Data Types, Variables and Operators

Nimesha Hewawasam
nimesha.h@nsbm.ac.lk

Start from here

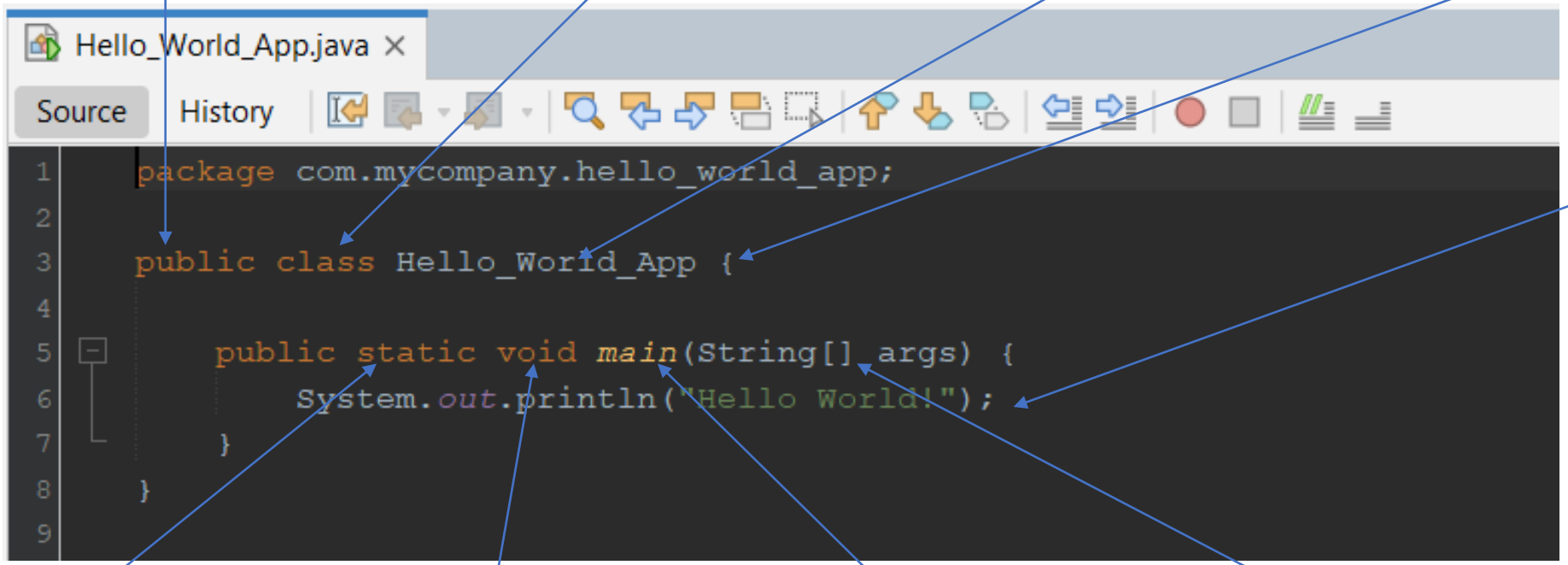
Introduction to class and main method

Access modifier — means the class or method is **visible to all** (any class can access it)

Keyword used to **declare a class** (a blueprint for objects)

Name of the class. By convention, should start with a **Capital Letter**

Braces - enclose the **body of the class or method**



Print statement: Print text to the console (no newline)

Means this method belongs to the **class itself**, not an object

Return type — means this method **does not return any value**

The **entry point** of any Java application (method that runs first)

(String [] arg) : Accept **command-line arguments** (an array of Strings named arg)

Part I

Data Types

What is Data Type?

- **Definition**: A data type specifies the kind of value a variable can hold.
- Java is **statically typed**: Every variable must have a declared type.
- **Purpose**:
  - **Memory efficiency:** Choosing the right type (byte vs int) saves memory
  - **Performance:** Proper types reduce runtime errors.
  - **Code Clarity:** Explicit typing makes code more readable.
- **Two Main Categories**:
  - **Primitive Data Types**: Directly holds values (e.g. int, char, boolean)
  - **Reference (Non-Primitive) Data Types**: Refers to objects (e.g. String, arrays, classes)
- Data types ensure **type safety**, help in **performance optimization**, and guide **operations** you can perform on variables.

Primitive Data Types

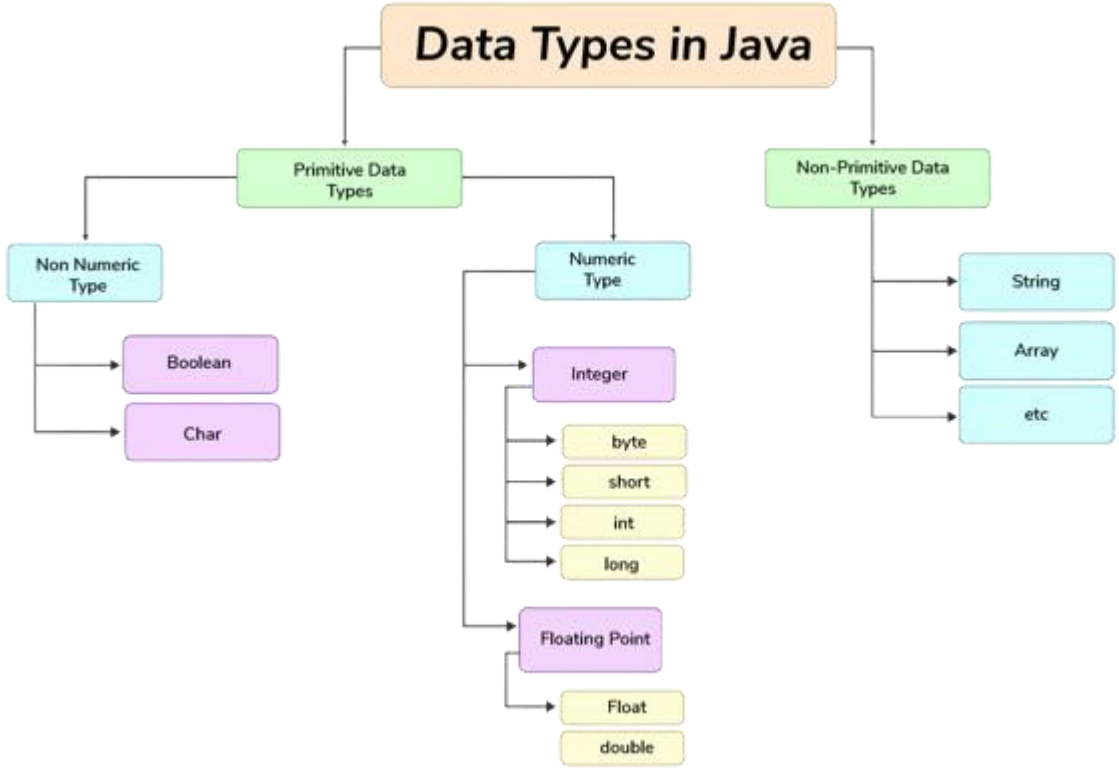
Category	Data Type	Size	Example	Description
Integers	byte	8 bits	byte b = 10;	Very small int values
	short	16 bits	short s = 500;	Smaller int range
	int	32 bits	int age = 25;	Whole numbers
	long	64 bits	long distance = 100000L;	Large whole number
Real Numbers	float	32 bits	float temp = 36.6f;	Decimal (end with f)
	double	64 bits	double pi = 3.14;	More precise decimal
Character	char	16 bits	char grade = 'A';	Single character
Boolean	boolean	1 bits	boolean isTrue = true;	True or False

Reference (Non- Primitive) Data Types

- Class:
  - User-defined blueprint or prototype from which objects are created
- Object
  - basic unit of Object-Oriented Programming and represents real-life entities
- Interface
  - A contract that defines what a class must do, but not how it does it.
    - Ex: Imagine an interface called *RemoteControl*, Any device that wants to act like a remote (TV, AC, Drone) must implement functions like *turnOn()*, *turnoff()*. But how each device does it may differ.
- Array
  - Collection of similar data elements stored under one name in contiguous memory locations.

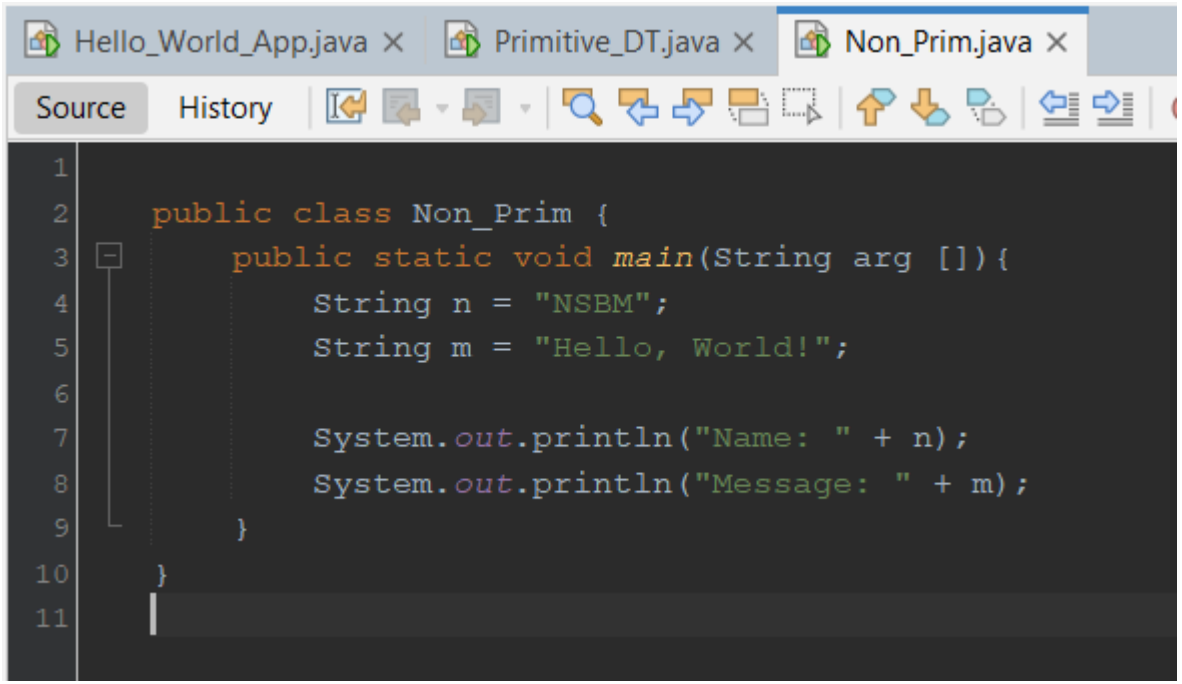
Primitive Vs Non-Primitive DTs

Aspect	Primitive	Non-Primitive
Memory	Stored on the stack	Stored on the heap
Speed	Primitive data types are faster	Non-primitive data types are slower
Example	int x = 5;	String s = “NSBM";



Reference (Non- Primitive) Data Types

- Contain a memory address of variable values because the reference types won't store the variable value directly in memory
  - String:
    - Difference between a **character array** and a **string** in Java is, string is designed to hold a sequence of characters in a single variable.
    - A character array is a collection of separate char-type entities.

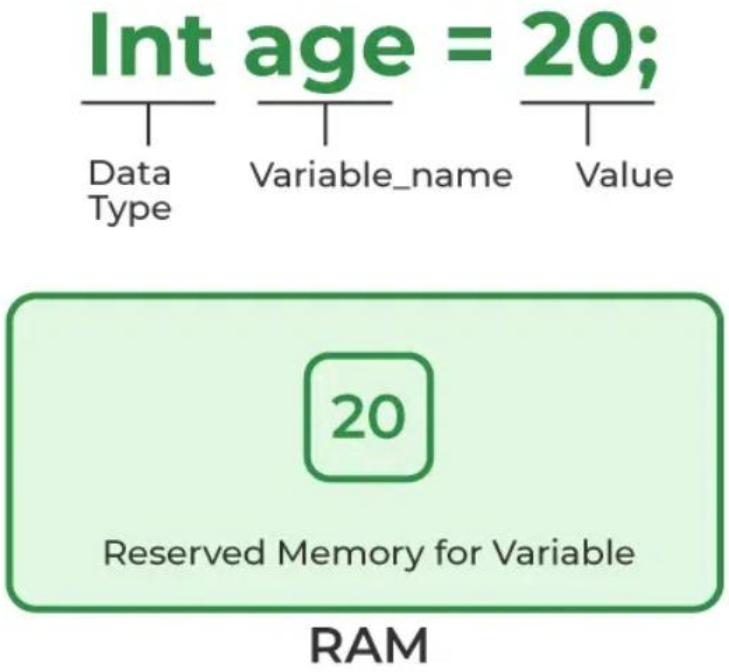


Part II

Variables

What is a Variable?

- **Definition**: A container that holds a value which can change during program execution.
- **Components**:
  - Data type - Defines the kind of data stored (e.g., int, String, float).
  - Variable name - A unique identifier following Java naming rules (camelCase).
  - Value - The actual data assigned to the variable.



Types of Java Variables

- Local Variables
  - Declared **inside methods, constructors, or blocks**
  - **Created at the time of declaration** and **destroyed when the function completed** its execution.
  - Exists only **within the block**
  - Need to **initialize** a local variable **before using it** within the scope.
- Instance Variables
  - Non-static variables
  - Declared in a class **outside of any method, constructor, or block**.
  - **Created** when an **object of the class is created** and **destroyed when the object is destroyed**.
  - We may use **access specifiers** for instance variables. If **we do not specify any access specifier**, then the **default access specifier will be used**.
  - **Initialization** is **not mandatory**. Its **default value is dependent on the data type of variable**.
    - Ex; For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for Wrapper classes like *Integer* it is *null*, *etc.*
  - Can be accessed only by creating objects.

Types of Java Variables

- Static Variables
  - Aka class variables
  - Declared similarly to instance variables. **Difference** is that static variables are declared using the **static** keyword **within a class outside of any method, constructor, or block**.
  - We can only have **one copy of a static variable per class**, irrespective of how many objects we create.
  - **Created** at the **start of program execution** and **destroyed automatically when execution ends.**
  - **Initialization** of a static variable is **not mandatory**. Its default value is **dependent on the data type of variable**.
    - Ex: for *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for *Wrapper classes* like *Integer* it is *null*, etc.
  - **Cannot declared locally inside an instance method.**

Instance vs Static Variables

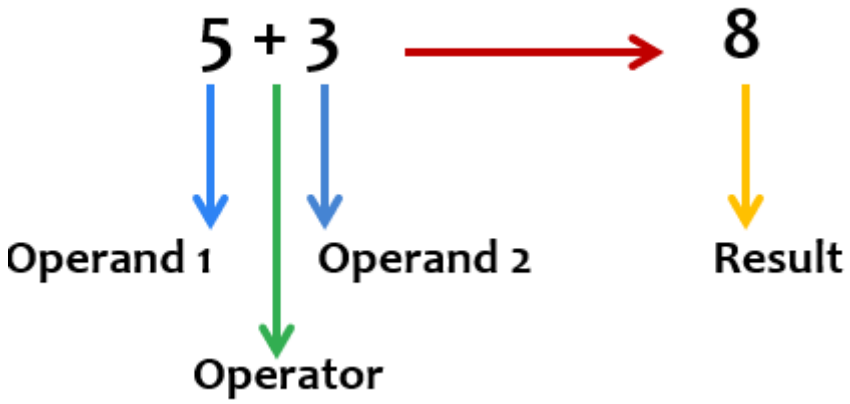
Feature	Instance Variable	Static Variable
Copies per object	Each object has its own copy	Only one copy per class (shared by all objects)
Effect of modification	Changes affect only the object	Changes affect all objects
Access method	Accessed using object reference	Accessed using class name
Lifetime	Created when object is created, destroyed with object	Created when program starts, destroyed when program ends
Syntax example	int a;	static int a;

Part III

Java Operators

Java Operators

- Special symbols that perform operations on variables or values.
- Allow you to manipulate data efficiently
- Types of Operators in Java
  - Arithmetic Operators
  - Unary Operators
  - Assignment Operator
  - Relational Operators
  - Logical Operators
  - Ternary Operator
  - Bitwise Operators
  - Shift Operators
  - instance of operator



Arithmetic Operators

Hello_World_App.java | Primitive_DT.java | Non_Prim.java | VariableTypes.java | JavaOperators.java

Source | History

```java
public class JavaOperators {
    public static void main(String args[]){
        //Arithmatic operators on integer
        int num1 = 10;
        int num2 = 2;

        //Arithmatic operators on strings
        String word1 = "15";
        String word2 = "25";

        //Convert strings into integers
        int num3 = Integer.parseInt(word1);
        int num4 = Integer.parseInt(word2);

        System.out.println("num1 + num2 = " + (num1 + num2));
        System.out.println("num1 - num2 = " + (num1 - num2));
        System.out.println("num1 * num2 = " + (num1 * num2));
        System.out.println("num1 / num2 = " + (num1 / num2));
        System.out.println("num1 % num2 = " + (num1 % num2));
        System.out.println("num3 + num4 = " + (num3 + num4));
    }
}
```

```
--- exec:3.1.0:exec (default-cli) @ Hel
num1 + num2 = 12
num1 - num2 = 8
num1 * num2 = 20
num1 / num2 = 5
num1 % num2 = 0
num3 + num4 = 40
------------------------------------------
BUILD SUCCESS
------------------------------------------
Total time:  1.489 s
Finished at: 2025-06-11T23:19:12+05:30
------------------------------------------
```

Unary Operators

```java
public class UnaryOperators {
    public static void main(String args[]){
        //Initilize number
        int num = 10;

        //Unary plus
        int result = +num;
        System.out.println("Result after unary plus:" + result);

        //Unary minus
        result = -num;
        System.out.println("Result after unary minus:" + result);

        //Pre-Increment
        result = ++num;
        System.out.println("Result after pre increment:" + result);

        //Post-increment
        result = num++;
        System.out.println("Result after post increment:" + result);

        //Pre-decrement
        result = --num;
        System.out.println("Result after pre decrement:" + result);

        //Post-decrement
        result = num--;
        System.out.println("Result after post decrement:" + result);
    }
}
```

```
--- exec:3.1.0:exec (default-cl
Result after unary plus:10
Result after unary minus:-10
Result after pre increment:11
Result after post increment:11
Result after pre decrement:11
Result after post decrement:11
------------------------------

BUILD SUCCESS
```

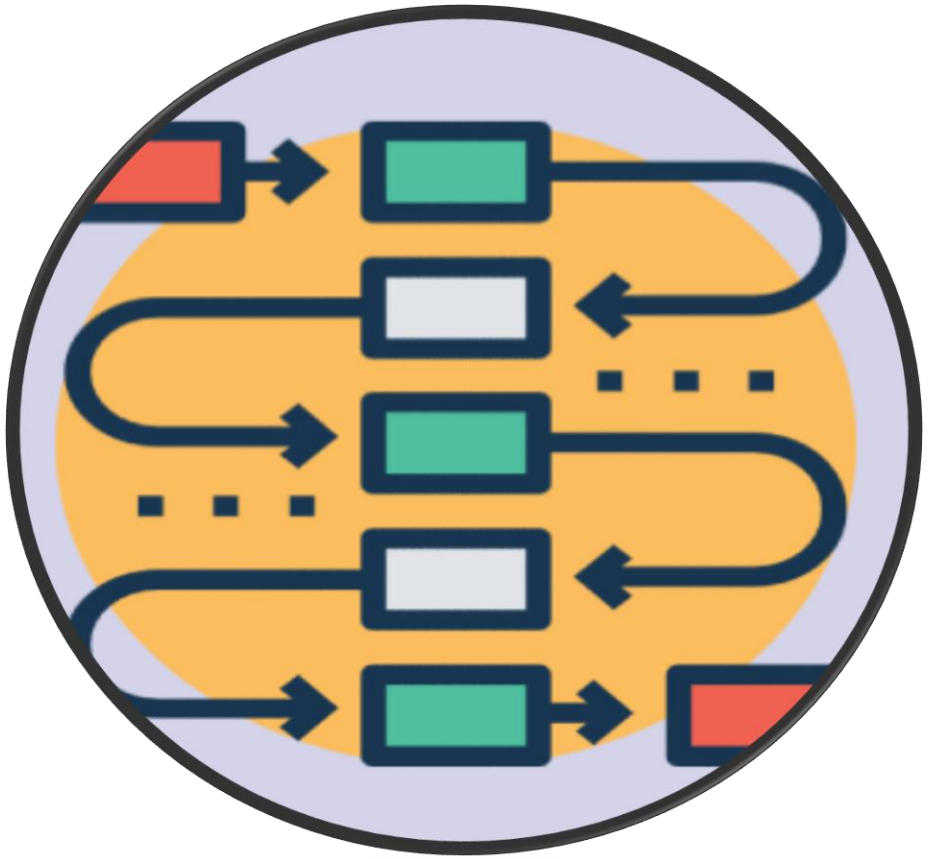
- **Unary plus operator (+)**
  - Perform a positive operation on num.
  - Result of the operation is stored in the *result* variable of type *int*.
- **Unary minus operator (-)**
  - Perform a negative operation on num.
  - The result of the operation is stored in the *result* variable.
- **Pre-increment operator (++num)**
  - Increment the value of num before using it in an expression.
  - The result of the operation is stored in the *result* variable.
- **Post-increment operator (num++)**
  - Increment the value of num after using it in an expression.
  - The result of the operation is stored in the result variable.
- **Pre-decrement operator (--num)**
  - Decrement the value of num before using it in an expression.
  - The result of the operation is stored in the result variable.
- **Post-decrement operator (num--)**
  - Decrement the value of num after using it in an expression.
  - The result of the operation is stored in the result variable.

Logical Operators

```java
public class LogicalOperators {
    public static void main (String args []){
        boolean x = true;
        boolean y = false;

        System.out.println("x && y: "+ (x && y));
        System.out.println("x &||y: "+ (x || y));
        System.out.println("!x: "+ (!x));
    }
}
```

```
--- exec:3.1.0:exec (de
x && y: false
x &||y: true
!x: false
------------------------
BUILD SUCCESS
```



Control Flow Statements

Part I

Decision Making Statements (if, if-else, if else if, switch)

Java Conditions and If Statements

- Java supports the usual logical conditions from mathematics:
  - 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
- Java has the following conditional statements:
  - Use if to specify a block of code to be executed, if a specified condition is true
  - Use else to specify a block of code to be executed, if the same condition is false
  - Use else if to specify a new condition to test, if the first condition is false
  - Use switch to specify many alternative blocks of code to be executed

if Statement

- Syntax

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

Q1. Take a user input and check if a number is positive

Q2. Take person age as keyboard input and check if the person is eligible to vote

Q3. Write a Java program to check if a given character is a vowel (a,e,i,o,u). Use lowercase characters only.

if-else Statement

- 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
}
```

Q1. Take a user input as a number. Check if the number odd or even

Q2. Ask user to enter their age and check he/she teenager.

if-else if Statement

- Syntax

```
if (condition 1){
     //block of code to be executed if the condition1 is true
} else if (condition 2){
    //block of code to be executed if the condition1 is false and
         condition2 is true
} else {
   //block of code to be executed if the condition1 is false and condition2 is
         false
}
```

Q1. Ask the user to enter a score. Print the grade based on this score:

- *90 above : A*
- *75 -89 : B*
- *60-74 : C*
- *Below 59 : D*

Switch Statement

- Syntax

```
switch (expression){
     case x:
         //code block
         break;
     case y:
         //code block
         break;
     default:
         //code block
}
```

- How this work
  - The switch expression is evaluated once
  - Compared with the values of each case.
  - If there is a match, the associated block of code is executed.
  - The break and default keywords are optional.

Switch Statement

Q1. Ask the user for a number (1 to 7) and print the corresponding day.

Q2. Input two numbers and perform calculator using switch.

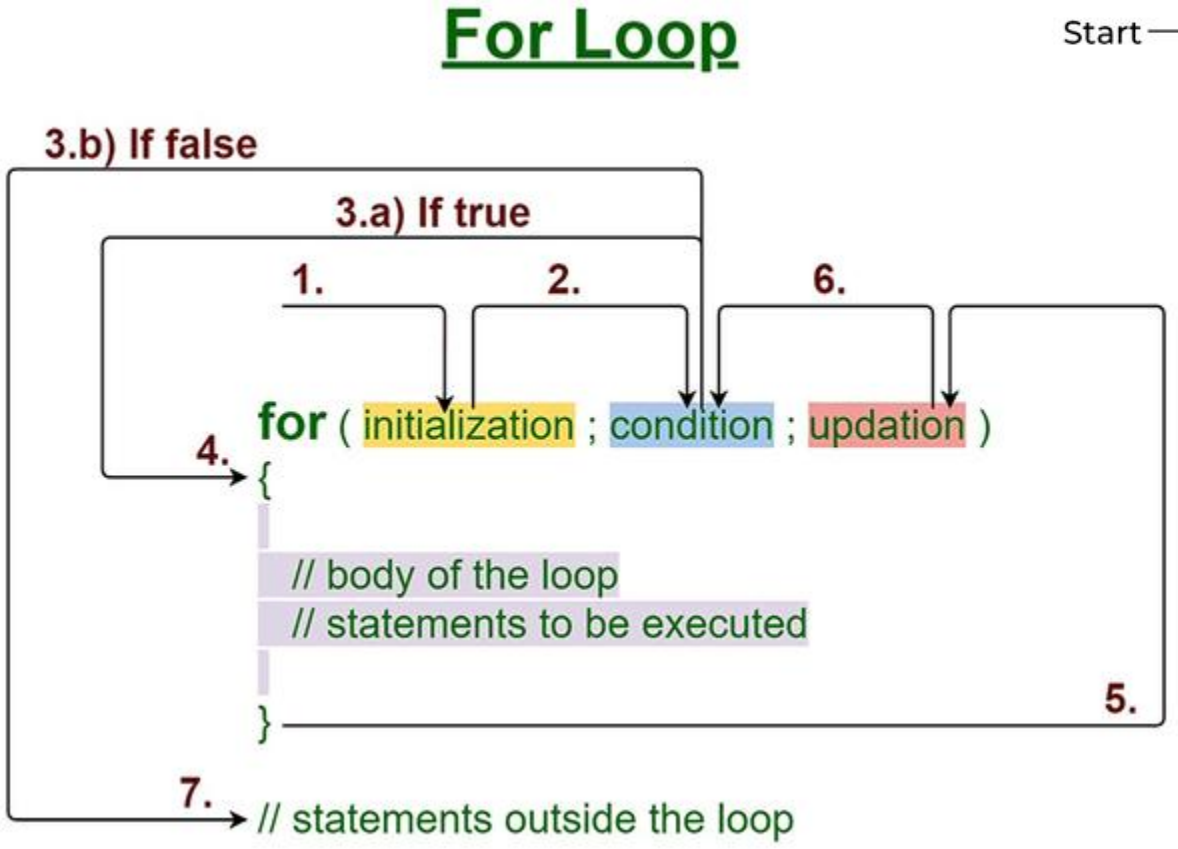
Q3. Input color from the user (red, green ,yellow) and display traffic light message.

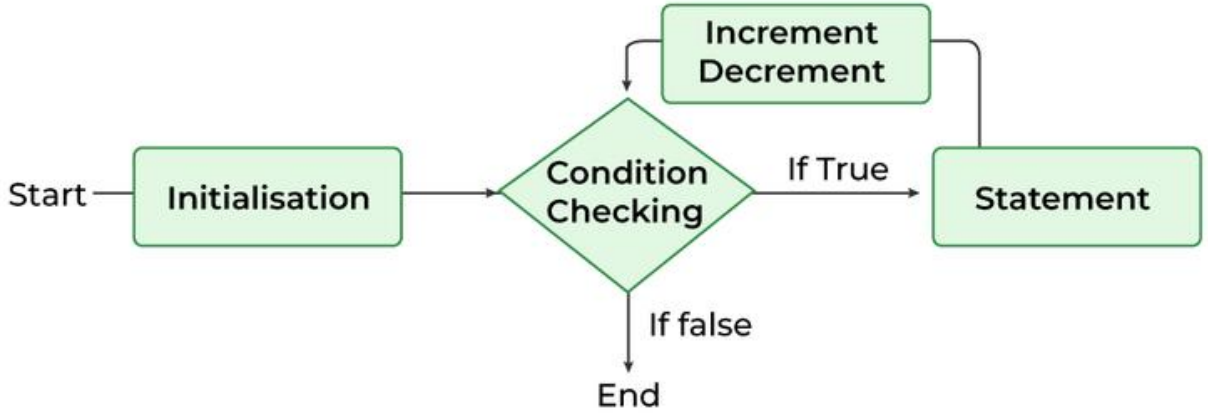
Q4. Ask the user for a month number (1–12) and print its name.

Part II

Looping Statements

for Loop in Java





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

```
for (initialization expr; test expr; update expr)
{
        // body of the loop
        // statements we want to execute
}
```

Q1. Print numbers from 1 to 5

```java
package com.mycompany.controlstructures;
public class ControlStructures {

    public static void main(String[] args) {
        for(int counter = 1; counter <= 5; counter++){
            System.out.println(counter);
        }
    }
}

```

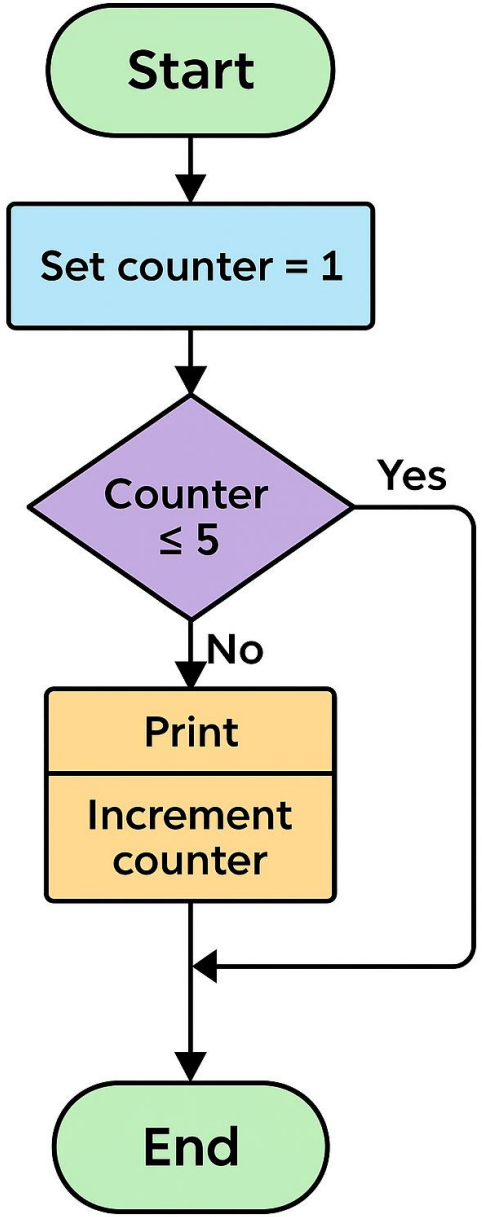
```
--- exec:3.1.0:exec (
1
2
3
4
5
------------------------
BUILD SUCCESS
------------------------
```

Pseudocode

Start

FOR counter FROM 1 TO 5 DO
    DISPLAY counter
END FOR

End



Nested for Loop

Q3. Print 5x5 star pattern

```java
package com.mycompany.controlstructures;
public class ControlStructures {

    public static void main(String[] args) {
        for(int row=1; row<=5; row++){
            for (int col=1; col<=5; col++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

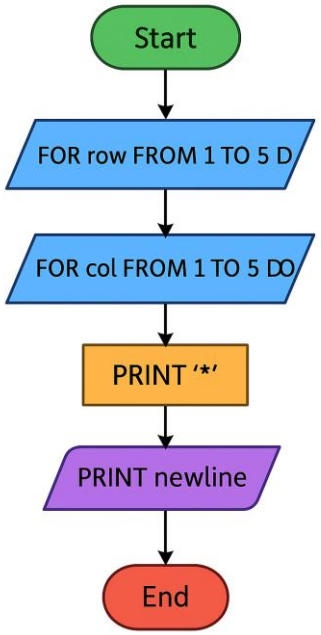
```
*****
*****
*****
*****
*****
----------------------
BUILD SUCCESS
```

Pseudocode

```
Start

FOR row  FROM 1 TO 5 DO
         FOR col FROM 1 TO 5 DO
                  DISPLAY "*"
         END FOR
         DISPLAY newline
END FOR

End
```



for-each Loop

- Also called **enhanced for loop**.
- Introduced in Java 5 to simplify iteration over arrays and collections.
- **Cleaner and more readable** than the traditional for loop
- Commonly used when the exact index of an element is not required
- Parameters:
  - **type: D**ata type of the elements in the array or collection.
  - **var:** Variable that holds the current element during each iteration.
  - **array:** Array or collection being iterated over.

```
for (type var : array) {
        statements using var;
}
```

Q4. Print each elements of an array using for-each loop

```java
package com.mycompany.controlstructures;
public class ControlStructures {

    public static void main(String[] args) {
        //array declare
        int arr [] = {1,2,3,4,5};

        for (int elements : arr ){
            System.out.print(elements + " ");
        }
    }
}
```

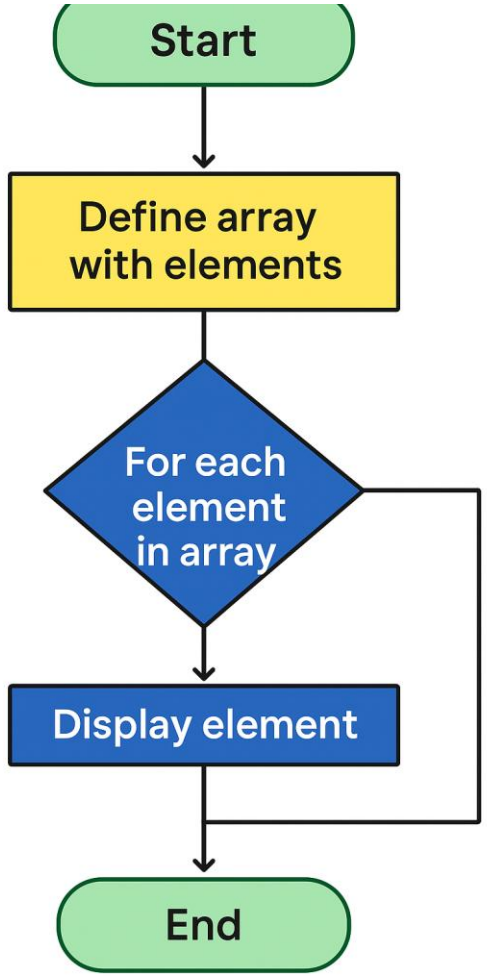
```
--- exec:3.1.0:exec
1 2 3 4 5
------------------------
BUILD SUCCESS
```

Pseudocode

Start

Define array with elements
FOR each elements IN array DO
    DISPLAY elements
END FOR

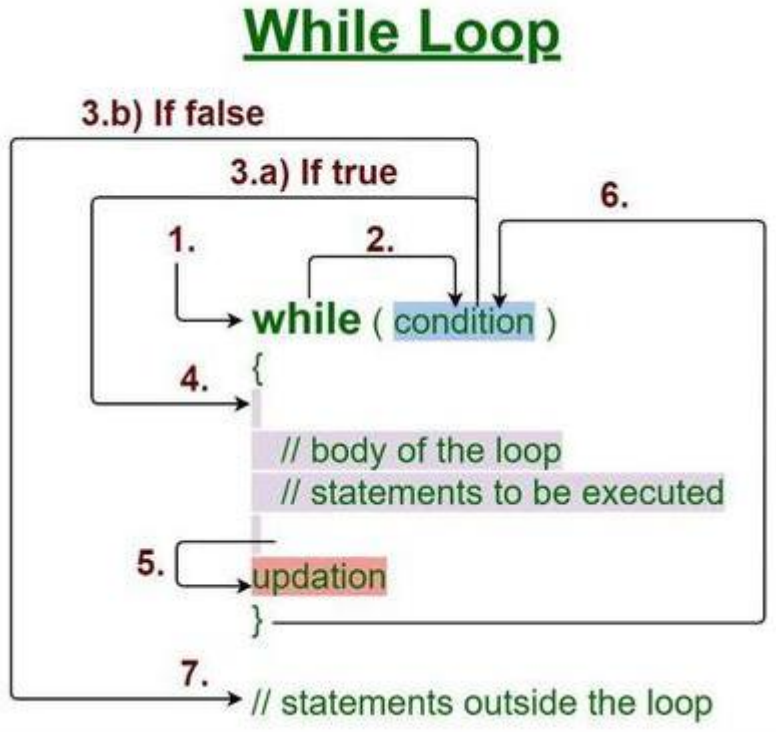
End



while Loop

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

```
while (condition) {
      // code block to be
executed
}
```



Quick Task

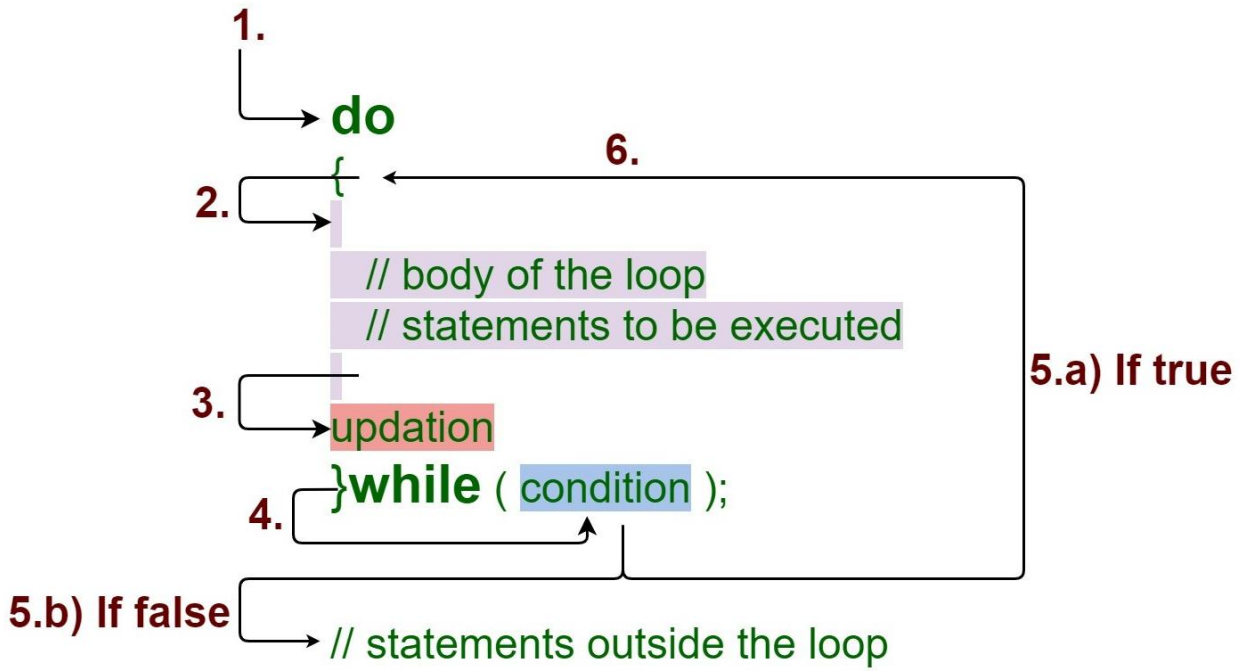
Q1. Write a program that asks the user to enter numbers. Keep adding them to a sum until the user enter 0. Print total sum.

Q2. Write a Java program that takes numbers as input and counts how many positive numbers the user entered. The program stops when the user enters a negative number.

do-while Loop

- Java **do-while loop** is an **Exit control loop.**
- Unlike for or while loop, a do-while **check for the condition after executing the statements of the loop body**

```
do{
   // Loop body
      Update Expression
}
//condition Check
while (test_expression);
```



- Note: This test_expression for the do-while loop must return a boolean value, else we would get compile time error

Quick Task

Q8. Write a program that displays a menu repeatedly until the user chooses to exit.

```
Menu:
1. Say Hello
2. Say Goodbye
3. Exit
Enter your choice: 1
Hello!
```

Q9. Write a Java program that repeatedly asks the user to enter a password until the correct password "secret123" is entered. When the correct password is entered, print "Access granted."

Part III

Branching Statements (Break, Continue)

Break Statement

- **Terminate from the loop immediately**
- When break call inside of the loop, the loop iteration stops there, and control returns from the loop immediately to the first statement after the loop
- If we are not sure about the actual number of iteration for the loop, or we want to terminate the loop based on some condition.

```
 */
package com.mycompany.ifelse;

public class BreakStatement {
    public static void main (String [] arg){
        for(int i = 0; i<10; i++){
            if(i==4){
                break;
            }
            System.out.println(i);
        }
    }
}
```

Continue Statement

- **Used to skip the current iteration of a loop**
- We can use continue statement inside any types of loops such as for, while, and do-while loop
- when we want to continue the loop but do not want the remaining statement after the continue statement.

```
package com.mycompany.ifelse;

public class BreakStatement {
    public static void main (String [] arg){
        int i = 0;
        while (i < 10) {
            if (i == 4) {
                i++;
                continue;
            }
            System.out.println(i);
            i++;
        }
    }
}
```

Break vs Continue

Break	Continue
The break statement is used to **terminate the loop immediately**.	The continue statement is used to **skip the current iteration of the loop**.
break keyword is used to indicate break statements in java programming.	continue keyword is used to indicate continue statement in java programming.
We can use a break with the switch statement.	We can not use a continue with the switch statement.
The break statement terminates the whole loop early.	The continue statement brings the next iteration early.
It stops the execution of the loop.	It does not stop the execution of the loop.

Quick Task

Q1. Skip Odd Numbers While Printing 1 to 10

Q2. Print All Letters Except Vowels

Q3. Find First Multiple of 7 Between 1 and 100

Thank you!