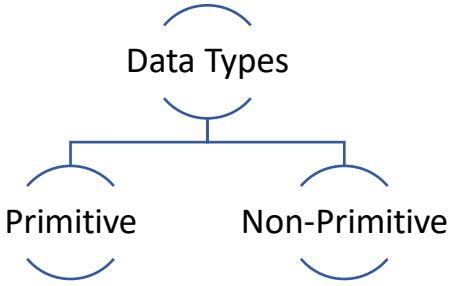
DAY 2

Data Types:

In Java, **data types** tell the computer what kind of data a variable can store.



Primitive: These are the built-in types i.e. predefined by Java.

There are 8 primitive data types in Java:

Data Type	Size	Example	Description	Range
byte	1 byte	byte b=100;	For very small integers	-128 to +127
short	2 byte	short s=3000;	For medium ranged integers	-32,768 to 32,767
Int	4 byte	int a=25;	For whole numbers(integers)	-2,147,483,648 to 2,147,483,647
Long	8 byte	long l=132456987L;	For very large integers	-9,223,372,036,854,775,808 to

				9,223,372,036,854,775,807
float	4 byte	float f=1.23f;	For small decimal numbers; ends with f.	~±3.4e−38 to ±3.4e+38 (7 decimal digits precision)
Double	8 byte	double d=123.45	For larger and more precise decimal numbers	~±1.7e−308 to ±1.7e+308 (15 decimal digits precision)
Char	2 byte	char s=’n’;	For single character, stored as Unicode	'\u0000' (0) to '\uffff' (65,535)
boolean	1 byte	boolean isjavafun= true;	Stores true or false only	True or false

Non-primitive: These data types refer to objects and can store multiple values and complex data.

They are:

1. **String**
   - Represents a sequence of characters (text).
   - Example: String name = "Java";

2. **Array**
   - Stores multiple values of the same type in a fixed-size structure.
   - Example: int[] numbers = {1, 2, 3};

3. Class

- A blueprint for creating objects.
- You define your own data type using classes.
- Example:

```
class Car {
    int speed;
}
```

4. Object

- The root of all non-primitive types.
- Every class in Java is an object.
- Example: Car myCar = new Car();

5. Interface

- A reference type similar to a class that only contains method signatures.
- Used for abstraction and multiple inheritance.
- Example:

```
interface Animal {
    void makeSound();
}
```

6. Enum (Enumeration)

- A special data type to define collections of constants.
- Example:
- enum Days { MONDAY, TUESDAY, WEDNESDAY }

7. **Wrapper Classes**
   - These are object versions of primitive types.
   - Examples:
     - Integer for int
     - Float for float
     - Double for double
     - Character for char
     - Boolean for boolean

Literals:

Literals are *fixed values* that we write directly in the code and assign to variables. These values don't change while the program runs. They represent constant values of different data types.

Example:

- int num = 100; // 100 is an integer literal
- float pi = 3.14f; // 3.14 is a float literal
- char ch = 'A'; // 'A' is a character literal
- String name = "Tom"; // "Tom" is a string literal

Variables:

We can think of variables as a placeholder attached to a specific location in the computer's memory. Now, instead of using that value again and again in the code, we can refer to the placeholder instead, which is storing that value.

This helps us to store, reuse and change the data whenever we want in the program.

Rules for Naming Variables:

1. Must start with a letter (A–Z or a–z), underscore _, or dollar sign $.
2. Cannot start with a number.
3. Cannot use Java keywords (like int, class, public) as variable names.
4. Variable names are case-sensitive (Age and age are different).

Types of Variables:

Java has three main types of variables, depending on where they are declared and how they are used.

1. Local Variable:

- Declared inside a method or block.
- Only accessible within that method or block.
- Not given any default value — you must initialize it before using.
- Example:

```
void show() {
    int x = 10;  // Local variable
    System.out.println(x);
}
```

x only exists inside the show() method.

2. Instance Variable

- Declared inside a class, but outside any method.
- It belongs to an object of the class.

- Each object gets its own copy of the variable.
- Gets default values if not initialized (e.g., 0 for int, null for objects).
- Example:
  class Student {
  int age; // Instance variable
  }
  Every Student object will have its own age.

3. Static Variable

- Declared with the static keyword inside a class.
- Shared by all objects of that class (only one copy exists).
- Can be accessed using the class name too.
- Example:
  class Student {
  static int count; // Static variable
  }
  count is common for all Student objects.

In short:

- Local – inside method, temporary, no default value.
- Instance – inside class (not method), per object.
- Static – inside class, shared by all objects.

Type Conversion & Casting:

Type Conversion in Java is the process of changing one data type to another. This is useful when you want to work with different types of data, like converting an integer to a floating-point number.

There are *two types of type conversion* in Java:

- **Automatic Type Conversion (Widening)** – Java does this automatically.
- **Manual Type Conversion (Narrowing)** – This requires explicit instruction from the programmer.

1. Automatic Type Conversion(Widening):

When Java automatically converts one data type to another, it's called automatic type conversion or widening.
This typically happens when we're moving from a smaller data type to a larger one. Java does this automatically because it knows that a larger type can hold all the values of a smaller type without losing any data.
Example:

```
int a = 10;       // 'a' is an integer (int)
double b = a;     // 'a' is automatically converted to double
```

(here, an int can easily be converted to a double because a double can hold all the values of an int (including decimals).
)

2. Manual Type Conversion (Narrowing):

Narrowing Conversion (from a larger type to a smaller one) doesn’t happen automatically because it can lose data (e.g., a double to int).
Example:

```
double x = 10.5;
int y = x;  // This will cause a compile-time error!
```

So we, explicitly convert a larger data type to a smaller one (like converting a double to an int) by using parentheses with the target type.

Example:

```
double x = 9.7;       // x is a double type with value 9.7
int y = (int) x;      // Manual type casting from double to int
```

This is also known as *Type Casting.*

Promotion:

During operations, smaller data types like byte, short, or char are automatically converted (promoted) to a larger type like int to avoid data loss or overflow.

Example:

```
byte a = 10;

byte b = 20;

int result = a + b;  // a and b are promoted to int before addition
```

Even though a and b are byte, Java promotes them to int during the addition. So, the result is stored in an int variable, not byte, to handle the operation safely.

Input Using Scanner Class:

The Scanner class is used to take user input in Java.

Steps:

- Import Scanner: import java.util.Scanner;
- Create Scanner object: Scanner sc = new Scanner(System.in);
- Use methods to take input:
  - int age = sc.nextInt(); // for integer input
  - String name = sc.next(); // for single word string
  - String fullName = sc.nextLine(); // for full sentence
  - float marks = sc.nextFloat(); // for decimal input

Operators:

Operators are symbols used to perform operations on variables and values.

Types of Operators in Java:

- **Arithmetic Operators:** Perform math operations (+, -, *, /, %).
- **Relational Operators:** Compare values (==, !=, >, <, >=, <=).
- **Logical Operators:** Combine conditions (&&, ||, !).
- **Assignment Operators:** Assign values and modify variables (=, +=, -=, *=, /=).
- **Unary Operators:** Modify a single variable (++, --).
- **Ternary Operator:** A short if-else statement (condition ? true : false).

Control Statements:

Control statements in Java are used to control the flow of the program.

They are as follows:

1. **Conditional (Decision-making) Statements:**
   These check conditions and decide which code to run.

   - **if – Runs a block if the condition is true.**

```
if (age > 18) {
System.out.println("Adult");
}
```

   - **if-else – Runs one block if true, another if false.**

```
if (age > 18) {
    System.out.println("Adult");
} else {
    System.out.println("Not adult");
}
```

   - **else-if – Checks multiple conditions one by one.**

```
if (marks >= 90) {
    System.out.println("Grade A");
} else if (marks >= 75) {
    System.out.println("Grade B");
} else {
    System.out.println("Grade C");
}
```

   - **switch – Checks one variable against many fixed values.**

```
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
```

```
    default: System.out.println("Other day");
}
```

2. Loops (Iteration Statements):

Used to repeat a block of code multiple times.

- **for – Best when you know how many times to loop.**

```
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

- **while – Runs while a condition is true. Checks first, then runs.**

```
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```

- **do-while – Runs at least once, then checks condition.**

```
int i = 1;
do {
    System.out.println(i);
    i++;
} while (i <= 5);
```

3. Jump Statements:

Used to jump or skip parts of the code.

- **break – Exits a loop or switch early.**

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) break;
    System.out.println(i);
}
```

- **continue – Skips the current loop step and goes to the next.**

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    System.out.println(i);
}
```

- **return – Exits from a method and goes back to where it was called.**