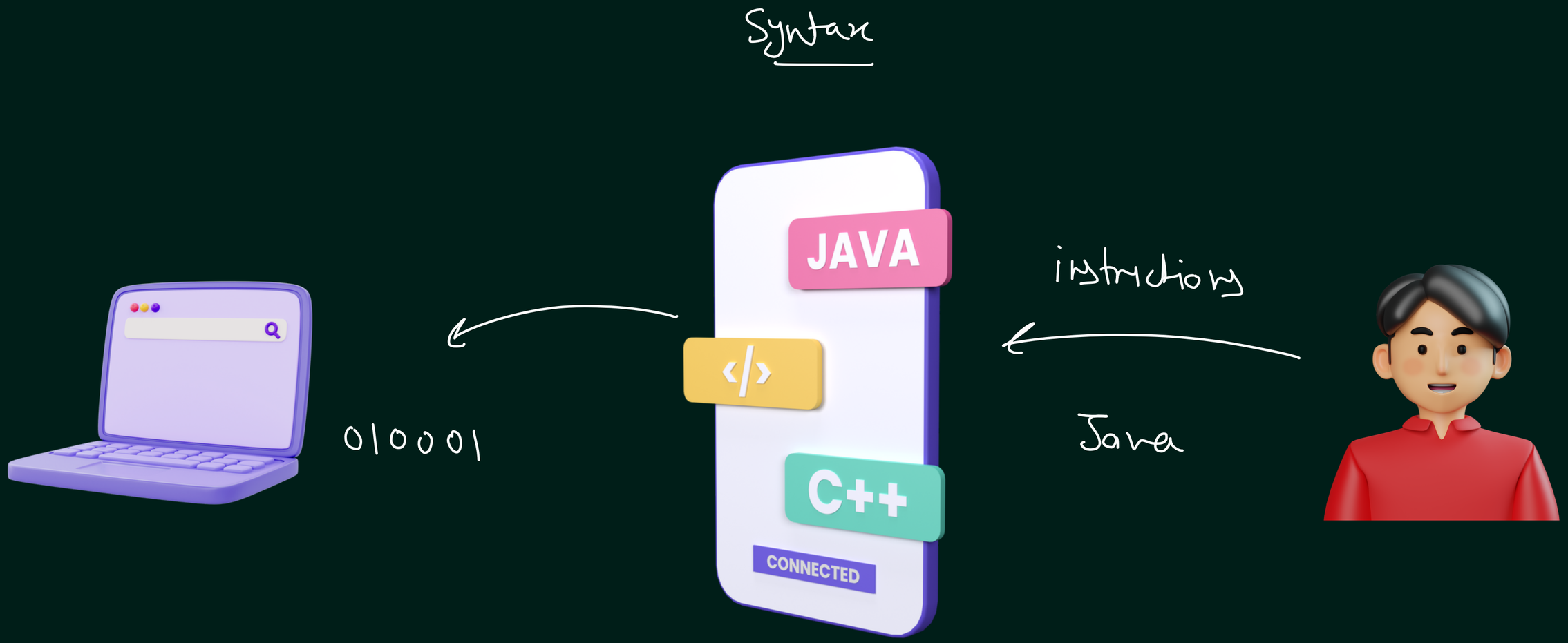
CODING SHUTTLE

Java Basics

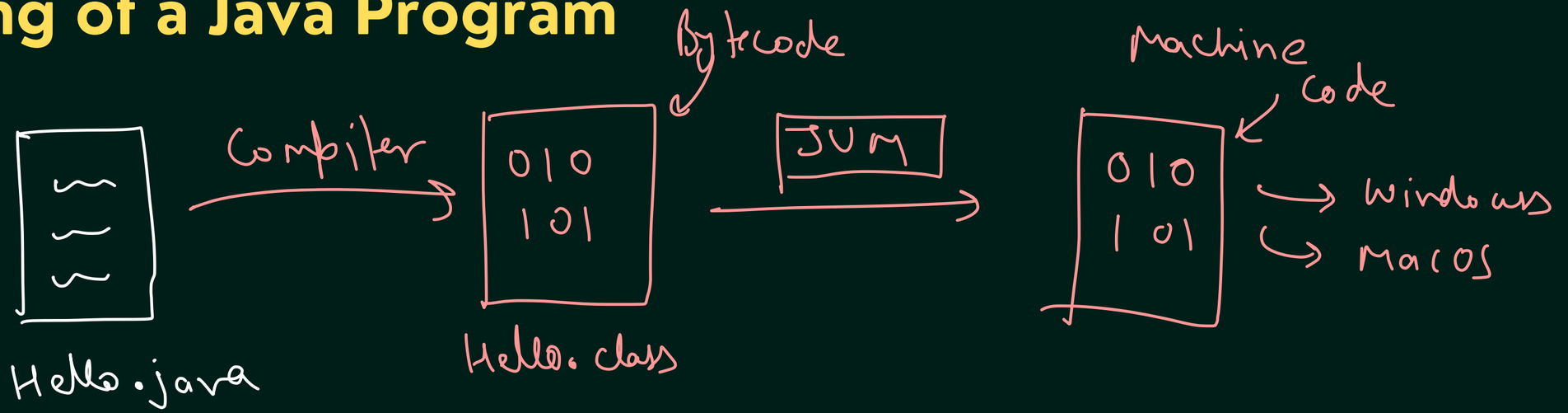
In This Lecture

1. Programming Language
2. Working of a Java Program
3. Basic Java Program
4. Keywords in Java
5. Variables in Java
6. Data Types in Java
7. Types Conversion in Java
8. Java Comments

Programming Language



Working of a Java Program

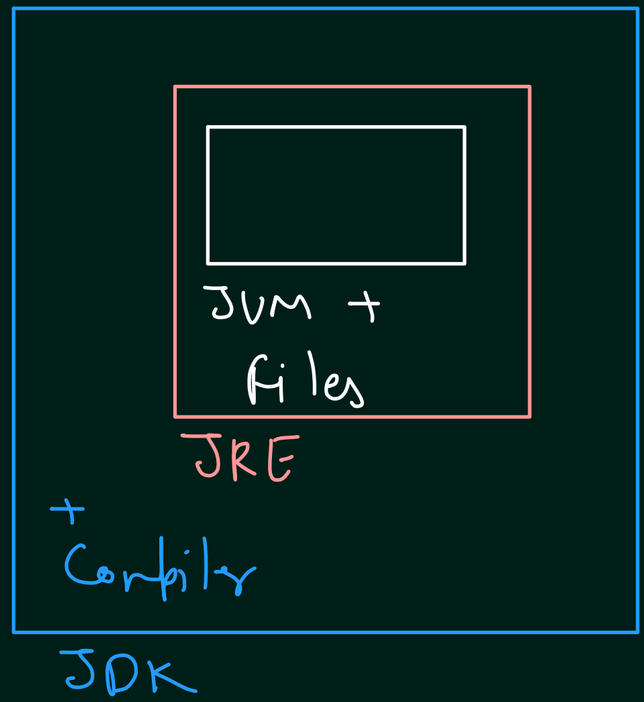


JVM: JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.
When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

JVM, JRE and JDK

JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.

JDK (Java Development Kit) is a software development kit required to develop applications in Java.
In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).



Basic Java Program

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

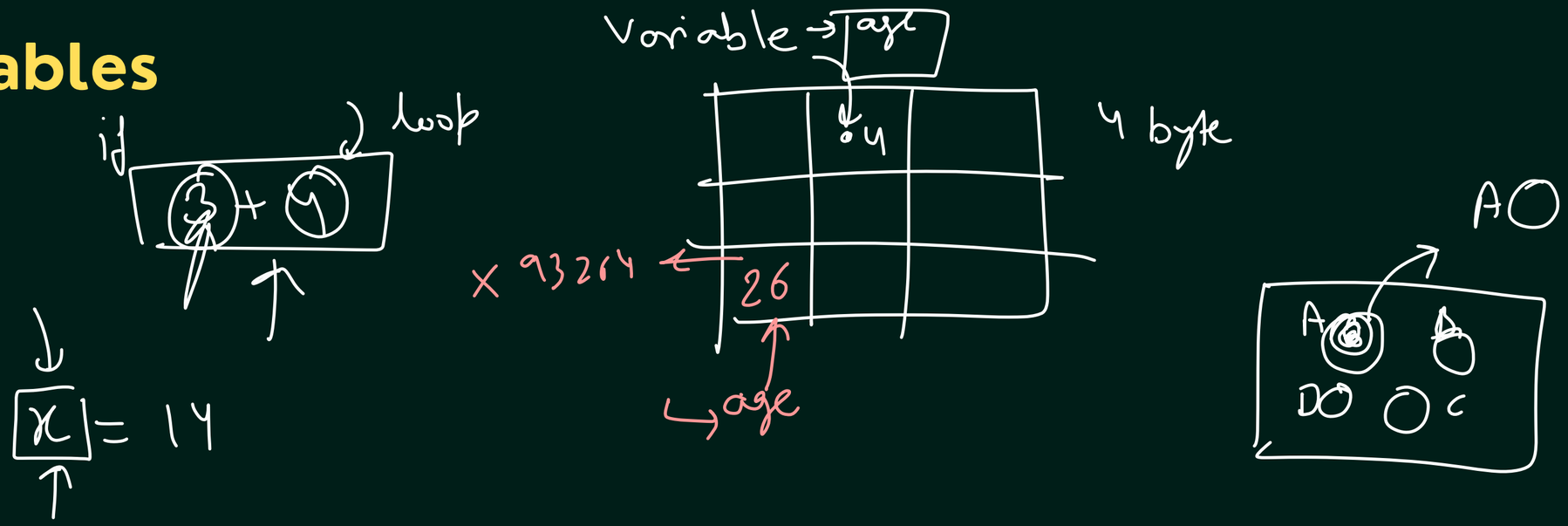
Java Keywords

Do NOT use as variable names

> 90 %

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Java Variables



Rules for Naming Variables in Java

- Java is case-sensitive. Hence, age and AGE are two different variables.
- Variables must start with either a letter or an underscore, _ or a dollar, $ sign.
- Variable names can't use whitespace.
- Variable names cannot be a keyword.

Java Data Types

4

4
12
3.14
'A'
false

4

2

1

2

8

8 Types of Data Types

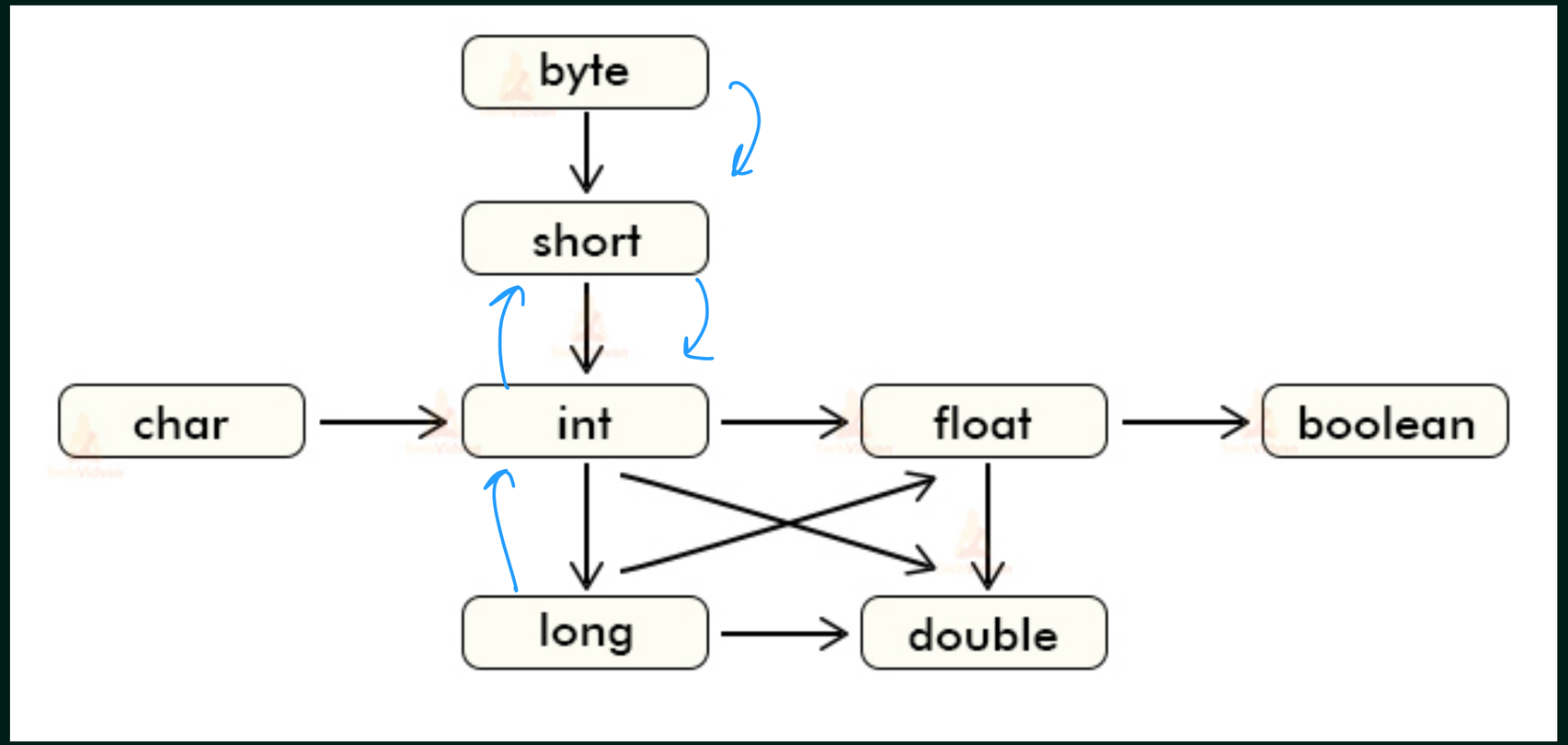
Primitive ?

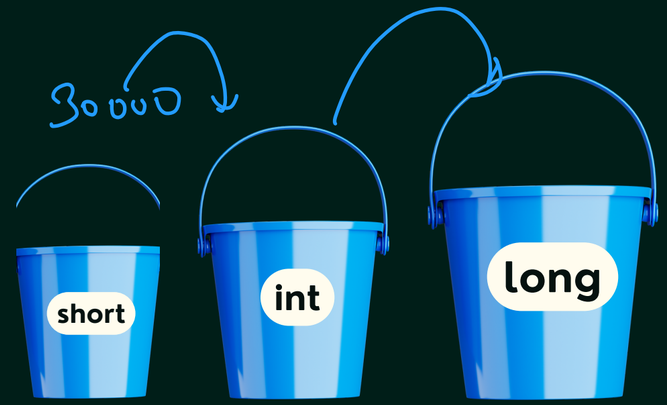
boolean marks = true;

DATA TYPES	SIZE	DEFAULT	EXPLAINATION
boolean	1 bit	false	Stores true or false values
byte	1 byte/ 8bits	0	Stores whole numbers from -128 to 127
short	2 bytes/ 16bits	0	Stores whole numbers from -32,768 to 32,767
int	4 bytes/ 32bits	0	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes/ 64bits	0L	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes/ 32bits	0.0f	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes/ 64bits	0.0d	Stores fractional numbers. Sufficient for storing 15 decimal digits
char	2 bytes/ 16bits	'\u0000'	Stores a single character/letter or ASCII values

Data Types Implicit Conversion

Direct / Widening





Data Types Explicit Conversion

```
int age = 12
short newAge = (short) age;
```

It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

Java Comments

In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are mainly used to help programmers to understand the code.

Types of Java Comments

Single Line Java Comments

A single-line comment starts and ends in the same line. To write a single-line comment, we can use the // symbol.

=

Cmd + / | Ctrl + /

Multi Line Java Comments

When we want to write comments in multiple lines, we can use the multi-line comment. To write multi-line comments, we can use the /*....*/ symbol.

Cmd + Shift + /

Ctrl + Shift + /

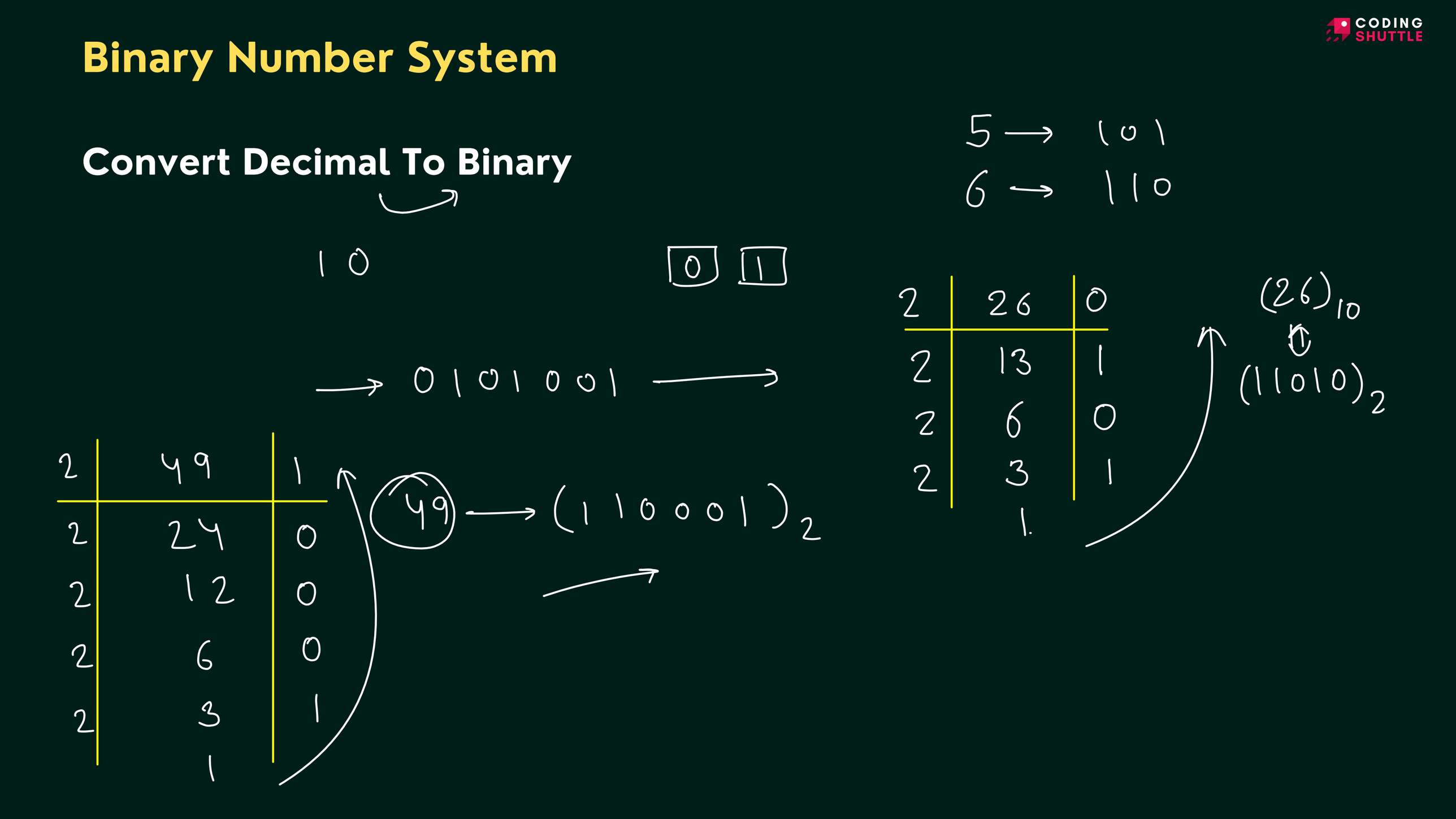
Binary Number System, Java Operators & Taking User Input

In This Lecture

1. Binary Number System
2. Operators in Java
3. Taking User Input

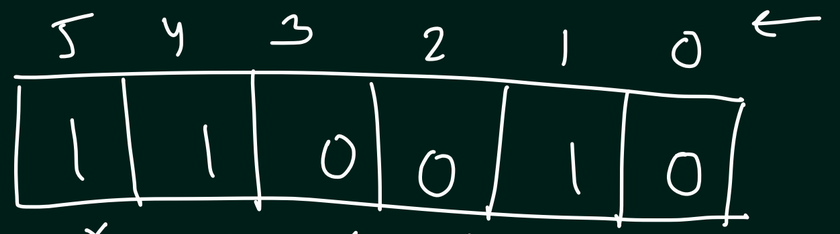
Binary Number System

Convert Decimal To Binary



Binary Number System

Convert Binary To Decimal



$(110010)_2 = (50)_{10}$

$32 + 16 + 0 + 0 + 2 + 0$

$= 50$

$(10101)_2 = (21)_{10}$

4 3 2 1 0

$16 + 4 + 1$

```
int age = 21;
```

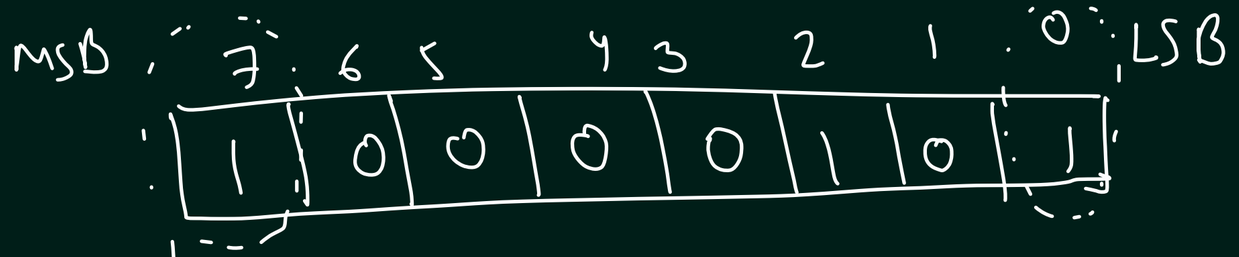
Binary Number System

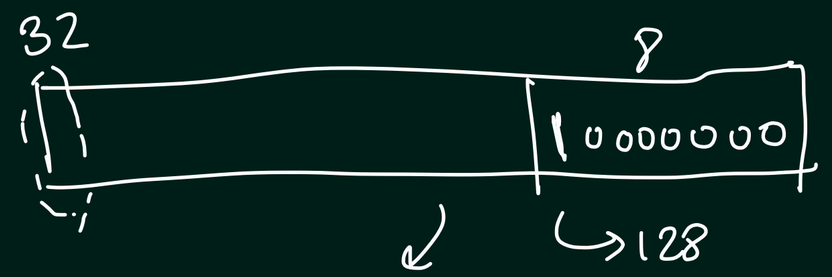
Convert Binary To Decimal

```
int age = 128;
byte newAge = (byte) age;
```

↳ -128

1 Byte → 8 bits





1 → -ve

0 → +ve

0 1 1 1 1 1 1 1

↓

+127

↓

-ve

1 0 0 0 0 0 0 0

↓

-128

1 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1

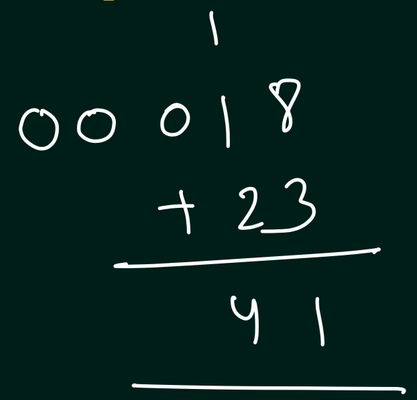
\+ 1

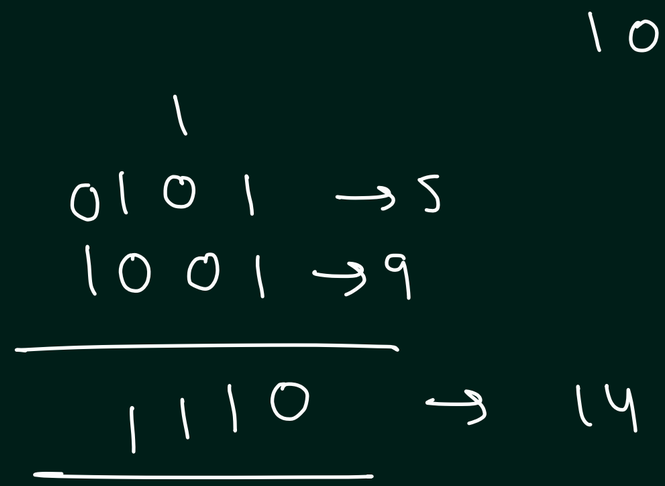
---

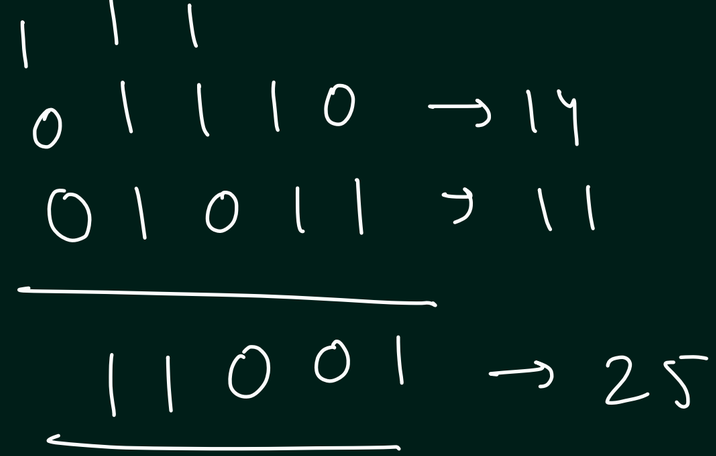
1 0 0 0 0 0 0 0

Byte

Binary Addition







Binary Subtraction

$$9 - 4 = 5$$

$$9 + (-4) = 5$$

$$(-4)_{10} \rightarrow (\quad)_2$$

$$\begin{array}{r} 1001 \\ -\ \ 100 \\ \hline \end{array}$$

2's Complement

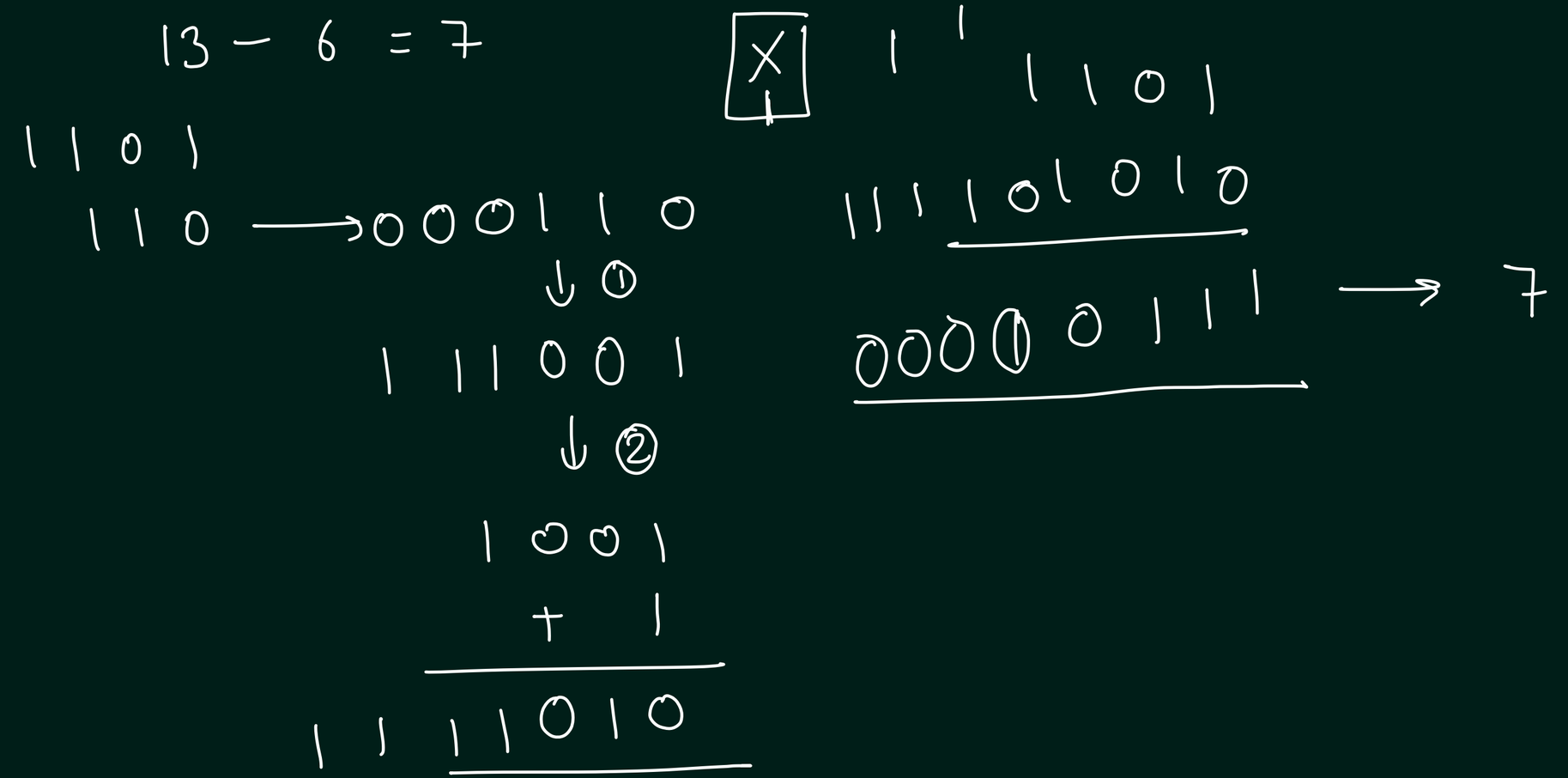
① swap the bits

② add 1

$$\begin{array}{r} 0000100\,1 \\ +\ 1111100 \rightarrow (-4) \\ \hline 00001101 \\ \hline \end{array}$$

$$\begin{array}{l} 00000100 \\ (1)\ 11111011 \\ (2)\ \qquad +\ 1 \\ \hline 11111100 \\ \hline \end{array}$$

Binary Subtraction

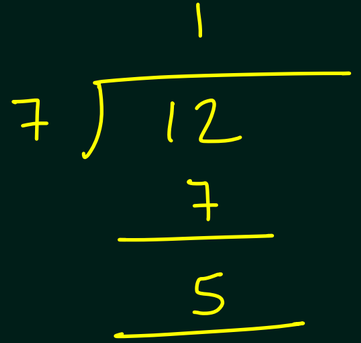


Types of Operators in Java

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators

1. Arithmetic Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)



12 % 7 = 5

12 / 7 = 1

2. Assignment Operators

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

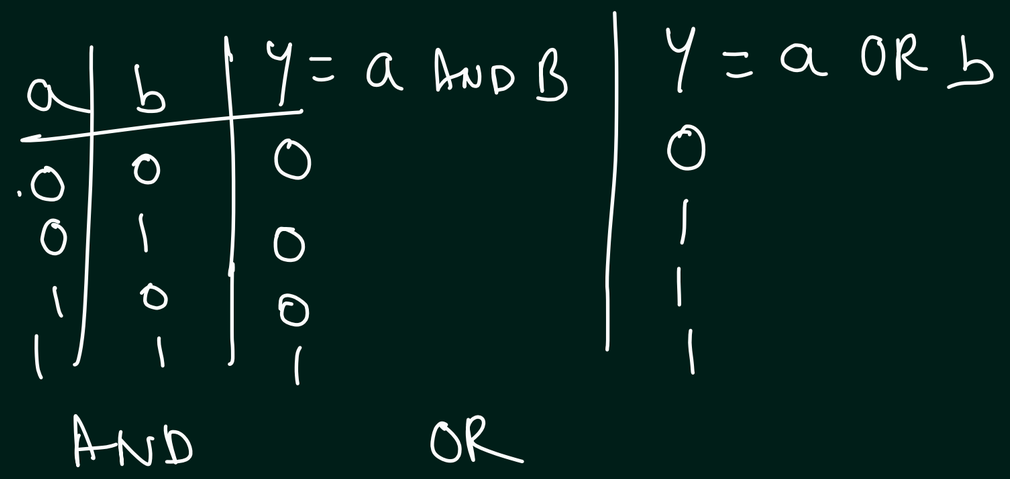
3. Relational Operators

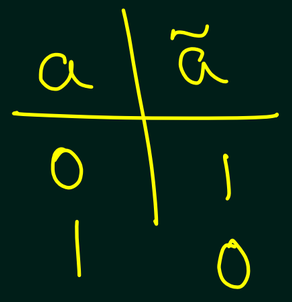
→ always return boolean value → true/ false

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **true**

4. Logical Operators

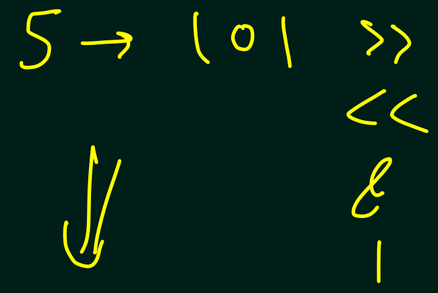
Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa





5. Bitwise Operators

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR



Bitwise operation
↳ Bit manipulation

Other Operators

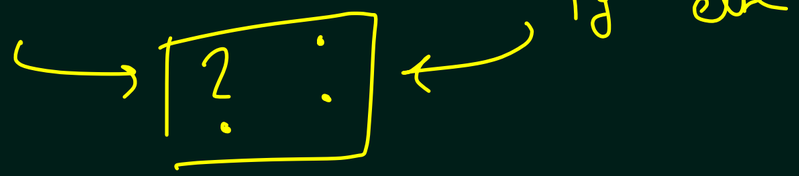
Increment/ Decrement Operators

++ --

a++; → a = a+1

a--; → a = a-1;

Ternary Operators



Taking User Input using Scanner

In order to use the object of Scanner, we need to import java.util.Scanner package.

Class

```
Scanner sc = new Scanner(System.in);
```

- sc.nextInt()
- sc.nextFloat()
- sc.nextLine()
- sc.next()

Various Input Types using Scanner

We can use nextLong(), nextFloat(), nextDouble(), and next() methods to get long, float, double, and string input respectively from the user.

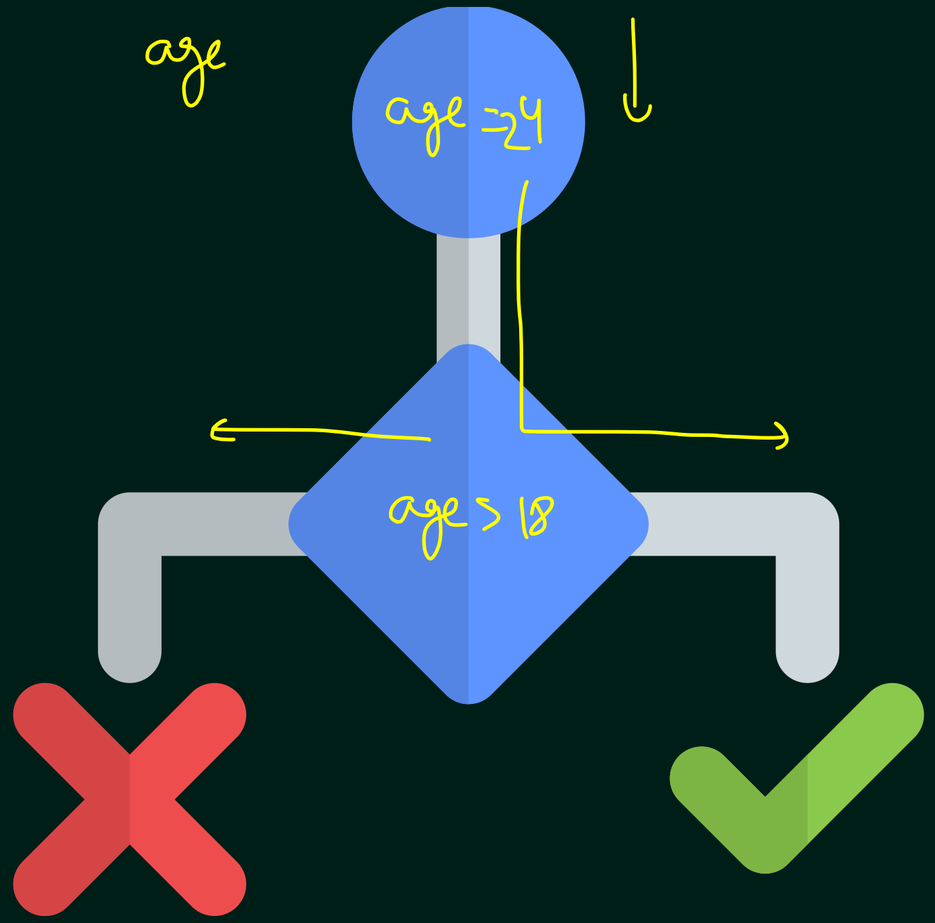
Note: It is recommended to close the scanner object once the input is taken using the close() method

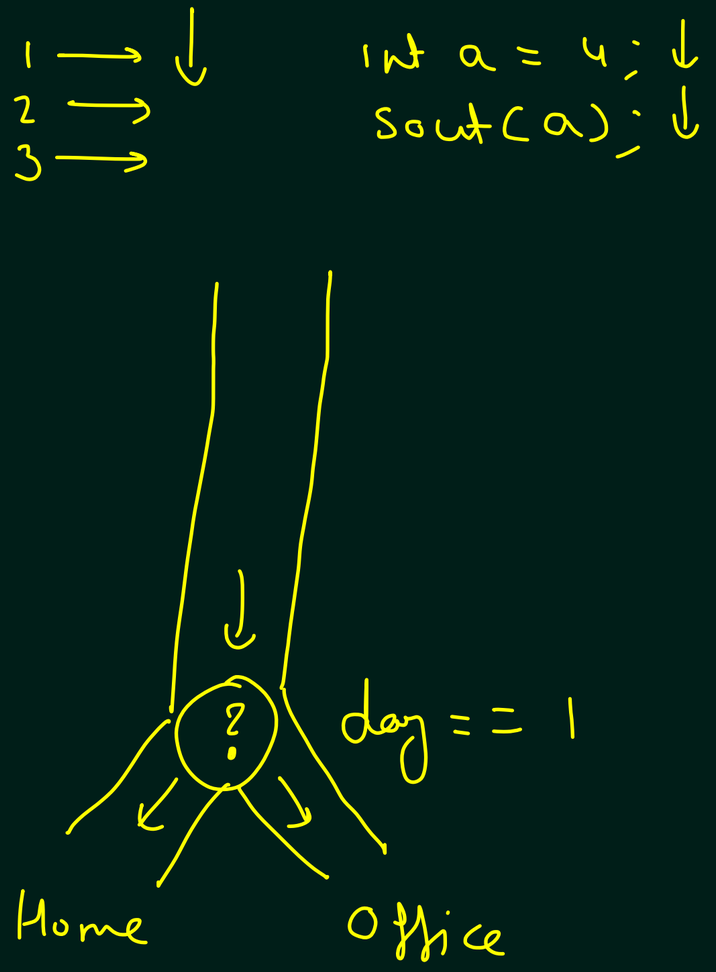
Java Conditional Statements

In This Lecture

1. if-else statement
2. if-else if- else statement
3. Nested if-else statement
4. Working with the Logical Operators
5. Java Ternary Operator
6. Java Switch statement

Java Conditional Statements





Java if-else statements

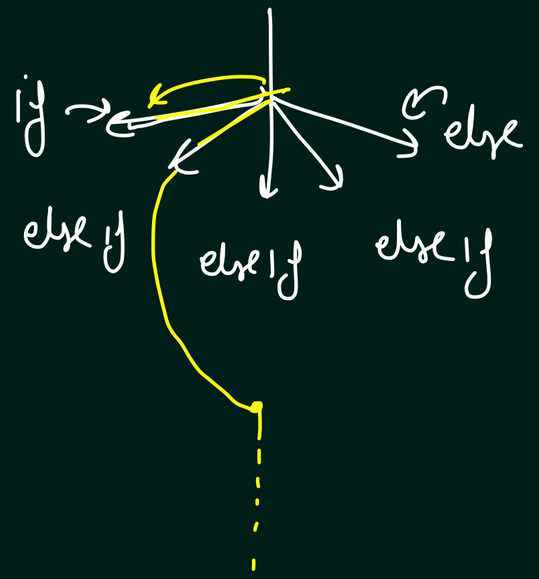
The if statement executes a certain section of code if the test expression is evaluated to true.

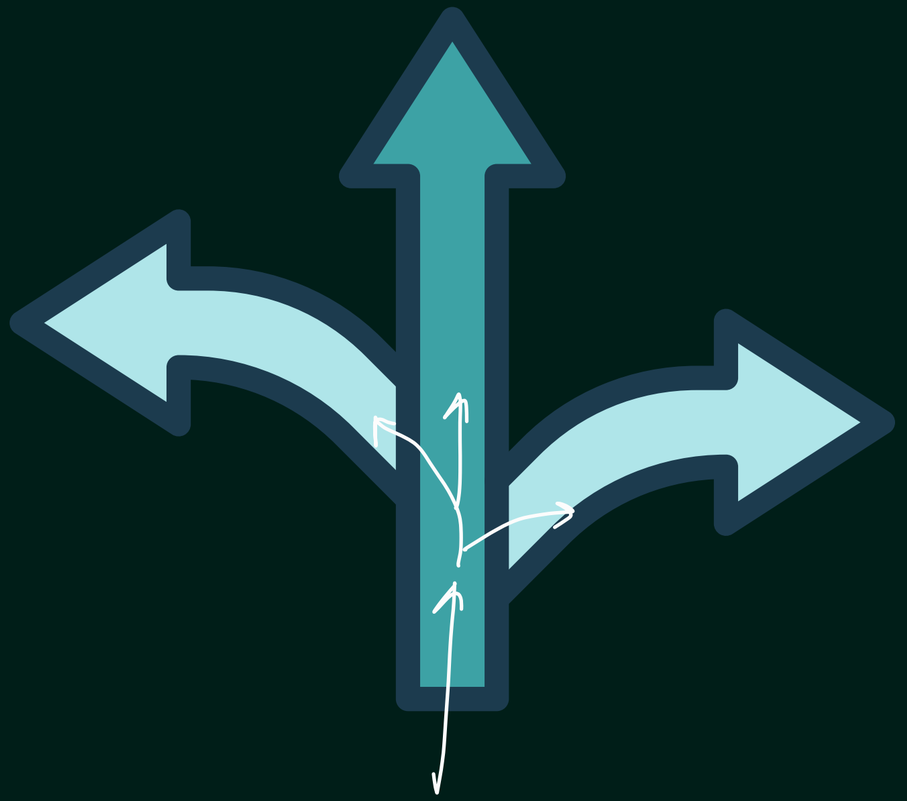
Statements inside the body of else block are executed if the test expression is evaluated to false. This is known as the if-...else statement in Java.

```
if ( expression) {
    true
} else {

    false

}
```

Java if - else if - else statements

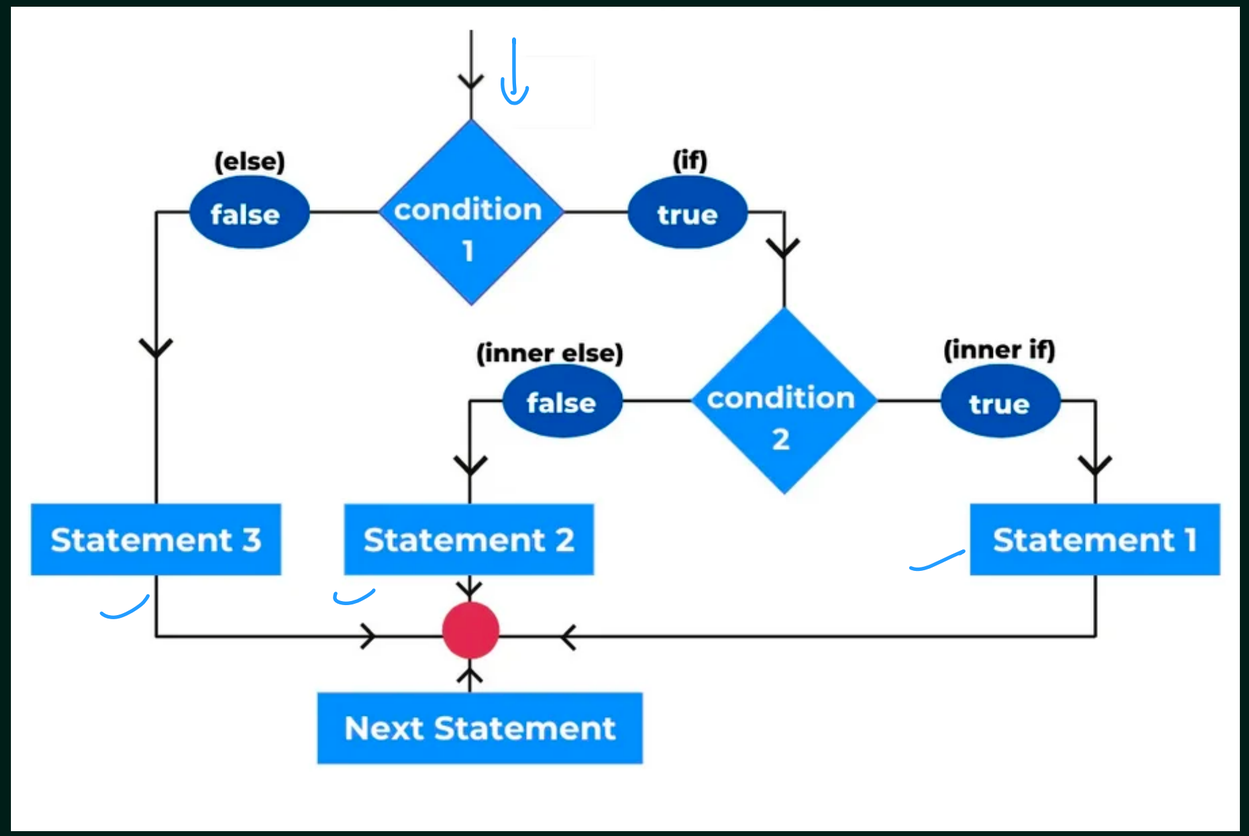
In Java, we have an if...else...if ladder, that can be used to execute one block of code among multiple other blocks.

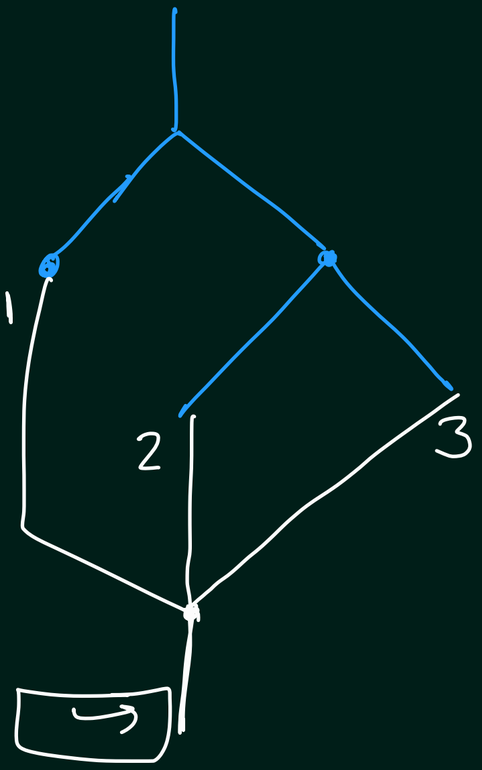




Nested if-else statements

In Java, we have an if...else...if ladder, that can be used to execute one block of code among multiple other blocks.



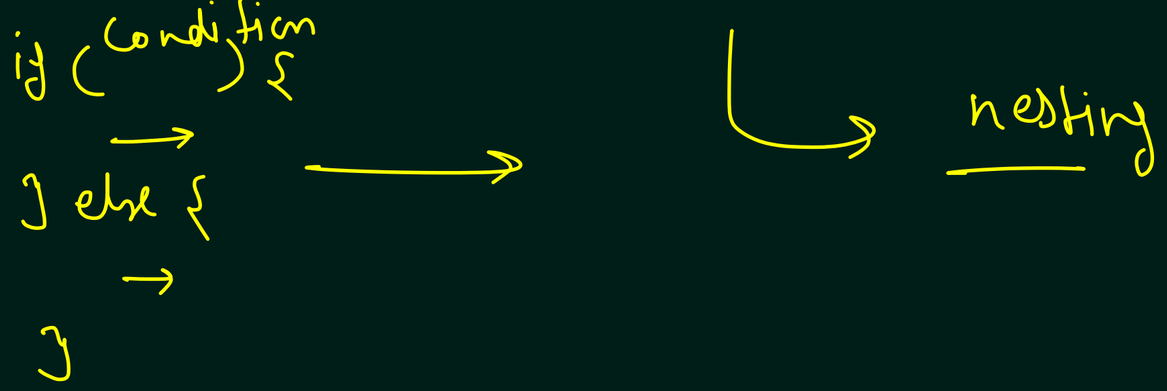


Java Ternary Operator

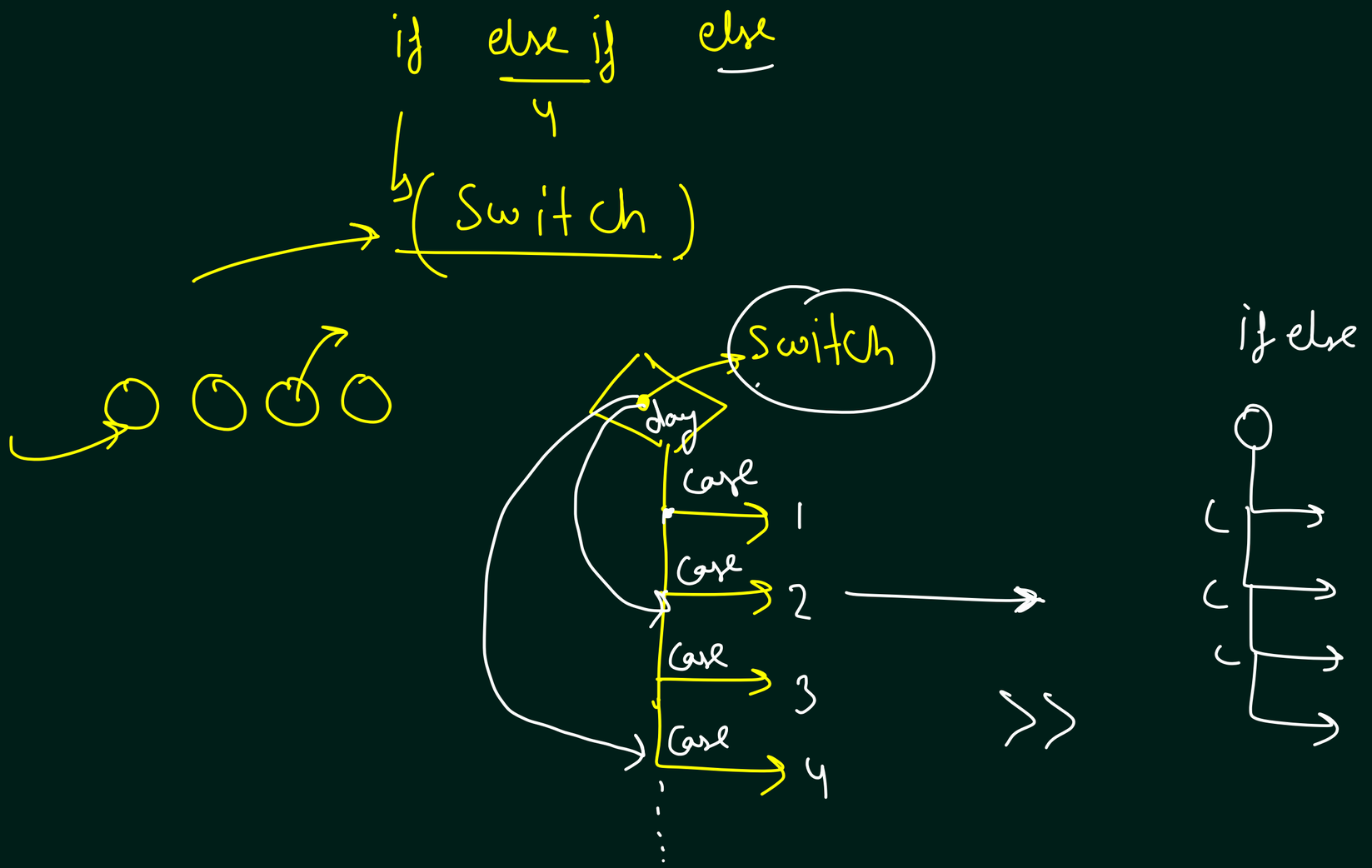
A ternary operator evaluates the test condition and executes a block of code based on the result of the condition.

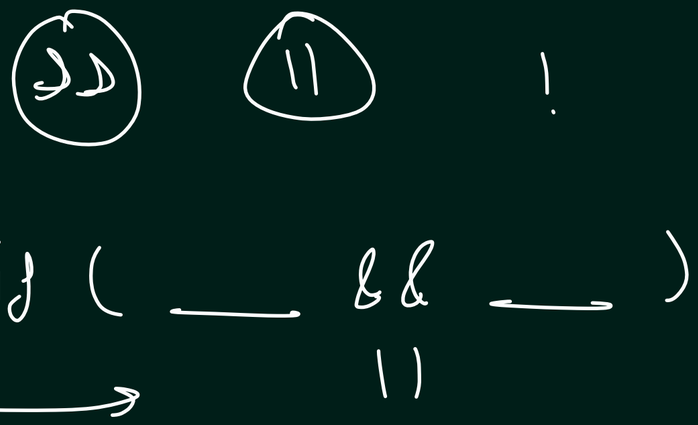
The ternary operator can be used to replace certain types of if...else statements.

condition ? expression1 : expression2;



Java switch Statement




&&
||
!
if (___ && ___)
||

CODING SHUTTLE

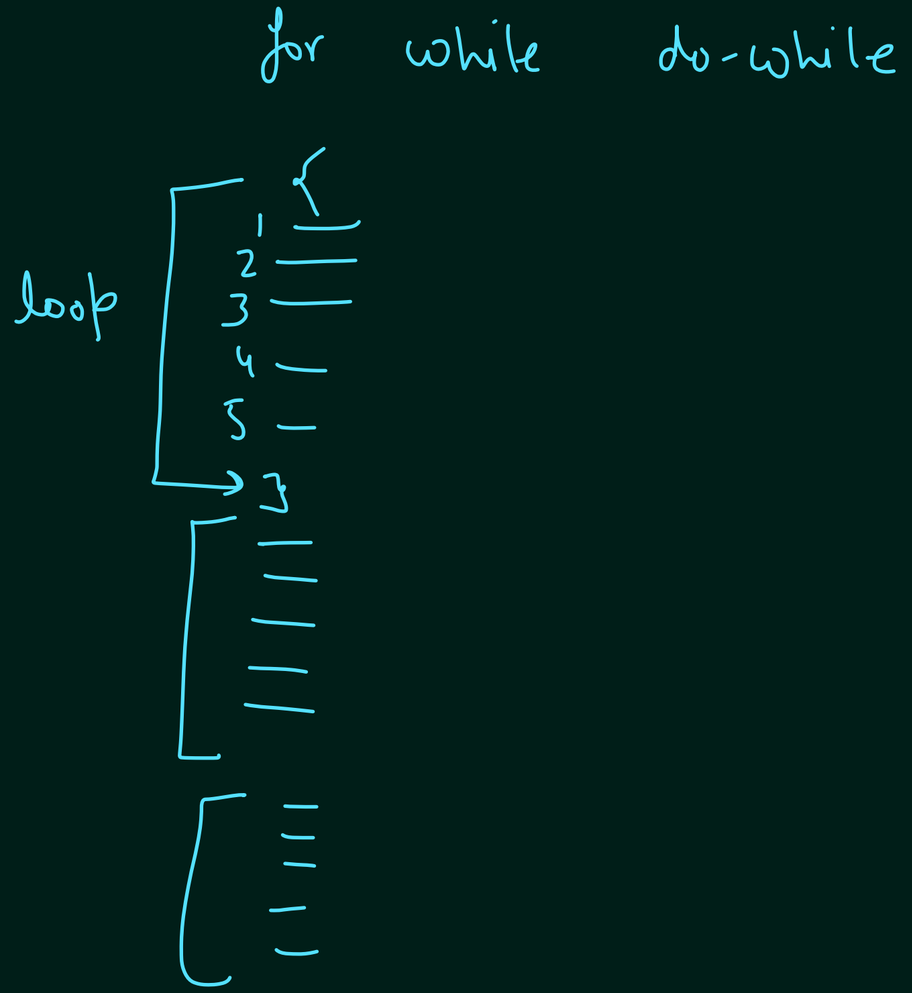
Loops in Java

In This Lecture

1. for loop
2. while loop
3. do-while loop
4. break & continue statements
5. Nested loops
6. Labeled break & continue statements

Elements of Java Loops

- Initialization Expression(s)
- Test Expression(Condition)
- Update Expression(s)
- Body of the loop



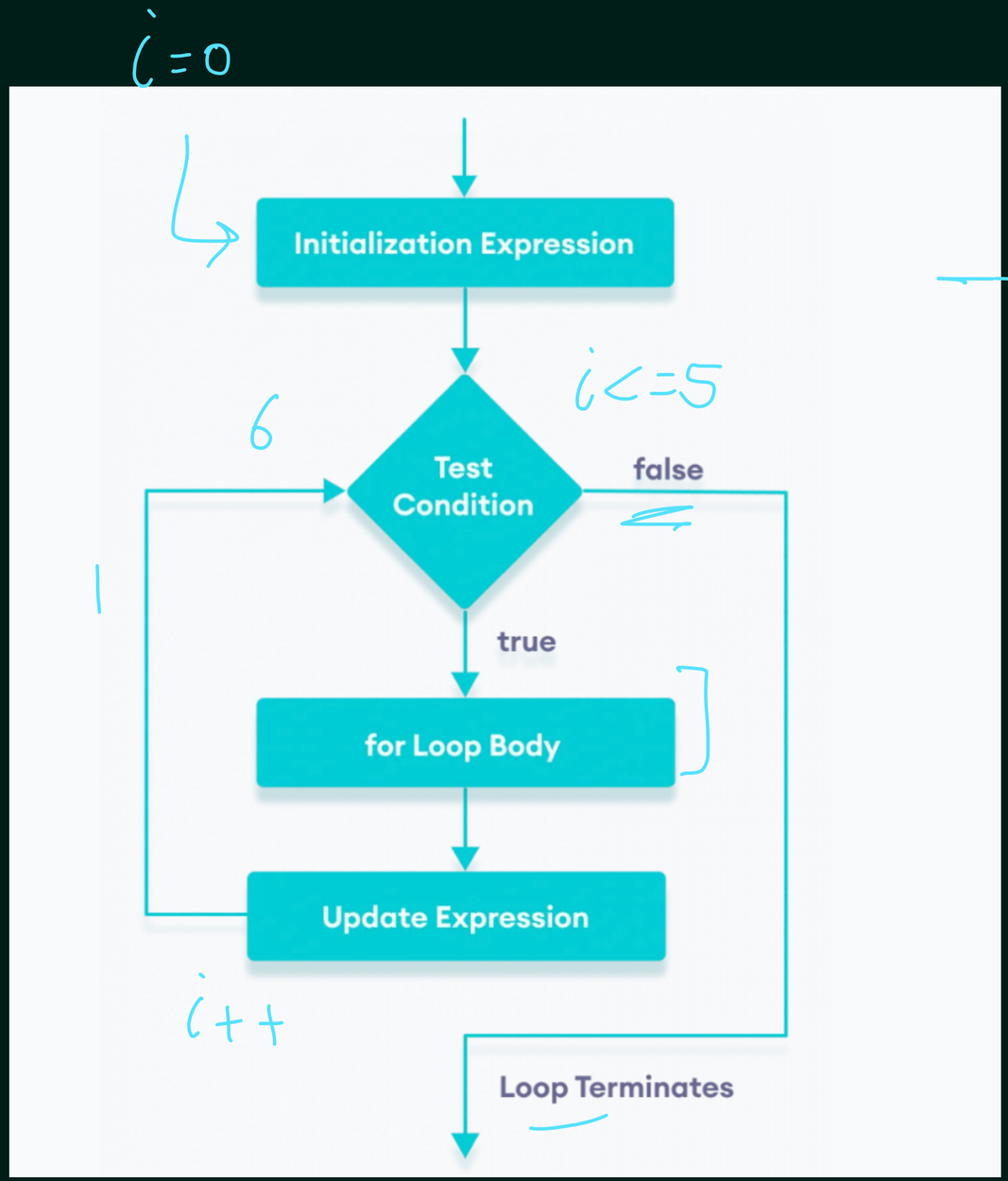
Java for Loops

① Table of n

3 x 1 = 3
3 * 2 = 6
⋮

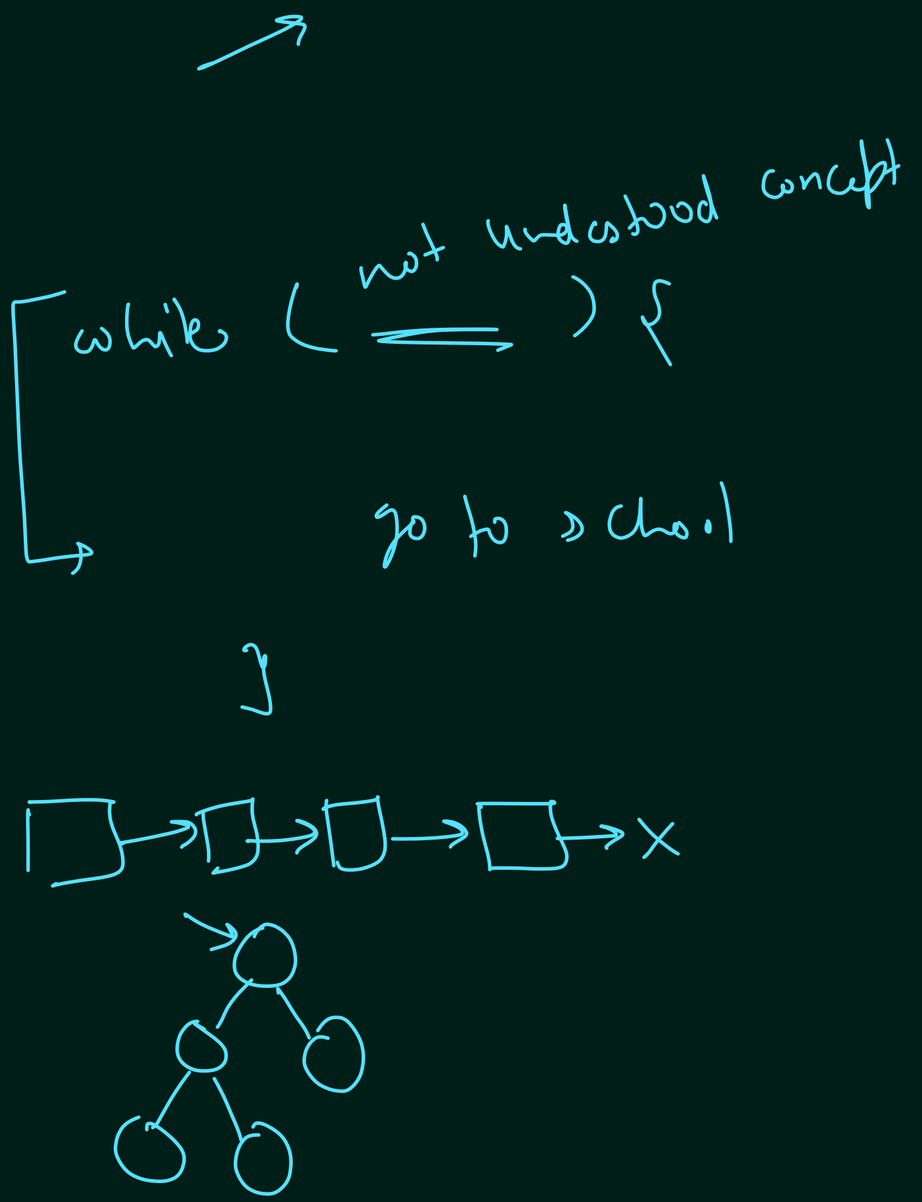
② Sum of N natural numbers

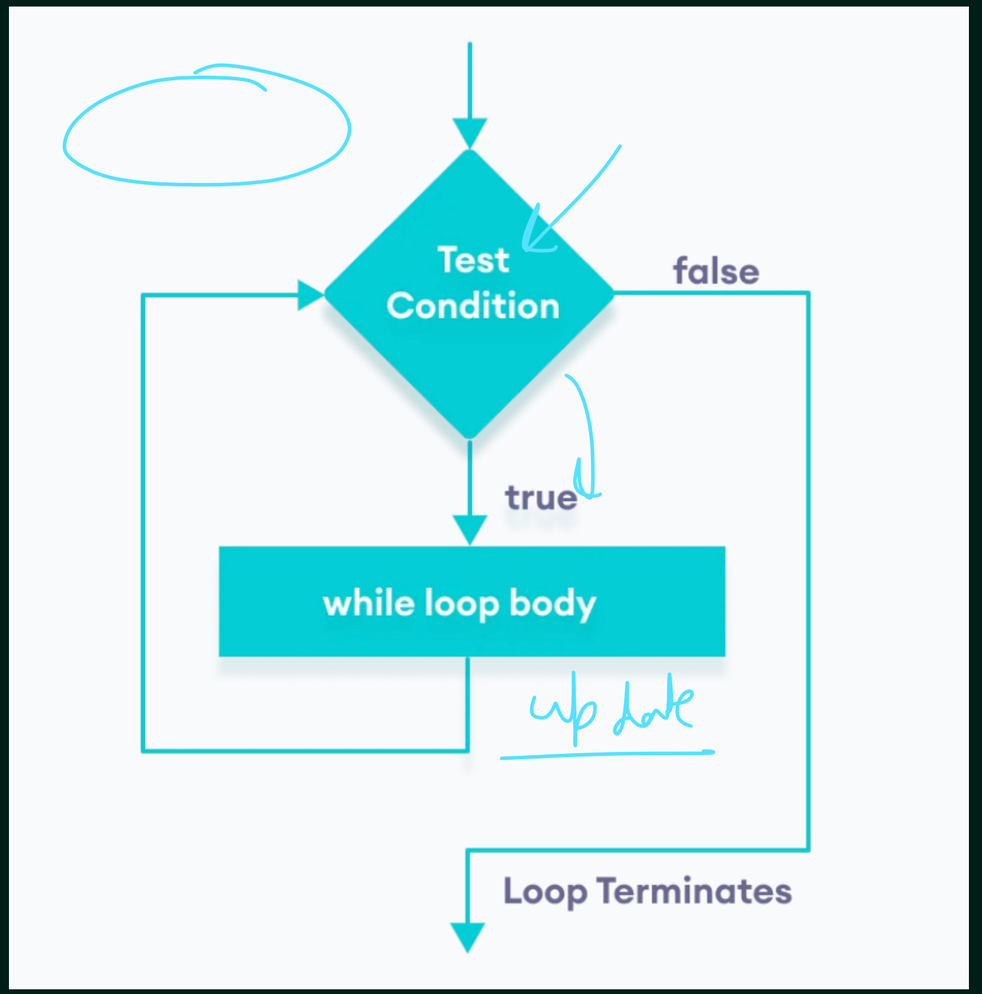
$= \frac{N(N+1)}{2}$



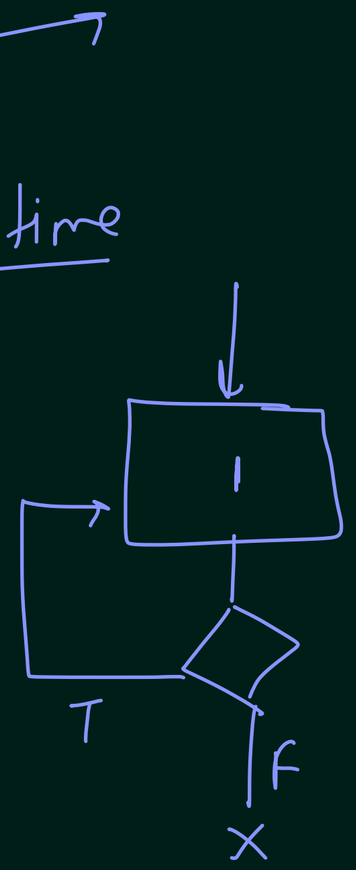
n = 5
(1 + 2 + 3 + 4 + 5)
→ 2 + 4 + 6 + 8 + 10

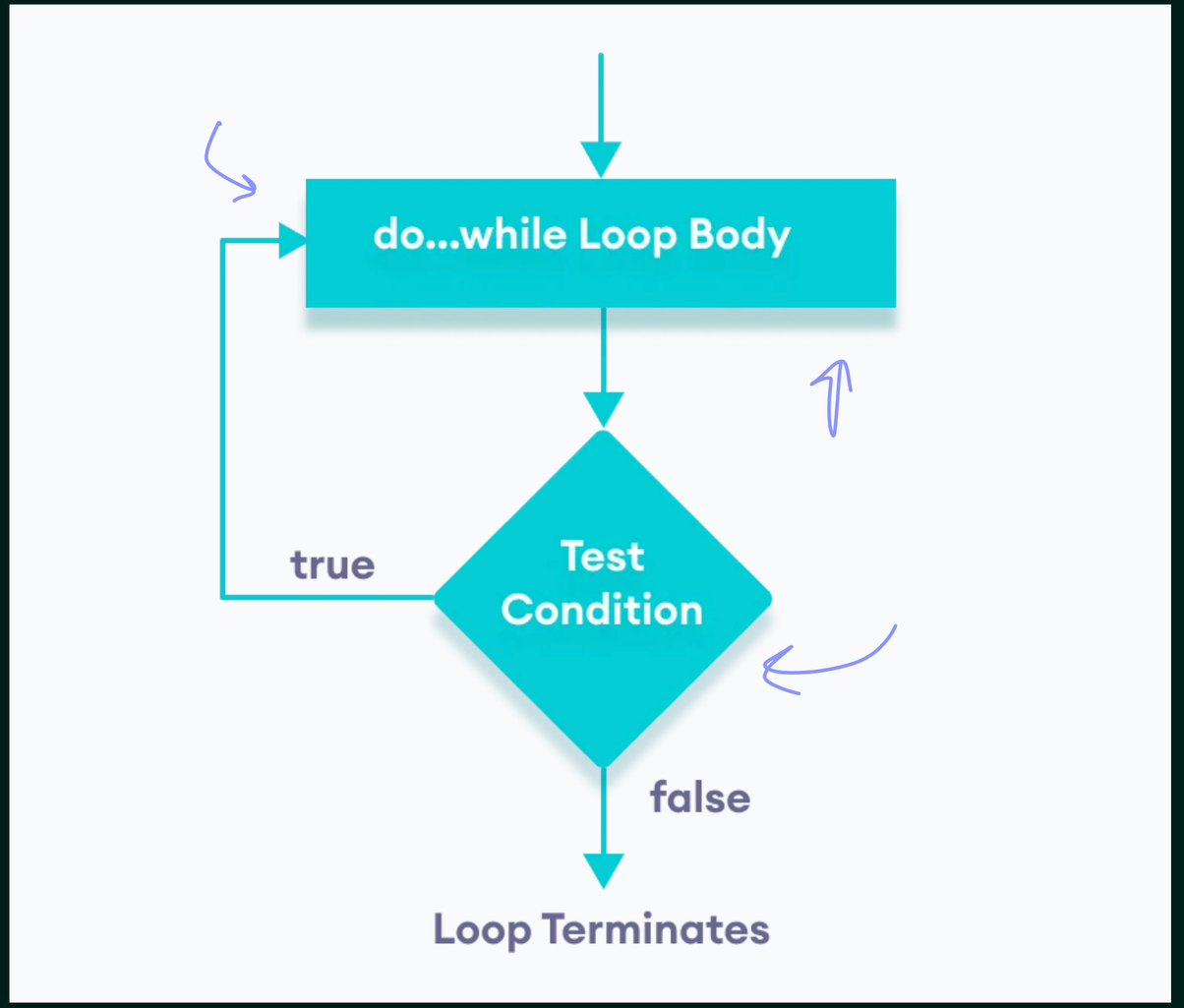
Java while Loops





Java do-while Loops





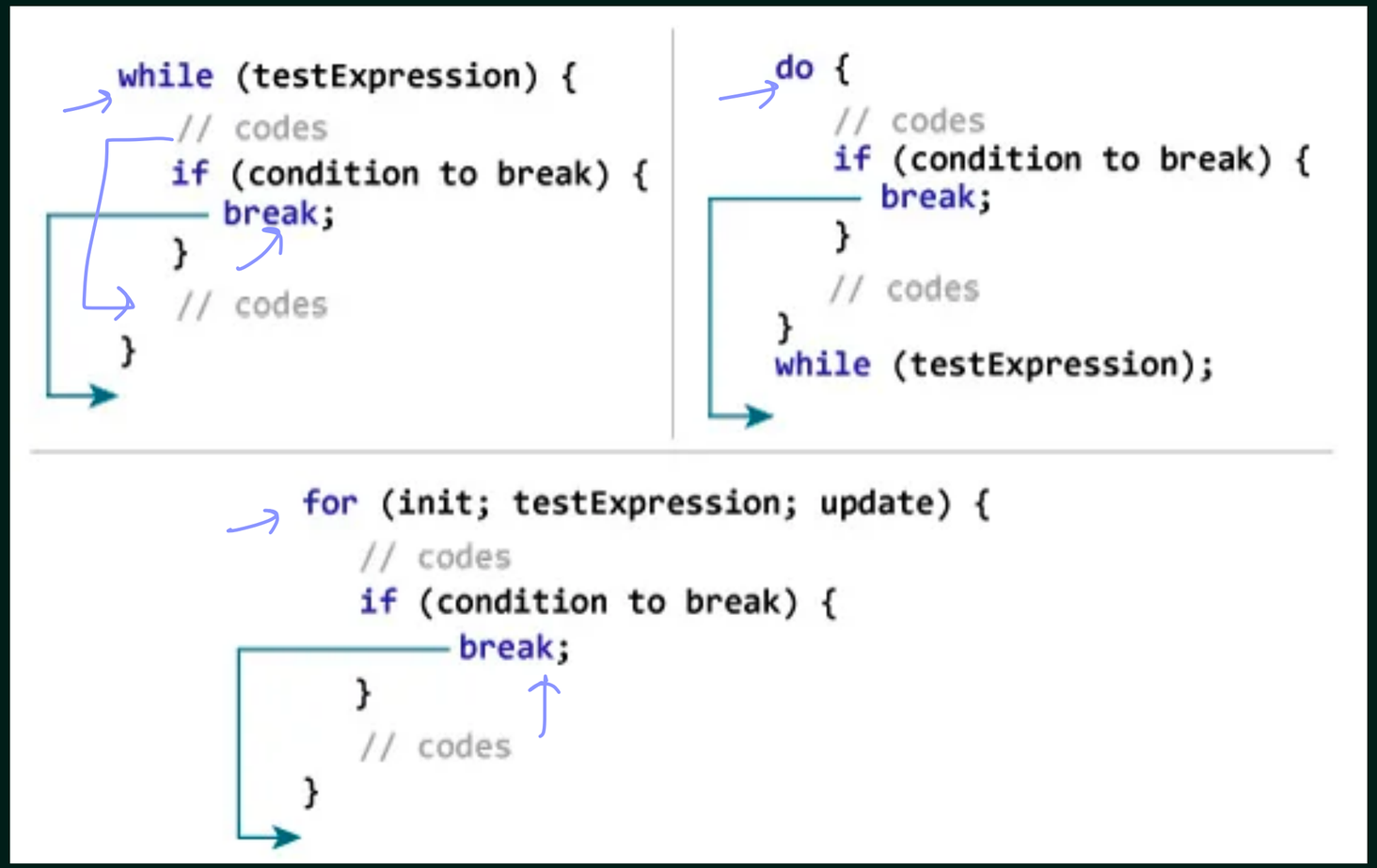
break statements in Java

switch

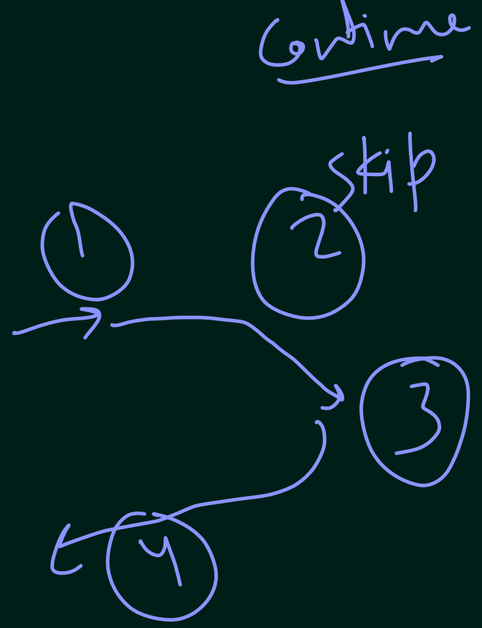
break

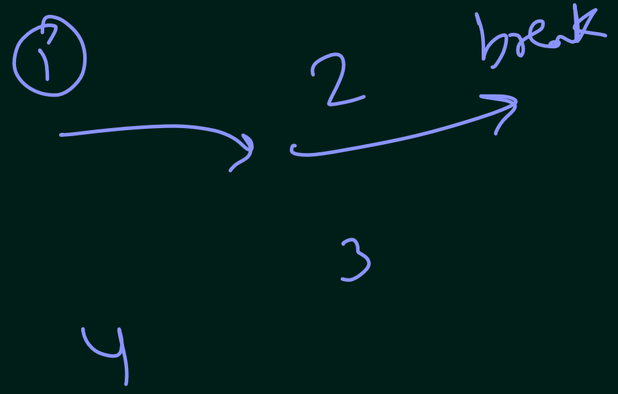
{

}



continue statements in Java





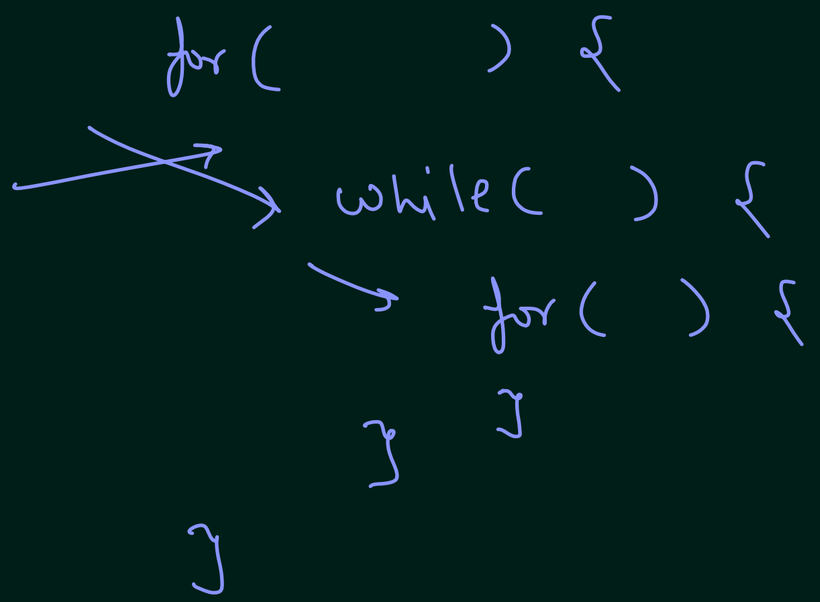

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

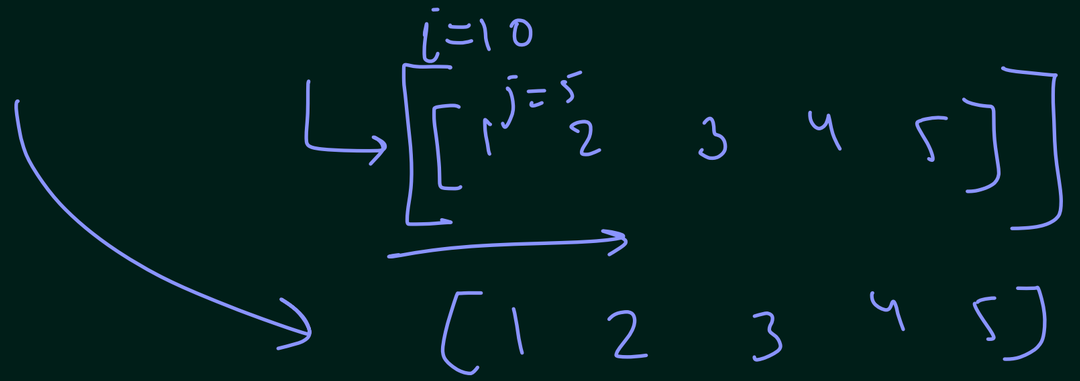
```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

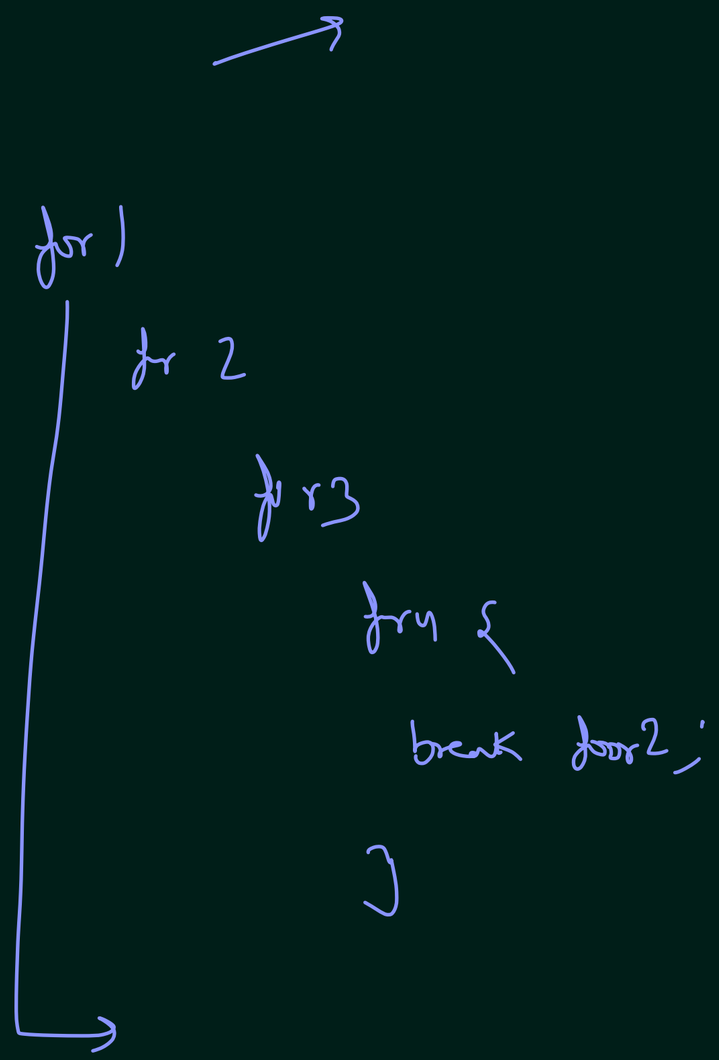
Nested loops

If a loop exists inside the body of another loop, it's called a nested loop.





Labeled break & continue statements




```
label:
while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (testExpression) {
            continue label;
        }
        // codes
    }
    // codes
}
```

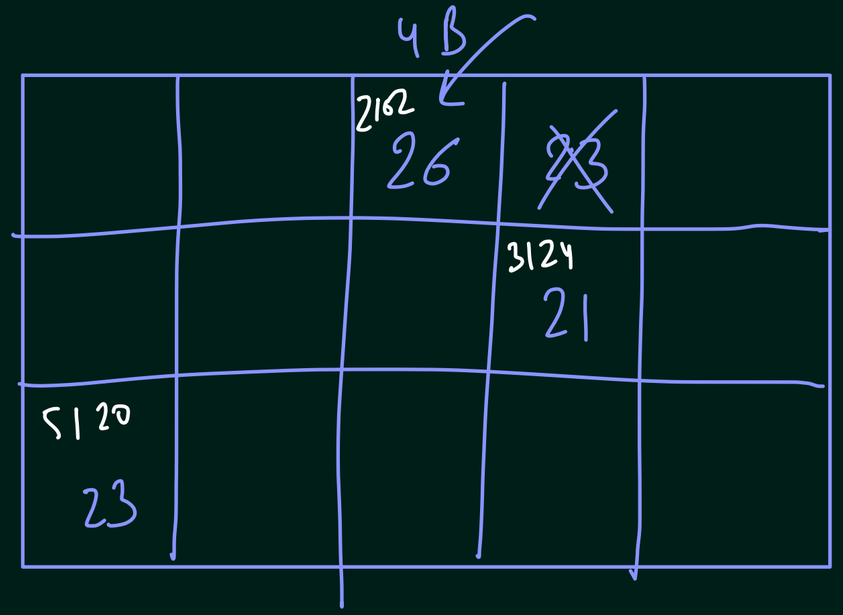
Arrays in Java

In This Lecture

1. How do the Arrays work?
2. Creating and Declaring Arrays
3. for-each loop
4. Multi-dimensional Arrays

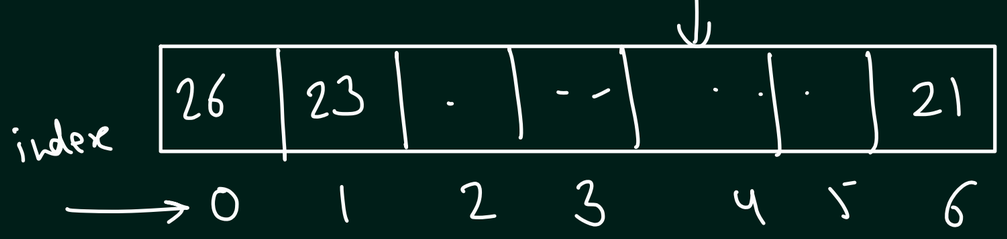
How do the Arrays work?

Arrays are stored in contiguous memory [consecutive memory locations].




```
int age1 = 26;
int age2 = 23;
:
:
int age10 = 21;
```

Arrays → Same Type data



length = 7

Creating an Array

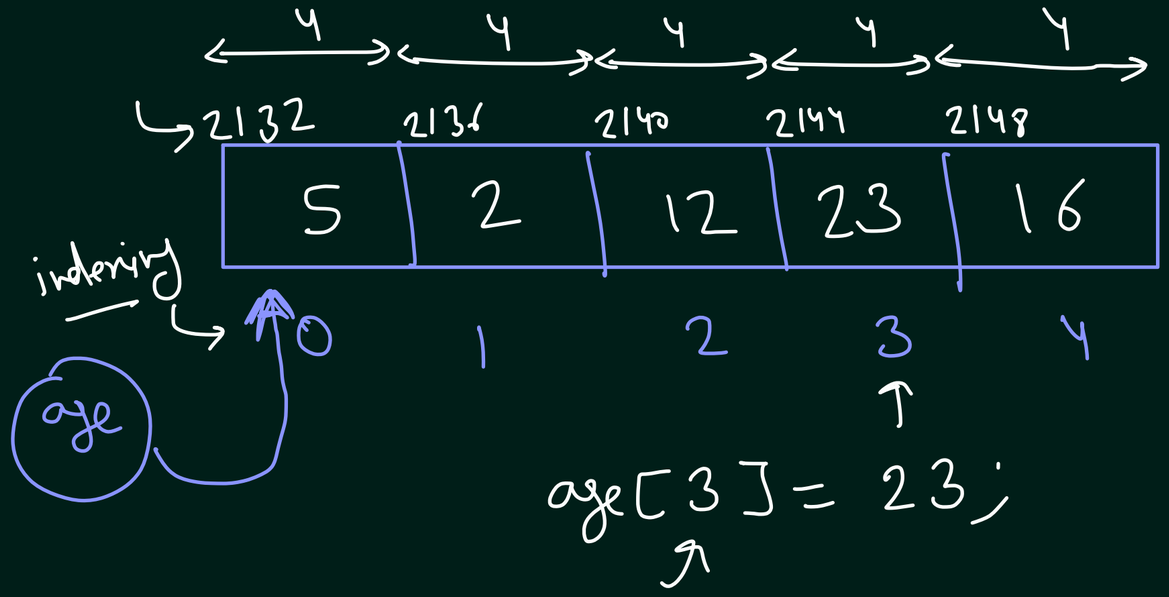
```
// both are valid declarations
int intArray[];
or int[] intArray;

intArray = new int[20]; // allocating memory to array
```

```
int age;  // Declaration
age = 23; // Initialisation
```

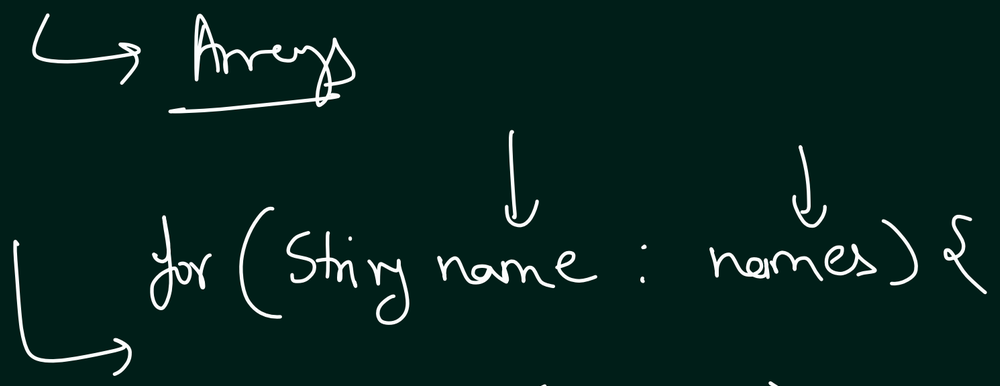
address of zero + index * size

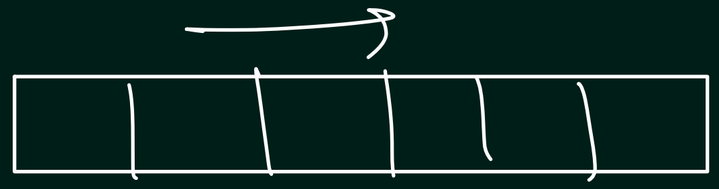
2132 + 3 * 4

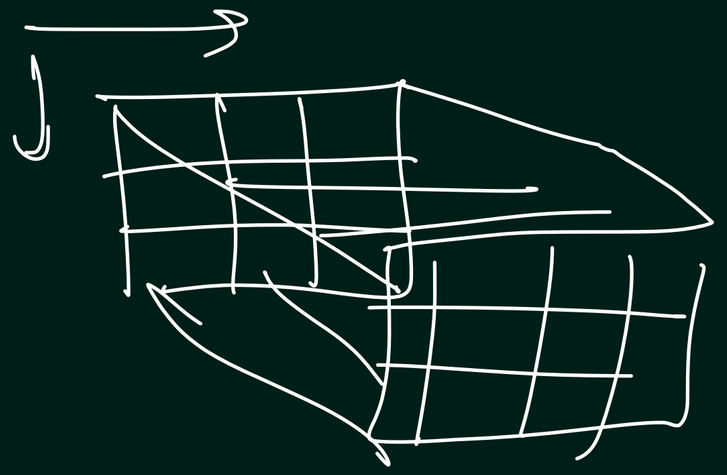
2144



for-each Loop





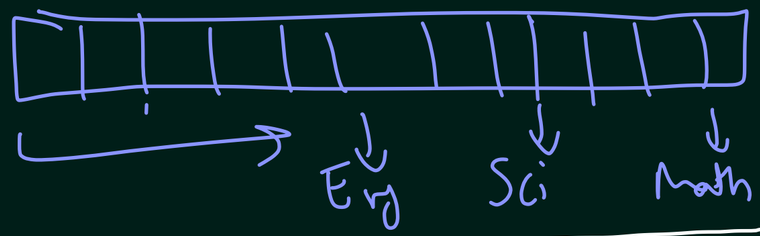


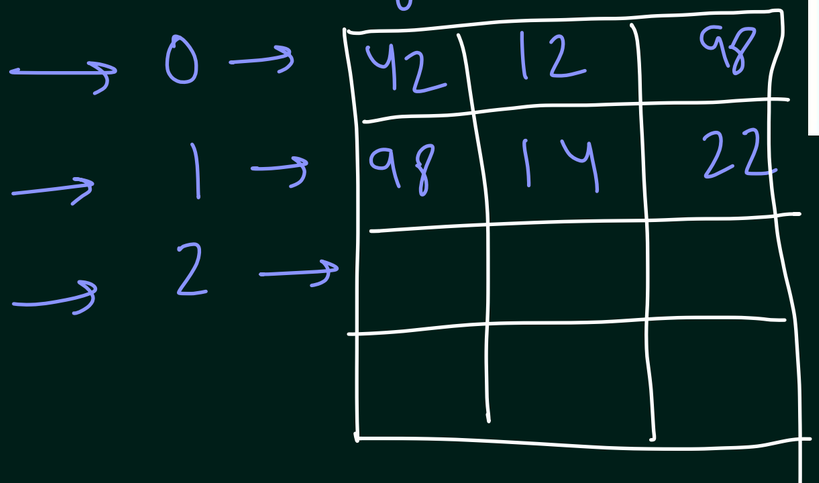
Multi-dimensional Array

2-D array / Matrix

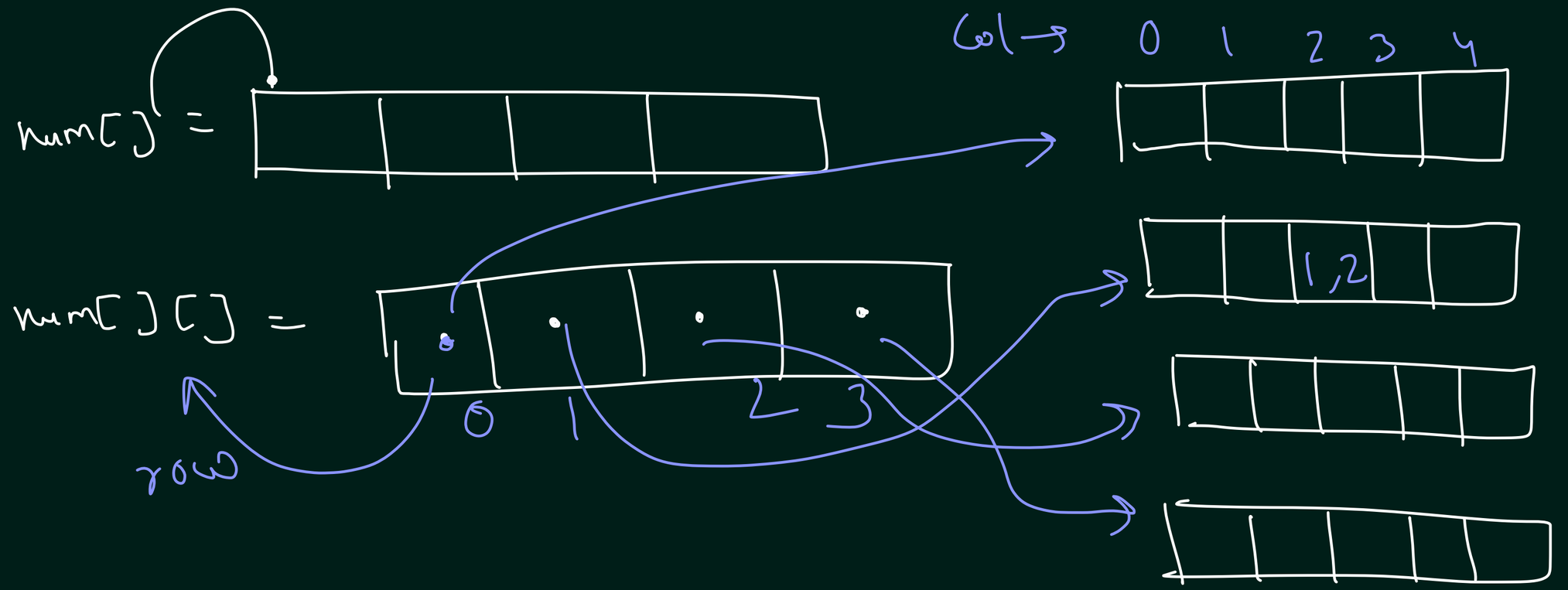
```
int a[][] = new int[3][4]
```

3 x 4 = 12





	Column 1 (0)	Column 2 (1)	Column 3 (2)	Column 4 (3)
Row 1 (0)	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2 (1)	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3 (2)	a[2][0]	a[2][1]	a[2][2]	a[2][3]


num[] =
num[][] =
row
col →
0 1 2 3 4
1,2
0 1 2 3

CODING SHUTTLE

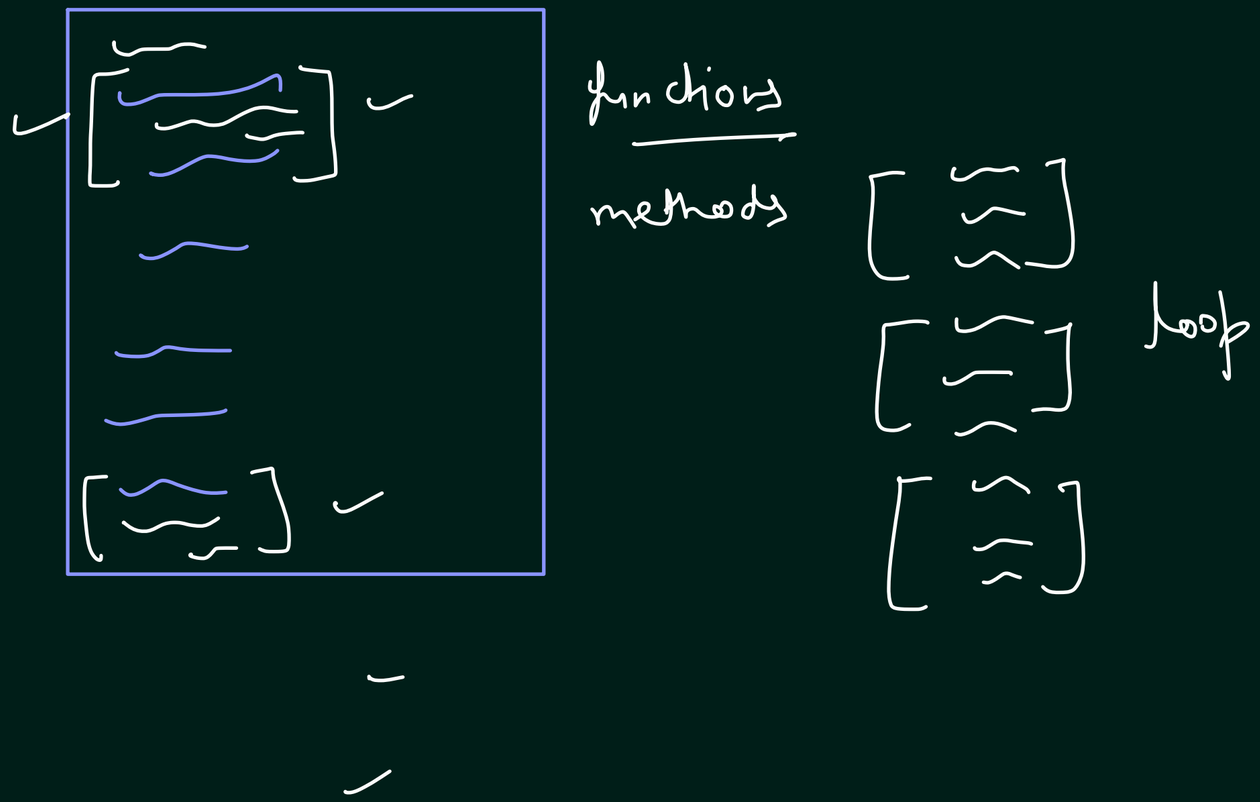
Methods in Java

In This Lecture

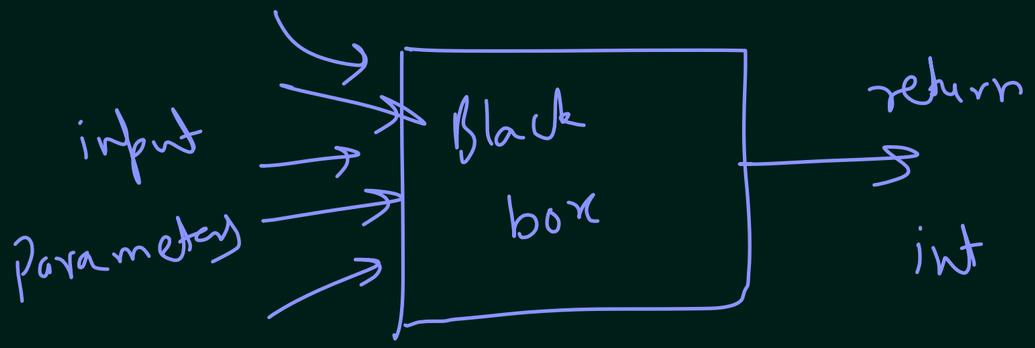
1. How do the functions work?
2. Declaring a Java Method
3. Calling a Method
4. Method return type
5. Method Parameters
6. Math Library Methods

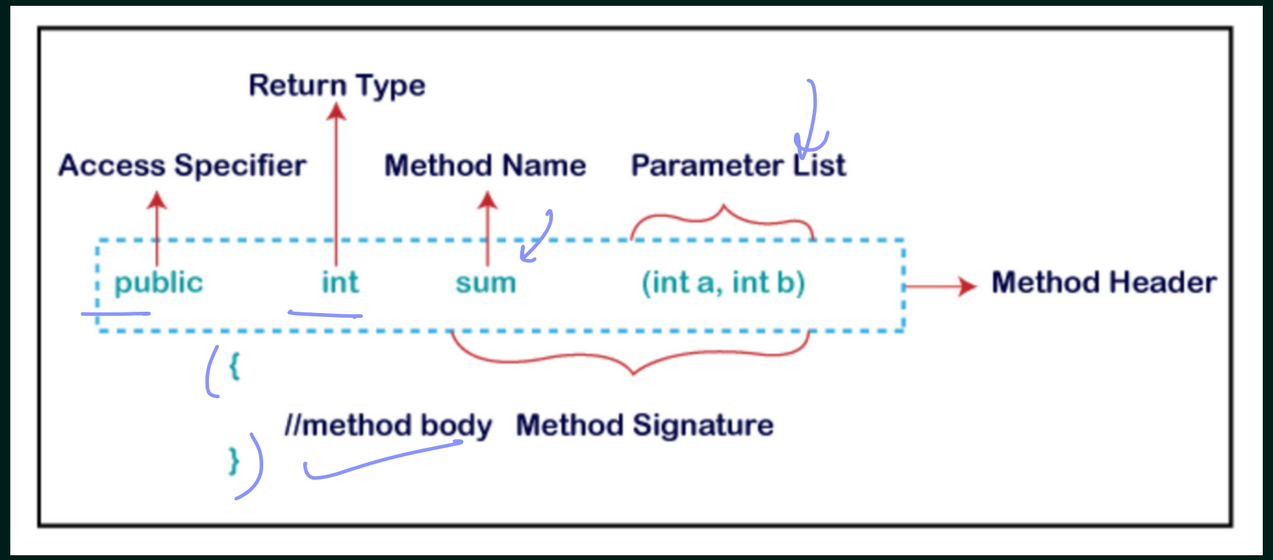
How do the Methods work?

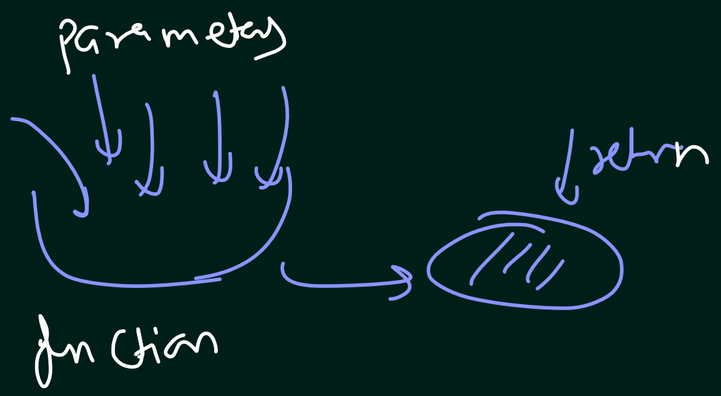
A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code.



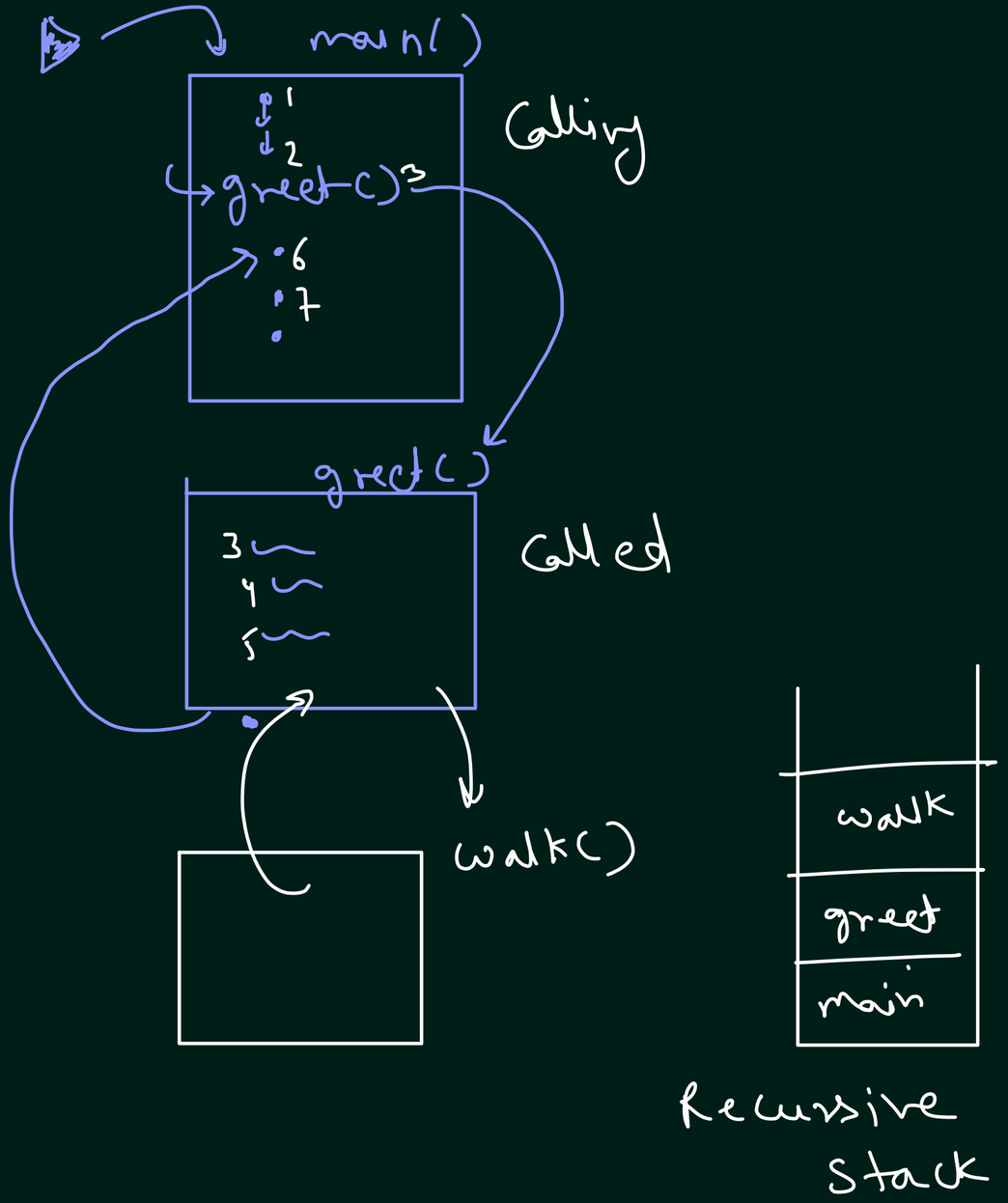
Components of a Method







Calling a Method



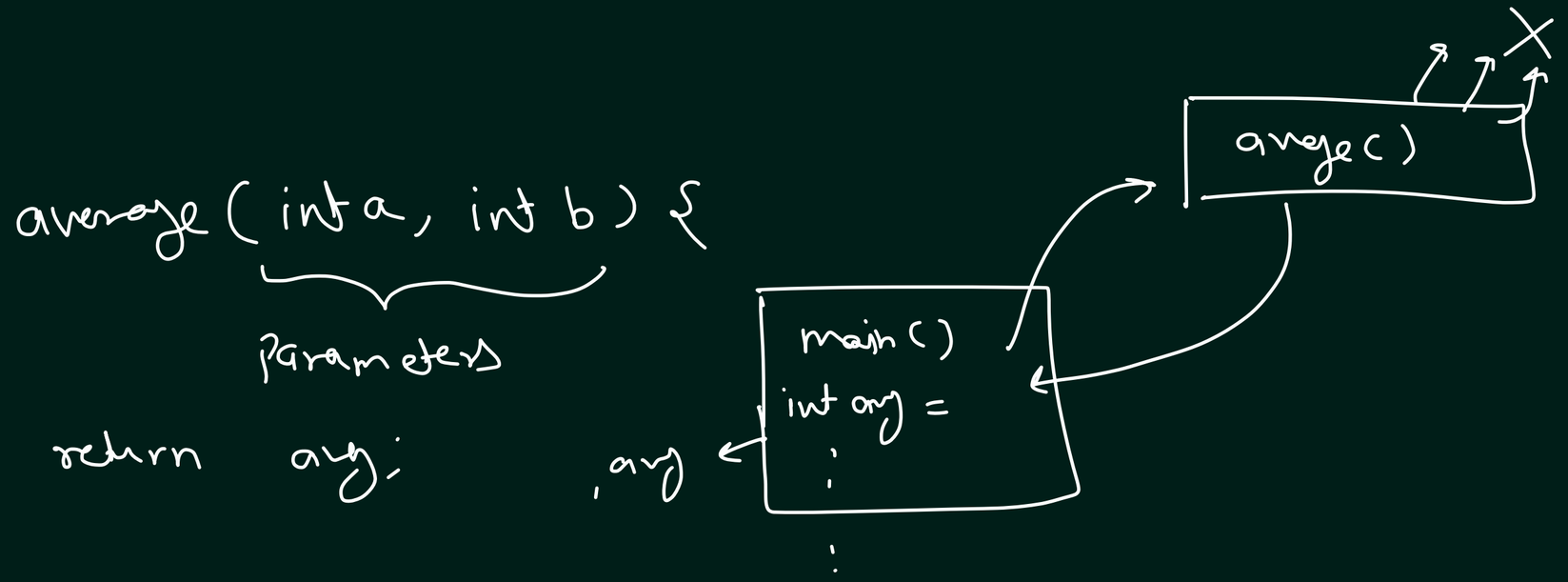
Recursive Stack

```
int addNumbers() {
  // code
}
...
...
addNumbers();
// code
```

method call

Method Parameters

A method parameter is a value accepted by the method. As mentioned earlier, a method can also have any number of parameters.



Math Class Methods

1. Math.min(x,y)
2. Math.max(x,y)
3. Math.sqrt(x)
4. Math.pow(x,y)
5. Math.abs(x)
6. Math.random() [0, 1)
7. Math.floor(x)
8. Math.ceil(x)
9. Math.round(x)

Find the complete list here

(a, b)

Math.Random()

[a + [0 ---- 1] * 6]

1 + [0, 1.2, 3, 6] = [1, 6]

CODING SHUTTLE

Java String

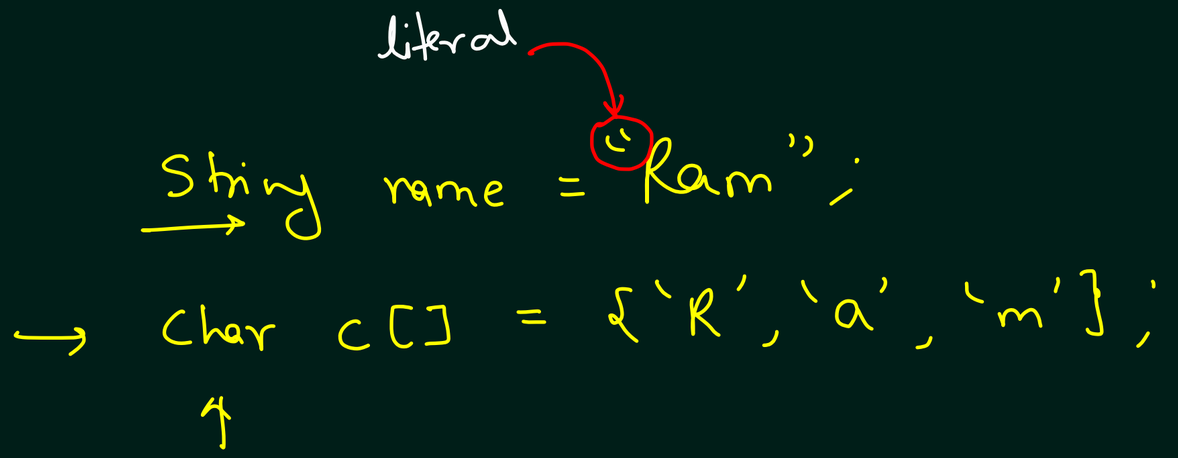
In This Lecture

1. Basics of String
2. How to create String in Java
3. How Strings are stored in Java
4. Immutability in Strings
5. Comparing two Strings in Java
6. Java String Methods

Java String

In Java, String is basically an object that represents a sequence of char values.

→ An array of characters works same as Java String.



How to create Java String

1. By string literal
2. By new keyword

Scanner sc = new Scanner(System.in);

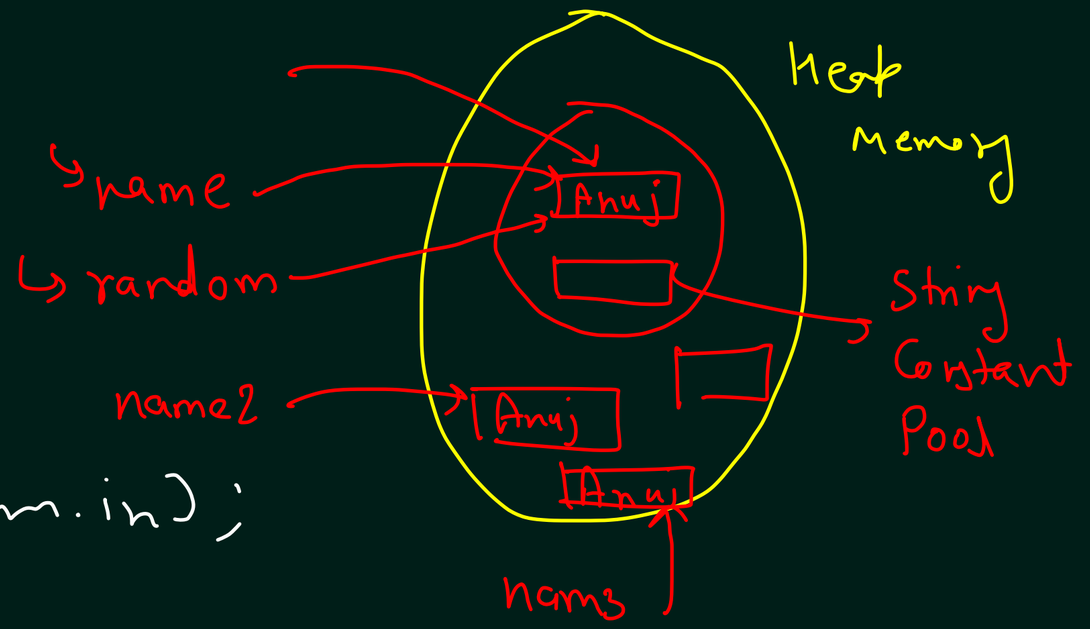
String name2 = new String("Anuj");

String name = "Anuj";

String random = "Anuj";

Stack Memory — Reference

Heap Memory

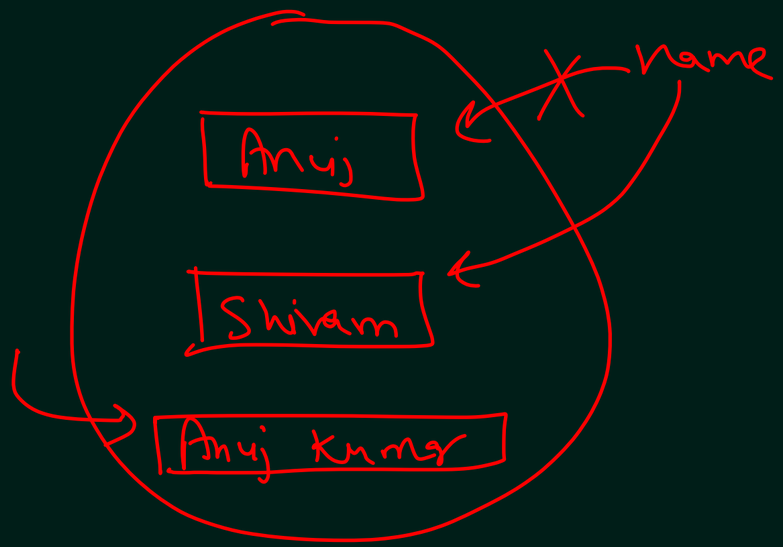


Immutable String in Java

String objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once a String object is created its data or state can't be changed

```
String name = "Anuj";
name = "Shivam"
name = name + "Kumar"
```



Comparing Two Strings in Java

The String class **equals()** method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

- public boolean **equals**(Object another) compares this string to the specified object.
- public boolean **equalsIgnoreCase**(String another) compares this string to another string, ignoring case.

Java String Methods

1. toUpperCase()
2. toLowerCase()
3. trim()
4. startsWith()
5. endsWith()
6. equals()
7. equalsIgnoreCase()
8. charAt()
9. valueOf()
6. replace()
7. contains()
8. substring()
9. split()
10. toCharArray()
11. isEmpty()

CODING SHUTTLE

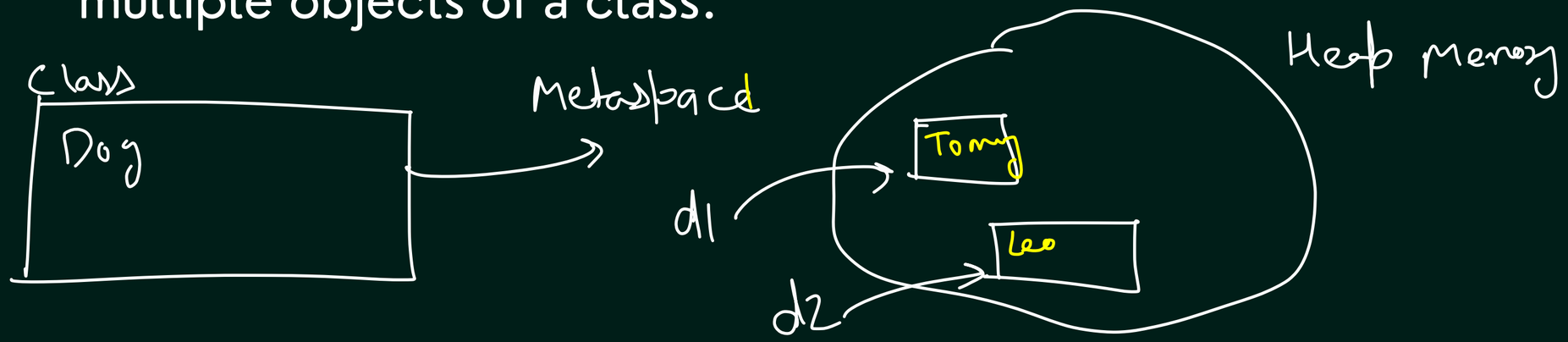
OOPS - 1

In This Lecture

1. Classes & Objects
2. Constructors
3. Method + Constructor Overloading
4. this keyword in Java

Classes & Objects

1. Class is a blueprint which defines some properties and behaviors. An object is an instance of a class which has those properties and behaviours attached.
2. A class is not allocated memory when it is defined. An object is allocated memory when it is created.
3. Class is a logical entity whereas objects are physical entities.
4. A class is declared only once. On the other hand, we can create multiple objects of a class.



Classes & Objects

5. A class is a way to arrange data and behavior information. It is a template that must be implemeted by its objects.
6. A class can also be seen as a user-defined data type where any object of defined data type has some predefined properties and behaviors.

Method Overloading

1. Two or more methods can have the same name inside the same class if they accept different arguments. This feature is known as method overloading.
2. Method overloading is achieved by either:
    a. changing the number of arguments.
    b. or changing the data type of arguments.
3. It is not method overloading if we only change the return type of methods. There must be differences in the number of parameters.

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

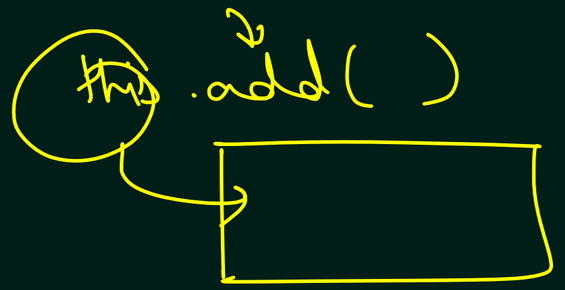
Constructors

1. Constructors are invoked implicitly when you instantiate objects.
2. The two rules for creating a constructor are:
    a. The name of the constructor should be the same as the class.
    b. A Java constructor must not have a return type.
3. If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time. The default constructor initializes instance variables with default values.
4. Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
5. A constructor cannot be abstract or static or final.
6. A constructor can be overloaded but can not be overridden.

The this keyword

In Java, this keyword is used to refer to the current object inside a method or a constructor.

We mostly use this keyword to remove any Ambiguity in Variable Names. We can also use this to invoke methods of the current class or to invoke a constructor of the current class.



CODING SHUTTLE

OOPS - 2

In This Lecture

1. Java Inheritance
2. Method overriding
3. super keyword
4. this vs super keyword
5. final keyword

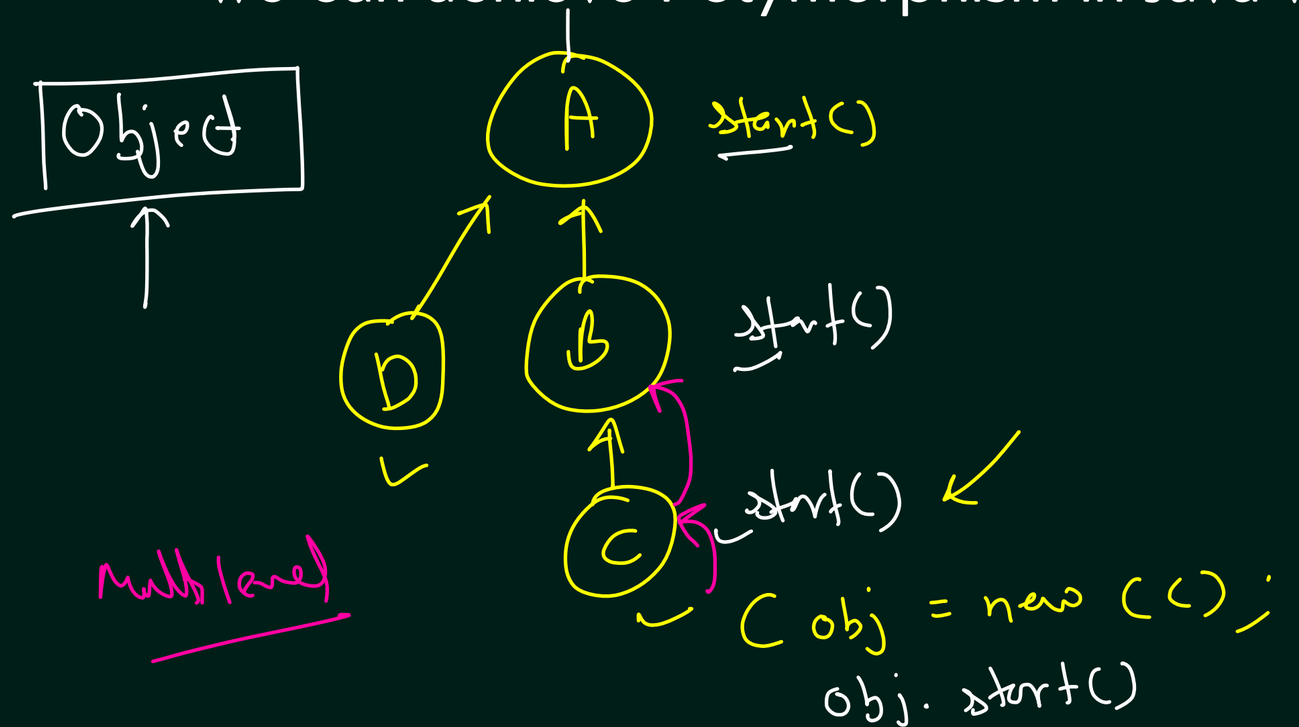
Java Inheritance

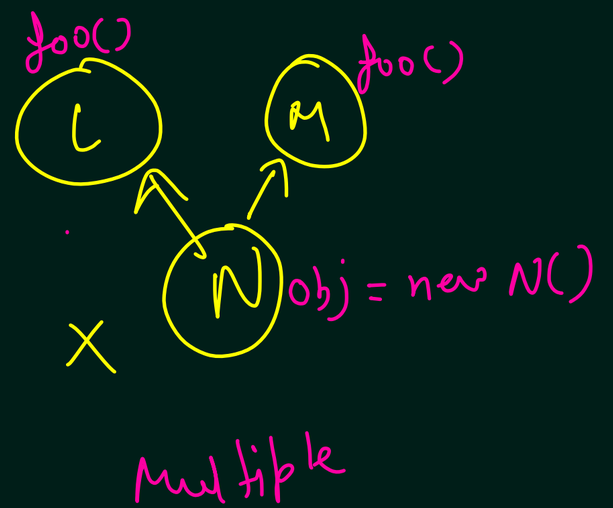
Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.
The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).

Method Overriding

If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

- Method overriding is also known as runtime polymorphism. Hence, we can achieve Polymorphism in Java with the help of inheritance.





super Keyword in Java

super is a special keyword in Java that is used to refer to the instance of the immediate parent class.

Uses of super Keyword in Java

- It is used to refer to an instance variable of the immediate parent class.
- It is used to invoke a method of the immediate parent class.
- It is used to invoke a constructor of immediate parent class.

this vs super

this keyword in Java	super keyword in Java
this is an implicit reference variable keyword used to represent the current class.	super is an implicit reference variable keyword used to represent the immediate parent class.
this is to invoke methods of the current class.	super is used to invoke methods of the immediate parent class.
this is used to invoke a constructor of the current class.	super is used to invoke a constructor of the immediate parent class.
this refers to the instance and static variables of the current class.	super refers to the instance and static variables of the immediate parent class.
this can be used to return and pass as an argument in the context of a current class object.	super can be used to return and pass as an argument in the context of an immediate parent class object.

final Keyword in Java

In Java, the final keyword is a non-access modifier that is used to define entities that cannot be changed or modified.

Type	Description
Final Variable	Variable with final keyword cannot be assigned again
Final Method	Method with final keyword cannot be overridden by its subclasses
Final Class	Class with final keywords cannot be extended or inherited from other classes

CODING SHUTTLE

OOPS - 3

In This Lecture

1. Java Packages
2. Access modifiers
3. Java Encapsulation
4. Data Hiding
5. The static keyword

Java Packages

A package is simply a container that groups related types (Java classes, interfaces, enumerations, and annotations).

To define a package in Java, you use the keyword package.

Java uses file system directories to store packages.

For example:

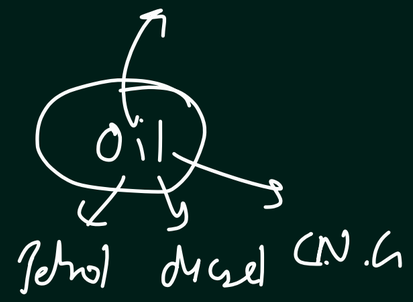
```
└── com
    └── test
        └── Test.java
```

Test.java

```java
package com.test;

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

Importing a Package

- Java has an import statement that allows you to import an entire package (as in earlier examples), or use only certain classes and interfaces defined in the package.




```
import java.util.Date; // imports only Date class
import java.io.*;      // imports everything inside java.io package
```

- In Java, the import statement is written directly after the package statement (if it exists) and before the class definition.

```
package package.name;
import package.ClassName; // only import a Class

class MyClass {
    // body
}
```

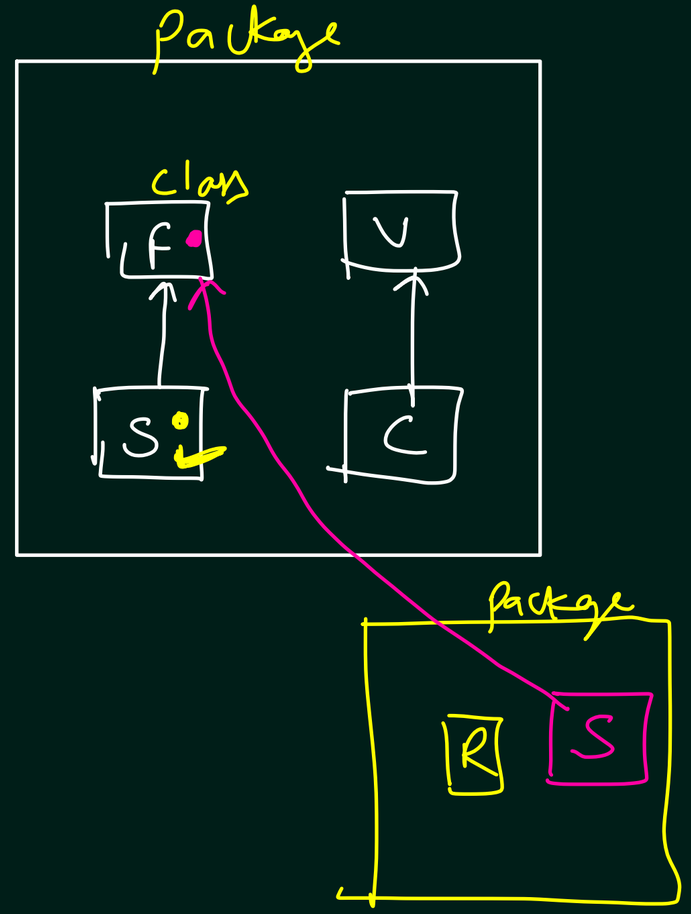
Java Access Modifiers

In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods. For example,

```
class Animal {
    public void method1() {...}
      ^
   private void method2() {...}
      ^
}
```

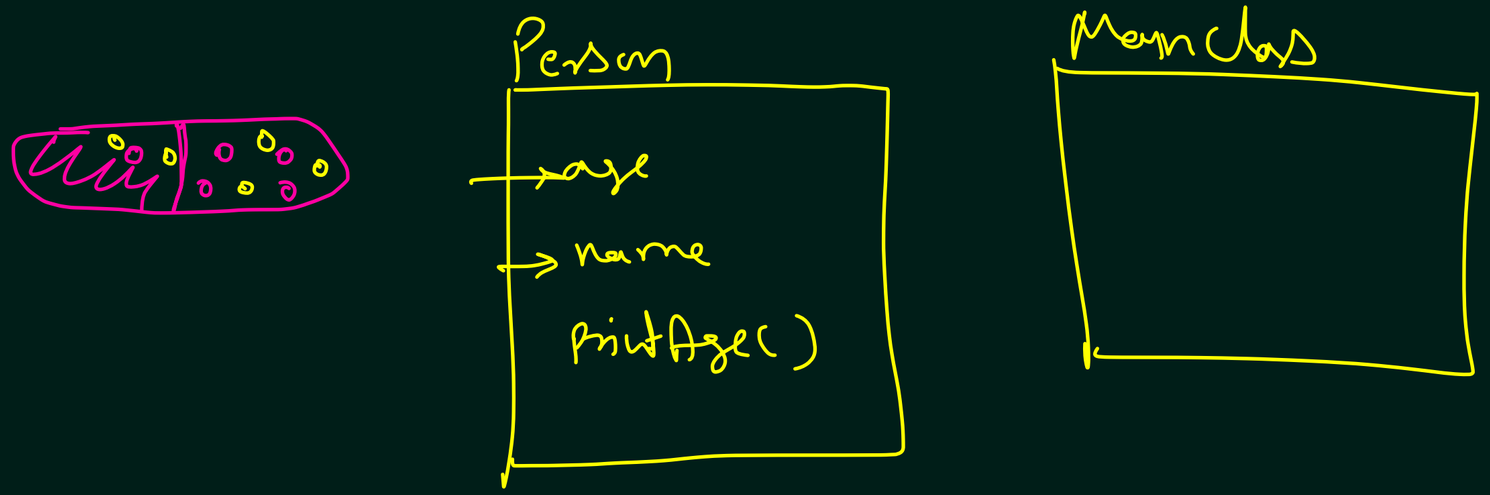
Types of Access Modifiers

Access Modifier	Same class	Same package subclass	Same package non-subclass	Difference Package subclass	Different package non-subclass
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No
Protected	Yes	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes	Yes



Java Encapsulation

Encapsulation refers to the bundling of fields and methods inside a single class. It prevents outer classes from accessing and changing fields and methods of a class. This also helps to achieve data hiding.



Data Hiding

Data hiding is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding.

We can use access modifiers to achieve data hiding.

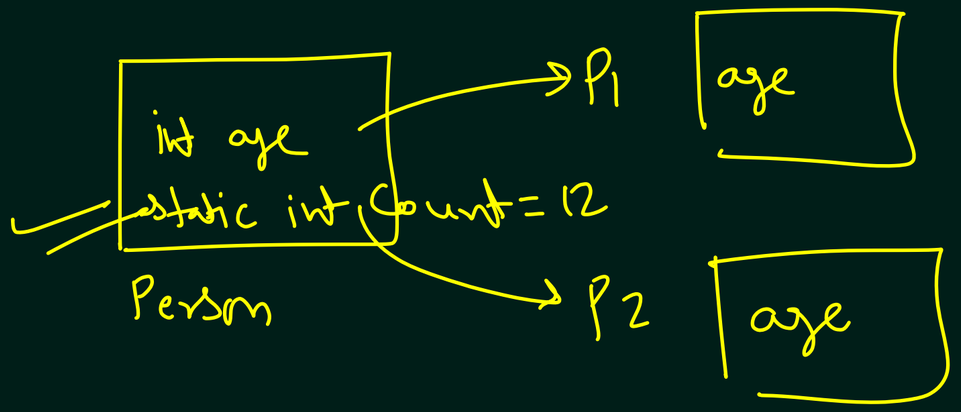
Note: People often consider encapsulation as data hiding, but that's not entirely true. Encapsulation refers to the bundling of related fields and methods together. This can be used to achieve data hiding.

Encapsulation in itself is not data hiding.

The static keyword

If we want to access class members without creating an instance of the class, we need to declare the class members static.

Static variables can be accessed by calling the class name of the class. There is no need to create an instance of the class for accessing the static variables because static variables are the class variables and are shared among all the class instances.



The static keyword

Static Variables

- Only a single copy of the static variable is created and shared among all the instances of the class.
- Because it is a class-level variable, memory allocation of such variables only happens once when the class is loaded in the memory.
- If an object modifies the value of a static variable, the change is reflected across all objects.
- Static variables can be used in any type of method: static or non-static.
- Non-static variables cannot be used inside static methods. It will throw a compile-time error.

The static keyword

Static Methods

- The static members and methods belong to the class rather than the instance of the class. When the implementation of the particular method is not dependent on the instance variables and instance methods, In this case, we can make that method to be static.
- They are accessed by the name of the class.
- The keywords such as this and super are not used in the body of the static method.
- The modification of the static field value is not allowed.

CODING SHUTTLE

OOPS - 4

In This Lecture

1. The abstract keyword
2. Abstraction
3. Java Interfaces
4. Inner class & Nested static classes
5. Anonymous Classes
6. Functional Interfaces
7. Lambda expressions

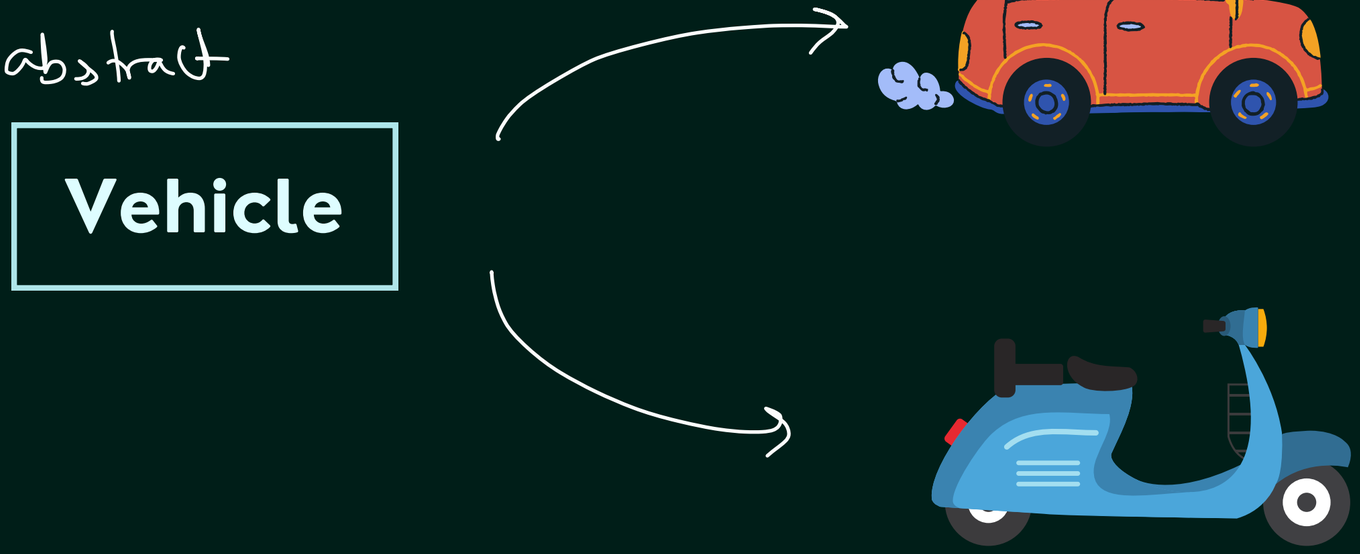
Java Abstract Class

The abstract class in Java cannot be instantiated (we cannot create objects of abstract classes). We use the abstract keyword to declare an abstract class.

- An abstract class can have both the regular methods and abstract methods.
- A method that doesn't have its body is known as an abstract method.
- Though abstract classes cannot be instantiated, we can create subclasses from it. We can then access members of the abstract class using the object of the subclass.
- If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method.

Java Abstraction

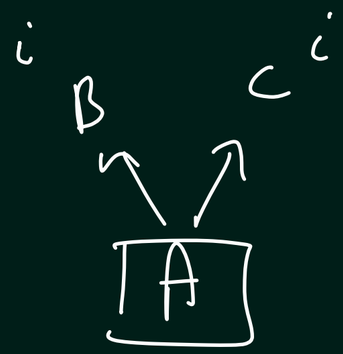
Abstraction is an important concept of object-oriented programming that allows us to hide unnecessary details and only show the needed information.
This allows us to manage complexity by omitting or hiding details with a simpler, higher-level idea.



Java Interfaces

An interface is a fully abstract class. It includes a group of abstract methods (methods without a body).
We use the interface keyword to create an interface in Java.

Like abstract classes, we cannot create objects of interfaces.
To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

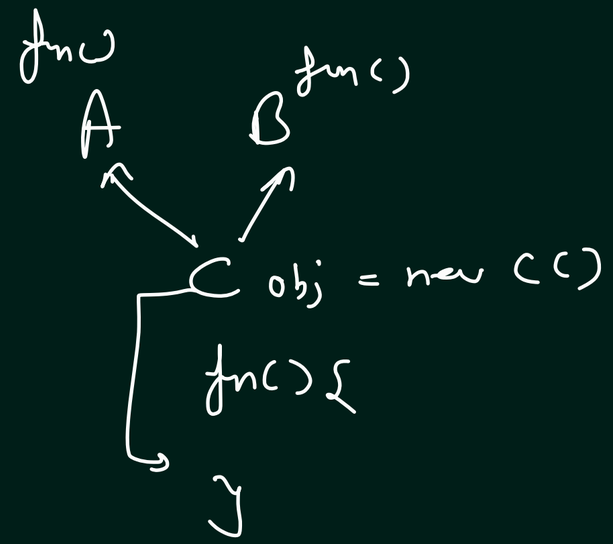



```
interface Language {
  public void getType();

  public void getVersion();
}
```

Advantages of Java Interfaces

- Similar to abstract classes, interfaces help us to achieve abstraction in Java.
- Interfaces are also used to achieve multiple inheritance in Java.
- Note: All the methods inside an interface are implicitly public and all fields are implicitly public static final.

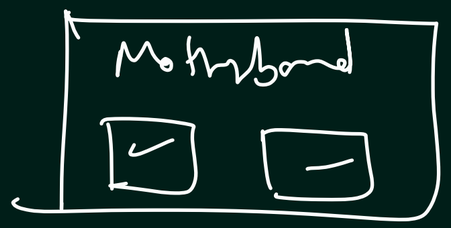
```
interface Line {
…
}

interface Polygon {
…
}

class Rectangle implements Line, Polygon {
…
}
```



Inner classes and Nested Static Class in Java

A non-static nested class is a class within another class. It has access to members of the enclosing class (outer class). It is commonly known as inner class.
Since the inner class exists within the outer class, you must instantiate the outer class first, in order to instantiate the inner class.

Unlike inner class, a static nested class cannot access the member variables of the outer class. It is because the static nested class doesn't require you to create an instance of the outer class.



Using the nested class makes your code more readable and provide better encapsulation.

Anonymous Classes in Java

In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
A nested class that doesn't have any name is known as an anonymous class.

Anonymous classes usually extend subclasses or implement interfaces.
Here, Type can be

1. a superclass that an anonymous class extends
2. an interface that an anonymous class implements

Functional Interfaces

An Interface that contains exactly one abstract method is known as a functional interface.

Functional Interfaces introduced in Java 8 allow us to use a lambda expression to initiate the interface's method and avoid using lengthy codes for the anonymous class implementation.

```
// interface
@FunctionalInterface
interface Sample{
    // abstract method
    int calculate(int val);
}
```

Lambda Expression

```
(int x) -> x+1                    // Single declared-type argument
(int x) -> { return x+1; }        // same as above
(x) -> x+1                        // Single inferred-type argument, same as below
x -> x+1                          // Parenthesis optional for single inferred-type case

(String s) -> s.length()          // Single declared-type argument
(Thread t) -> { t.start(); }      // Single declared-type argument
s -> s.length()                   // Single inferred-type argument
t -> { t.start(); }               // Single inferred-type argument

(int x, int y) -> x+y             // Multiple declared-type parameters
(x,y) -> x+y                      // Multiple inferred-type parameters
```

CODING SHUTTLE

Week 5 LIVE

Java Memory, Polymorphism and Object Class

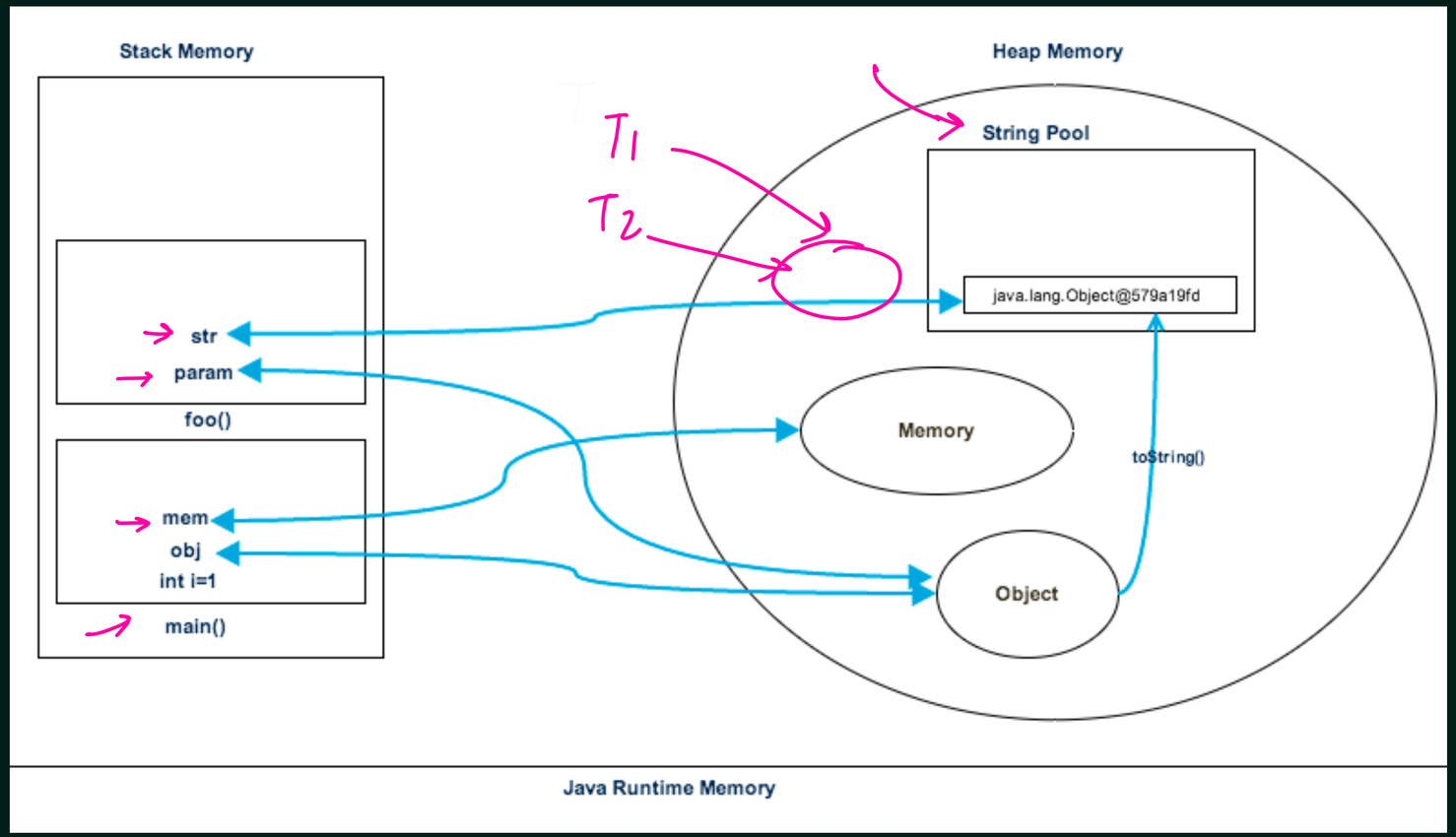
In This Lecture

1. How Java Memory Works
2. Java Object Class
3. Java Polymorphism

Java Heap And Stack Memory

↳ Garbage Collector

```
public class Memory {

    public static void main(String[] args) { // Line 1
        int i=1; // Line 2
        Object obj = new Object(); // Line 3
        Memory mem = new Memory(); // Line 4
        mem.foo(obj); // Line 5
    } // Line 9

    private void foo(Object param) { // Line 6
        String str = param.toString(); //// Line 7
        System.out.println(str);
    } // Line 8

}
```



Java Heap Memory

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create an object, it's always created in the Heap space.

Any object created in the heap space has global access and can be referenced from anywhere of the application.

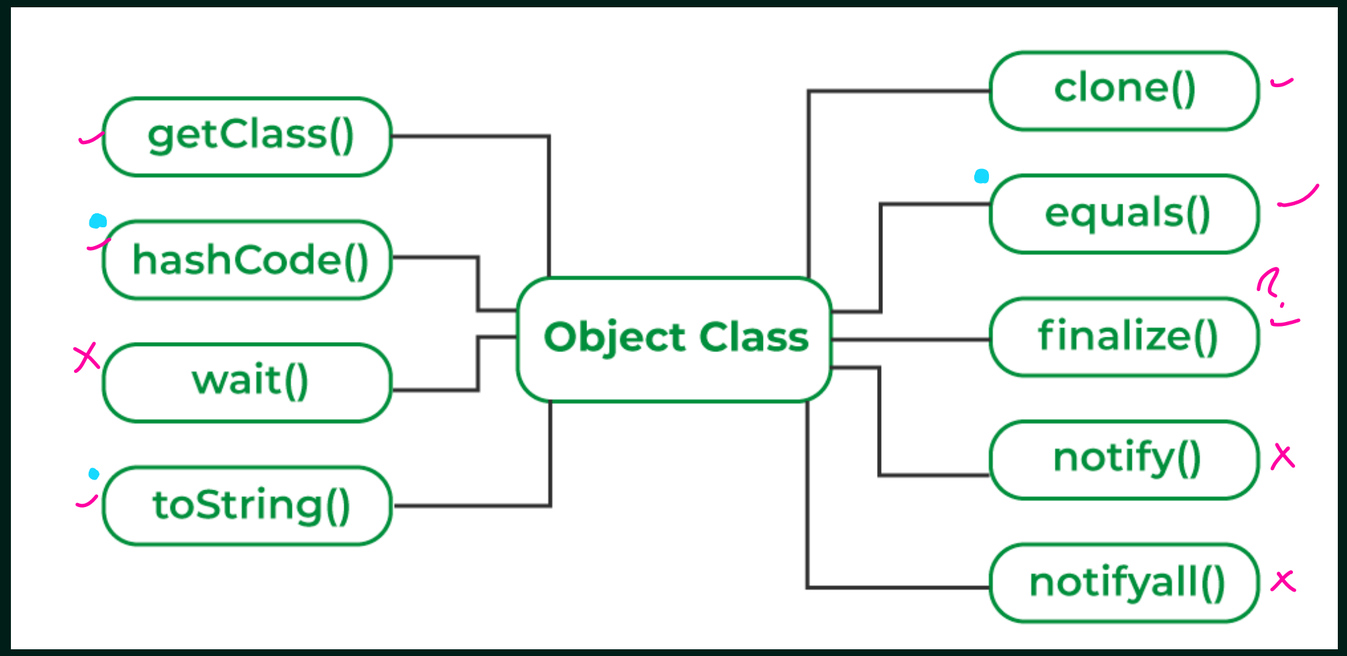
Java Stack Memory

Java Stack memory contains method-specific values that are short-lived and references to other objects in the heap that is getting referred from the method.

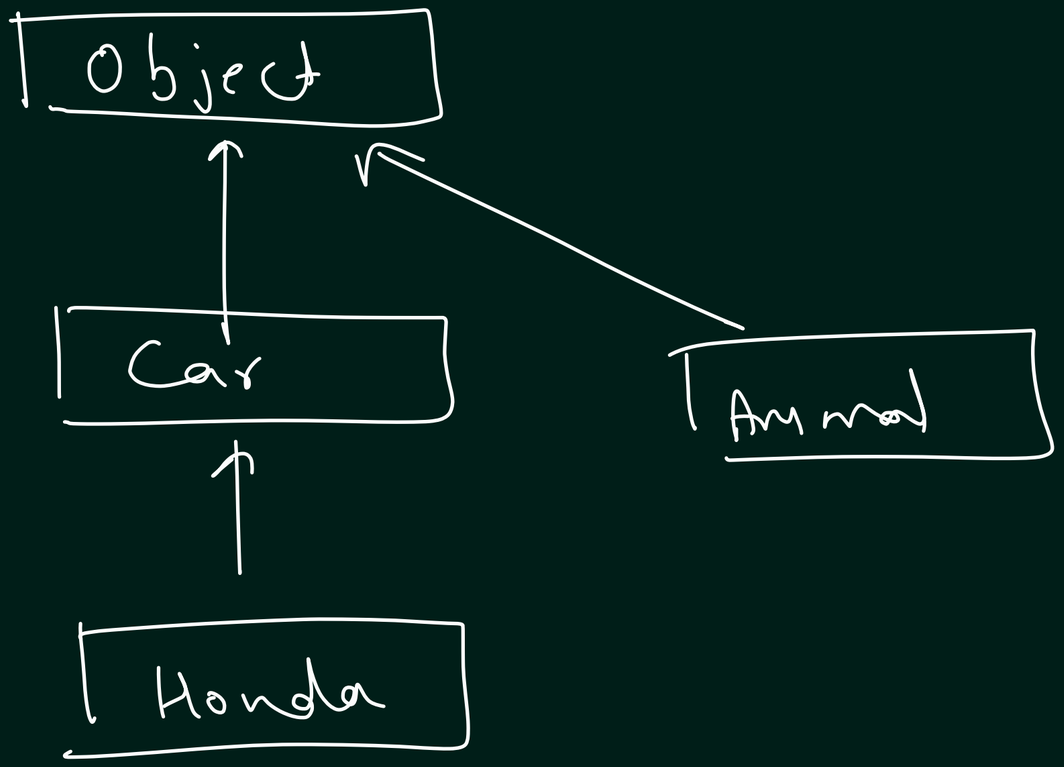
Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method. As soon as the method ends, the block becomes unused and becomes available for the next method. Stack memory size is very less compared to Heap memory.

Java Object Class

Object class is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class. If a class does not extend any other class then it is a direct child class of Object and if extends another class then it is indirectly derived. Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of the inheritance hierarchy in any Java Program.



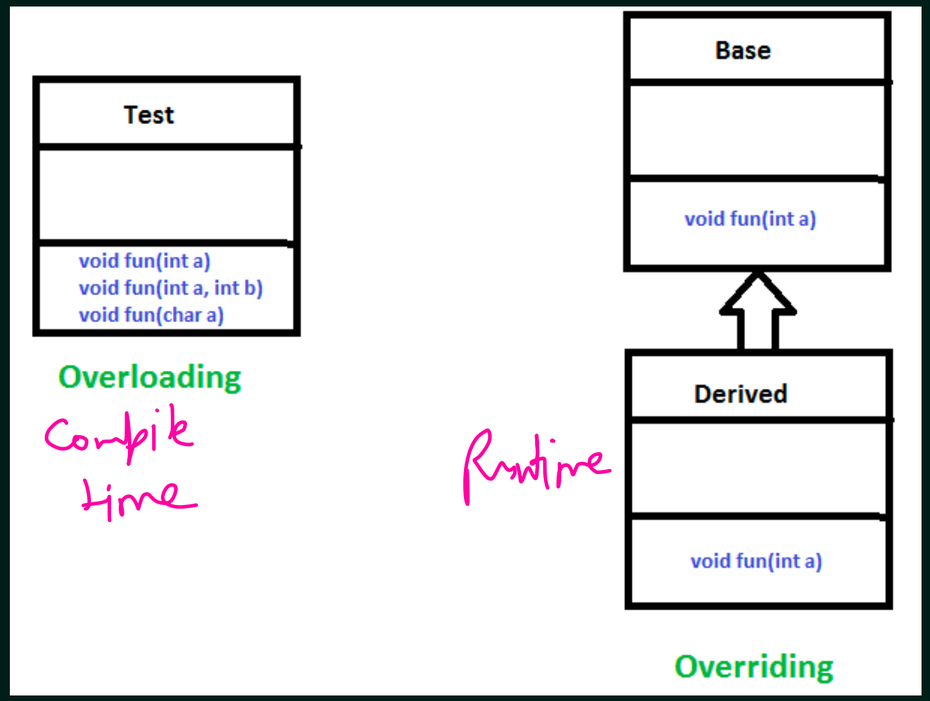
Java Object Class



Java Polymorphism

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms. There are two types of Polymorphisms.

- Compile-time Polymorphism
- Runtime Polymorphism



Java Polymorphism