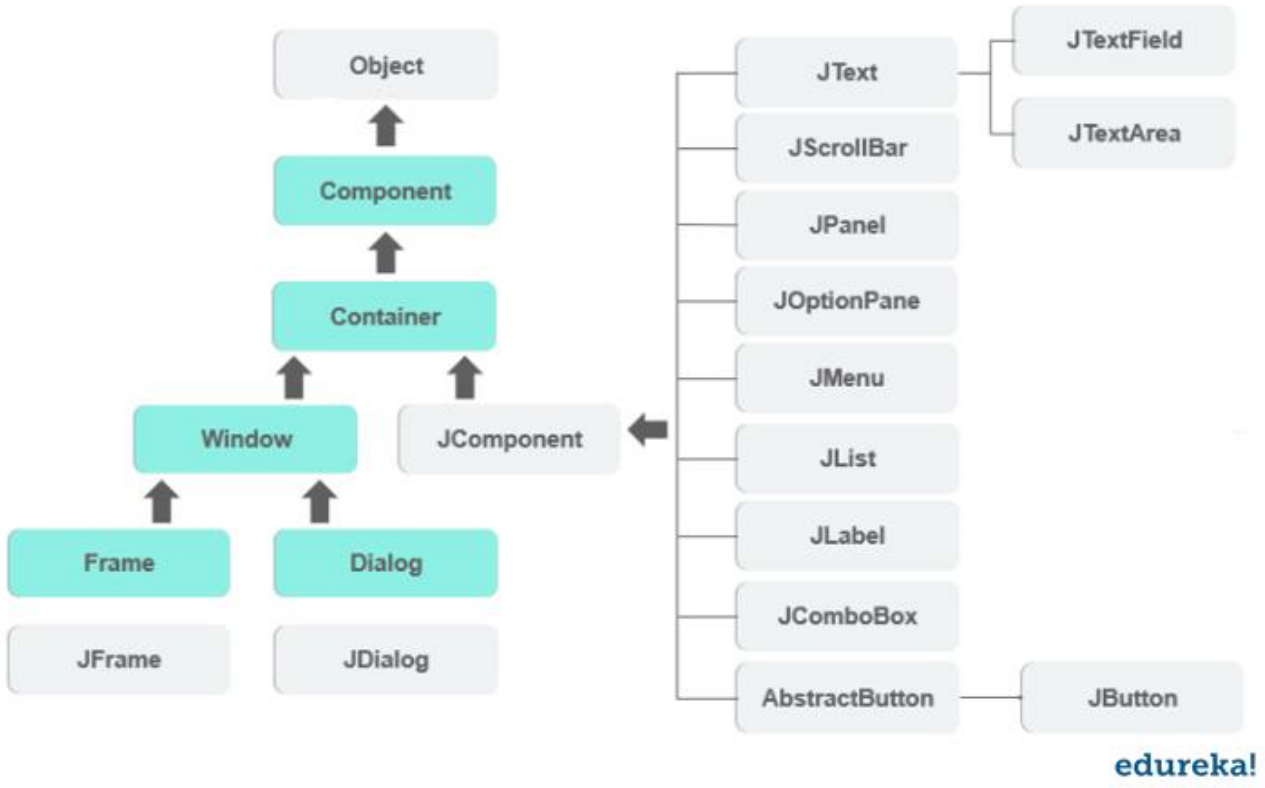
18. JAVA SWING

Swing in Java is a lightweight and platform-independent component of the Java foundation class (JFC) used to create window-based applications. JFC includes tools for developing graphical user interfaces (GUIs) and providing rich graphics functionality and interactivity to Java applications. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser, etc. When these components are combined, they form a graphical user interface.

Java Swing is built on the AWT API and written entirely in Java. Building applications using Java Swings is easier because we already have GUI components such as buttons and checkboxes. This is useful because we don't have to start from scratch.

In computing, lightweight software, sometimes called lightweight programs and lightweight applications, is a computer program that is designed to have a tiny memory footprint (RAM consumption) and low CPU usage, resulting in minimal usage of system resources. This article will review the concepts involved in building applications with Swing in Java.

18.1 Java Swing Class Hierarchy



Explanation: All components in Swing, such as JButton, JComboBox, JList, and JLabel, are derived from the JComponent class and can be added to container classes. Containers are window-like frames and dialog boxes. Essential swing components are the foundation of any GUI application. Methods like setLayout override the default layout in each container. All components in Java Swing are JComponent, which can be added to container classes.

18.2 Container Class

A container class refers to any class that contains other components. A minimum of one container object is required while developing GUI apps. There are three(3) types of Java Swing containers.

1. **Frame**: It is a fully functioning window with its title and icons.
2. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components into a window.

3. **Dialog**: It functions as a pop-up window when a message has to be displayed. It is not a fully functional window, like the Frame.

https://www.youtube.com/watch?v=kVOym20oTR4

https://www.youtube.com/watch?v=-p2eGZ7_Lr4&t=262s

https://www.youtube.com/watch?v=Mp7hkPQjKeM