Subject Name: Java Programming
Subject Code:28541
Computer Science & Technology-4th Semester

BulBul Ahamed
Chief Instructor (Computer)
Mymensingh Polytechnic Institute.

What is Java?

Java is a popular programming language, created in 1995. It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world
- It has a large demand in the current job market
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

Java Install

Some PCs might have Java already installed.

To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

```
C:\Users\Your Name>java –version
```

If Java is installed, you will see something like this (depending on version):

java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)

Java Syntax

```
public class Main {
public static void main(String[] args) {
System.out.println("Hello World");
}
}
```

Every line of code that runs in Java must be inside a class. In our example, we named the class **Main**. A class should always start with an uppercase first letter.

Note: Java is case-sensitive: "MyClass" and "myclass" has different meaning.

Java Output / Print

```
System.out.println("Hello World!");
System.out.println("I am learning Java.");
System.out.println("It is awesome!");
```

Print Numbers

```
System.out.println(3);
System.out.println(358);
System.out.println(50000);
```

Java Comments

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

Single-line Comments

Single-line comments start with two forward slashes (//).

Example

```
// This is a comment
```

Java Multi-line Comments

```
/* The code below will print the words Hello World to the screen, and it is amazing */
```

Java Variables

Variables are containers for storing data values.

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

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

type variableName = value;

Example

Create a variable called **name** of type String and assign it the value "**John**":

```
String name = "John";
System.out.println(name);
```

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

```
String name = "John";
System.out.println("Hello " + name);
```

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

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

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `byte`, `short`, `int` and `long`. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: `float` and `double`.

```
byte myNum = 100; System.out.println(myNum);
short myNum = 5000; System.out.println(myNum);
int myNum = 100000; System.out.println(myNum);
long myNum = 15000000000L;
System.out.println(myNum);
float myNum = 5.75f; System.out.println(myNum);
double myNum = 19.99d; System.out.println(myNum);
```

Java Operators

Operators are used to perform operations on variables and values.
In the example below, we use the + **operator** to add together two values:

```
int sum1 = 100 + 50; // 150 (100 + 50)
int sum2 = sum1 + 250; // 400 (150 + 250)
int sum3 = sum2 + sum2; // 800 (400 + 400)
```

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Assignment operators are used to assign values to variables.

assignment operator (=) to assign the value **10** to a variable called **x**:

```
int x = 10;
```

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Comparison operators are used to compare two values (or variables).

```
int x = 5; int y = 3;
System.out.println(x > y); // returns true, because 5 is higher than 3
```

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Strings

Strings are used for storing text.

```
String greeting = "Hello";
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```

Conditions and If Statements

Less than: a < b
Less than or equal to: a <= b
Greater than: a > b
Greater than or equal to: a >= b
Equal to a == b
Not Equal to: a != b

```
if (condition) {
// block of code to be executed if the condition is true
}
if (20 > 18) {
System.out.println("20 is greater than 18");
}
```

else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

```
if (condition) { // block of code to be executed if the condition is true }
else { // block of code to be executed if the condition is false }
```

Example:

```
int time = 20;
if (time < 18) {
System.out.println("Good day."); }
else {
System.out.println("Good evening."); } // Outputs "Good evening."
```

else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Example:

```
int time = 22;
if (time < 10) {
System.out.println("Good morning."); }
else if (time < 18) {
System.out.println("Good day."); }
else {
System.out.println("Good evening."); } // Outputs "Good evening."
```

Switch Statements

Instead of writing **many** if..else statements, you can use the switch statement.

Example:

```
int day = 4;
switch (day) {
case 1: System.out.println("Monday");
break;
case 2: System.out.println("Tuesday");
break;
case 3: System.out.println("Wednesday");
break;
case 4: System.out.println("Thursday");
break;
case 5: System.out.println("Friday");
break;
case 6: System.out.println("Saturday");
break;
case 7: System.out.println("Sunday");
break;
} // Outputs "Thursday" (day 4)
```

Loops

Loops can execute a block of code as long as a specified condition is reached.
Loops are handy because they save time, reduce errors, and they make code more readable.

While Loop

The while loop loops through a block of code as long as a specified condition is true:

Example:

```
int i = 0;
while (i < 5) {
System.out.println(i);
i++;
}
```

For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Example:

```
for (int i = 0; i < 5; i++)
{
System.out.println(i);
}
```

Nested Loops

It is also possible to place a loop inside another loop. This is called a **nested loop**.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example:

```
// Outer loop
for (int i = 1; i <= 2; i++)
{
System.out.println("Outer: " + i); // Executes 2 times
// Inner loop
for (int j = 1; j <= 3; j++)
{
System.out.println(" Inner: " + j); // Executes 6 times (2 * 3)
}
}
```

For-Each Loop

There is also a "**for-each**" loop, which is used exclusively to loop through elements in an **array**:

Example:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars)
{
System.out.println(i);
}
```

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

Example:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

int[] myNum = {10, 20, 30, 40};

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Methods

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Example:

```
public class Main {
static void myMethod()
{
// code to be executed
}
}
```

Method Overloading

With **method overloading**, multiple methods can have the same name with different parameters:

Example:

```
static int plusMethodInt(int x, int y)
{
return x + y;
}
static double plusMethodDouble(double x, double y)
{
return x + y;
}
public static void main(String[] args){
int myNum1 = plusMethodInt(8, 5);
double myNum2 = plusMethodDouble(4.3, 6.26);
System.out.println("int: " + myNum1);
System.out.println("double: " + myNum2);
}
```

What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Classes/Objects

ava is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

Example:

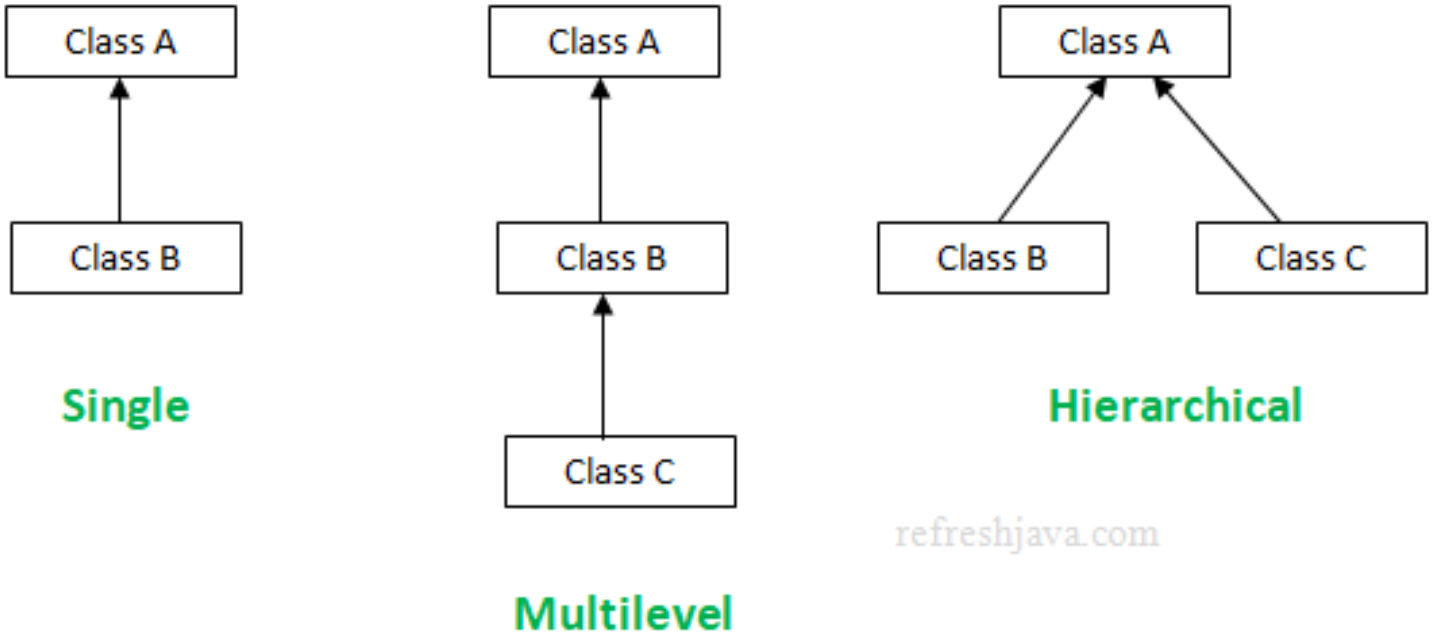
```
public class Main {
int x = 5;
public static void main(String[] args)
{
Main myObj = new Main();
System.out.println(myObj.x);
}
}
```

Inheritance

In general the meaning of inheritance is something that you got from your predecessor or parent, the same applies with java inheritance as well. Inheritance in java is a mechanism by which one class is allowed to inherit the features(fields and methods) of another class.

Different Types of Inheritance in Java

- Single inheritance
- Multilevel inheritance
- Hierarchical inheritance



Single inheritance

Example:

```
class student
{
int roll;
String name;
void getdata()
{
roll = 101;
name = "Karim";
}
}
class display extends student
{
void display()
{
System.out.println("Roll is :" + roll);
System.out.println("Name is :" + name);
}
}
class single_inheritance
{
public static void main(String[] args)
{
display d = new display();
d.getdata();
d.display();
}
}
```

Multilevel Inheritace

Example:

```
class student
{
int roll;
String name;
float mark;
}
class exam extends student
{
void getdata()
{
roll=101;
name="Karim";
}
}
class result extends exam
{
void getmark()
{
mark=50.55f;
}
void display()
{
System.out.println("Roll is "+roll);
System.out.println("Name is "+name);
System.out.println("Mark is "+mark);
}
}
class multilevel_inheritace
{
public static void main(String args[])
{
result r=new result();
r.getdata();
r.getmark();
r.display();
}
}
```

Hierarchical Inheritace

Example:

```
class student
{
int roll;
String name;
float mark;
}
class exam extends student
{
void getdata()
{
roll=101;
name="Karim";
System.out.println("Roll is "+roll);
System.out.println("Name is "+name);
}
}
class result extends student
{
void getmark()
{
mark=50.55f;
System.out.println("Mark is "+mark);
}
}
class p8
{
public static void main(String args[])
{
student st=new student();
exam e=new exam();
result r=new result();
e.getdata();
r.getmark();
}
}
```

Interface

In java programming language an interface is a reference type, similar as class, that can contain only constants, method declaration

Syntax:

```
interface interfaceName {
// constant declarations
// Method signatures
}
```

Example :

```
interface MyInterface {
int id = 20; void print();
public int calculateArea();
}
```

Thank You