

Java Course By CodeWithHarry

Java is an Object Oriented programming language developed by Sun Microsystems of USA in 1991

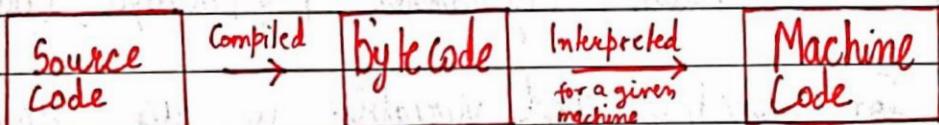
It was originally called Oak by James Gosling

↳ one of the inventors of Java!

JAVA = Purely Object Oriented

How JAVA Works?

Java is compiled into the bytecode and then it is interpreted to machine code



JAVA Installation

Go to Google & type "Install JDK" \Rightarrow Installs JAVA JDK

Go to Google & type "Install IntelliJ Idea" \Rightarrow Installs JAVA IDE

JDK \rightarrow JAVA Development Kit = Collection of tools used for developing and running Java programs

JRE \rightarrow JAVA Runtime Environment = Helps in executing programs developed in JAVA

Basic Structure of a Java Program

```
package com.Company; → Groups classes!  
public class Main { → Entry point into the application  
    public static void main (String [] args) {  
        System.out.println ("Hello World");  
    } }
```

Naming Conventions

→ For classes, we use Pascal Convention. First and Subsequent characters from a word are Capital letters (uppercase)

Example :

Main, MyScanner, MyEmployee, CodeWithHarry

→ For functions and variables, we use CamelCase Convention. Here first character is lowercase and the subsequent characters are uppercase like below:

main, myScanner, myMarks, CodeWithHarry

Chapter 1 - Variables and datatypes

Just like we have some rules that we follow to speak English (the grammar), we have some rules to follow while writing a Java program. The set of these rules is called syntax.

↳ Vocabulary & Grammar of Java

Variables

A variable is a container that stores a value. This value can be changed during the execution of the program.

Example :

int number = 8; *Value it stores!*
Data type *variable name*

Rules for declaring a variable name

We can choose a name while declaring a Java variable if the following rules are followed :

1. Must not begin with a digit → int 1arry; is invalid!
2. Name is case sensitive → harry and Harry are different!
3. Should not be a keyword (like void)
4. White Space not allowed. → int Code With Harry; is invalid
5. Can contain alphabets, \$ character, _ character and digits if the other conditions are met.

Data Types

Data types in Java fall under the following categories

1. Primitive Data Types (Intrinsic)
2. Non-Primitive Data Types (Derived)

Primitive Data Types

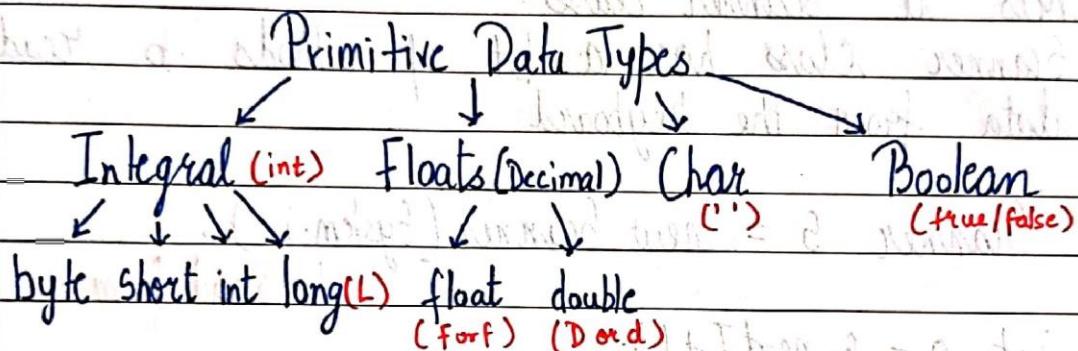
Java is statically typed. → Variables must be declared before use.
There are 8 primitive data types supported by Java:

- 1, byte →
 - Value ranges from -128 to 127
 - Takes 1 byte
 - Default value is 0
- 2, short →
 - Value ranges from $(2^{16})/2$ to $(2^{16})/2 - 1$
 - Takes 2 bytes
 - Default value is 0
- 3, int →
 - Value ranges from $(2^{32})/2$ to $(2^{32})/2 - 1$
 - Takes 4 bytes
 - Default value is 0
- 4, float →
 - Value ranges from (See Docs)
 - Takes 4 bytes
 - Default value is 0.0f
- 5, long →
 - Value ranges from $(2^{64})/2$ to $(2^{64})/2 - 1$
 - Takes 8 bytes
 - Default value is 0
- 6, double →
 - Value ranges from (See docs)
 - Takes 8 bytes
 - Default value is 0.0d
- 7, char →
 - Value ranges from 0 to 65535 ($2^{16} - 1$)
 - Takes 2 bytes → because it supports unicode
 - Default value is '\u0000'

- 8) boolean →
- Value can be true or false
 - Size depends on JVM
 - Default value is false

Quick Quiz : Write a Java program to add three numbers.

How to choose data types for our Variables



In order to choose the data type we first need to find the type of data we want to store. After that we need to analyze the Min & Max value we might use

Literals

A Constant value which can be assigned to the variable is called as a literal

- 101 → Integer literal
- 10.1f → Float literal
- 10.1 → double literal (default type for decimals)
- 'A' → character literal
- true → boolean literal
- "Harry" → String literal

Keywords

Words which are reserved and used by the Java Compiler. They cannot be used as an Identifier.



Go to docs.oracle.com for a comprehensive list!

Reading data from the Keyboard

In order to read data from the Keyboard, Java has a Scanner class.

Scanner class has a lot of methods to read the data from the keyboard

Scanner s = new Scanner(System.in);

↳ Read from the keyboard

int a = s.nextInt();

↳ Method to read from the keyboard

(Integer in this case)

Exercise 1.1

Write a Program to calculate percentage of a given student in CBSE board exam. His marks from 5 subjects must be taken as input from the keyboard. (Marks are out of 100).

Chapter 2 - Operators and Expressions

Operators are used to perform operations on variables and values.

$$7 + 11 = 18$$

↓ ↓ → ↓
 operand operator operand Result

Types of operators

- Arithmetic Operators → +, -, *, /, %, ++, --
- Assignment operators → =, +=
- Comparison operators → ==, >=, <=
- Logical operators → &, ||, !
- Bitwise Operators → &, | (operates bitwise)

Arithmetic operators cannot work with booleans
% operator can work on floats & doubles

Precedence of operators

The operators are applied and evaluated based on precedence. For example (+, -) has less precedence compared to (*, /). Hence * & / are evaluated first.

In case we like to change this order, we use parenthesis

Associativity

Associativity tells the direction of execution of operators
It can either be Left to Right or Right to left

* / → L to R

+ - → L to R

++, -= → R to L

Quick Quiz : How will you write the following expressions in Java ?

$$\frac{x-y}{2}, \frac{b^2 - 4ac}{2a}, \sqrt{v^2 - u^2}, a * b - d$$

Resulting data type after arithmetic operation
 following table summarizes the resulting data types after arithmetic operation on them

$R = b + 5 \rightarrow$	int	$b \rightarrow$ byte	$f \rightarrow$ float
$R = s + i \rightarrow$	int	$s \rightarrow$ short	$d \rightarrow$ double
$R = l + f \rightarrow$	float	$i \rightarrow$ integer	$c \rightarrow$ character
$R = i + f \rightarrow$	float	$l \rightarrow$ long	
$R = c + i \rightarrow$	int		
$R = c + s \rightarrow$	int		
$R = l + d \rightarrow$	double		
$R = f + d \rightarrow$	double		

Increment and Decrement operators

$a++$, $++a \rightarrow$ Increment operators \rightarrow Data type
 $a--$, $--a \rightarrow$ Decrement operators \rightarrow remains same

These will operate on all data types except booleans

Quick Quiz : Try increment and decrement operators on a Java variable

$a++ \rightarrow$ first use the value and then increment
 $++a \rightarrow$ first increment the value then use it



Quick Quiz : What will be the value of the following expression (x)

int y = 7;

int x = ++y * 8;

Value of x ?

char a = 'B';

a++; → a is now 'C'

Chapter 3 - Strings

A string is a sequence of characters.
A string is instantiated as follows:

```
String name;  
name = new String ("Harry");
```

String is a class but can be used like a data type:
[Strings are immutable and cannot be changed.]

```
String name = "Harry";  
      Reference       Object
```

Different ways to print in Java

We can use the following ways to print in Java:

1. `System.out.print()` → No newline at the end!
2. `System.out.println()` → Prints a new line at the end
3. `System.out.printf()`
4. `System.out.format()`

`System.out.printf("%c", ch)`

→
%d for int
%f for float
%c for char
%s for string

String Methods

String methods operate on Java Strings. They can be used to find length of the string, convert to lowercase, etc.

Some of the commonly used String methods are -

String name = "Harry";
 ^ 0 1 2 3 4

1. name.length() → Returns length of String name.
(5 in this case)
2. name.toLowerCase() → Returns a new String which has all the lowercase characters from the String name.
3. name.toUpperCase() → Returns a new String which has all the uppercase characters from the String name.
4. name.trim() → Returns a new String after removing all the leading and trailing spaces from the original String.
5. name.substring(int start) → Returns a substring from start to the end.
 ↳ substring(3)
 ↳ returns "ry"
 [Note that index starts from 0]
6. name.substring(int start, int end) → Returns a substring from Start index to the end index. Start index is included and end is excluded.
7. name.replace('r', 'p') → Returns a new string after replacing r with p. Happy is returned in this case.
 char char
 ↑ ↑

8. `name.startsWith("Ha")` → returns true if name starts with string "Ha". true in this case!
9. `name.endsWith("ry")` → returns true if name ends with string "ry". true in this case.
10. `name.charAt(2)` → returns character at a given index position. r in this case!
11. `name.indexOf(s)` → returns the index of the given string.
for ex: `name.indexOf("ar")` returns 1 which is the first occurrence of ar in string "Harry", -1 otherwise
12. `name.indexOf("s", 3)` → returns the index of the given string starting from the index 3 (int). -1 is returned in this case!
13. `name.lastIndexOf("r")` → returns the last index of the given string. 3 in this case!
14. `name.lastIndexOf("r", 2)` → returns the last index of the given string before index 2.
15. `name.equals("Harry")` → returns true if the given string is equal to "Harry" false otherwise [case sensitive]

= 16 name.equalsIgnoreCase("harry") → returns true if two strings are equal ignoring the case of characters.

Escape Sequence Characters

Sequence of characters after backslash ``

= Escape sequence characters

Escape sequence characters consist of more than one characters but represents one character when used within the strings.

Examples : \n, \t, \', \", etc.

newline Tab single quote → backslash

Chapter 4 - Conditionals in Java

Sometimes we want to watch comedy videos on YouTube if the day is Sunday.

Sometimes, we order junk food if it is our friend's birthday in the hostel.

You might want to buy an Umbrella if its raining and you have the money.

You order the meal if also or your favorite bhindi is listed on the menu.

All these are decisions which depends on a certain condition being met.

In Java, we can execute instructions on a condition being met.

Decision making Instructions in Java

- If - Else Statement
- Switch Statement

If - else Statement

The syntax of an If - Else statement in C looks like that of C++ and JavaScript. Java has a similar Syntax too. It looks like:

```
if (Condition - to - be - checked) {  
    Statements - if - Condition - true;
```

}

```
else {
```

```
    Statements - if - Condition - false;
```

}

Code Example :

```
int a = 29;  
if ( a > 18 ) {  
    System.out.println(" You can drive");  
}
```

Note that the else block is optional

Relational Operators in Java

Relational operators are used to evaluate conditions (true or false) inside the if statements.
Some examples of relational operators are:

$= =$, $> =$, $>$, $<$, $< =$, \neq
 \downarrow equals \downarrow greater than or equal to \downarrow Not equals

Note : '=' is used for assignment where as ' $= =$ ' is used for equality check.

The condition can be either true or false.

Logical Operators

$\&\&$, $||$ and $!$ are most commonly used logical operators in Java

These are read as :

$\&\&$ → AND

$||$ → OR

$!$ → NOT

⇒ Used to provide logic to our JAVA programs

AND operator

Evaluates to true if both the conditions are true

$$Y \& \& Y = Y$$

$Y \rightarrow \text{true}$

$$Y \& \& N = N$$

$N \rightarrow \text{false}$

$$N \& \& Y = N$$

$$N \& \& N = N$$

OR operator

Evaluates to true when at least one of the conditions is true.

$$Y || Y = Y$$

$Y \rightarrow \text{true}$

$$Y || N = Y$$

$N \rightarrow \text{false}$

$$N || Y = Y$$

$$N || N = N$$

Not Operator

Negates the given logic (true becomes false and false becomes true)

$$! Y = N$$

$Y \rightarrow \text{true}$

$$! N = Y$$

$N \rightarrow \text{false}$

else if clause

Instead of using multiple if statements, we can also use else if along with if thus forming an if-else-if-else ladder

Using such kind of logic reduces indents. Last else is executed only if all the conditions fail.

```
if (Condition) {  
    // Statements;  
}  
else if {  
    // Statements;  
}  
else {  
    // Statements;  
}
```

Switch Case Control Instruction

Switch - Case is used when we have to make a choice between number of alternatives for a given variable

```
Switch (Var) {  
    Case C1 :  
        // Code;  
        break;  
    Case C2 :  
        // Code  
        break;  
    Case C3 :  
        // Code  
        break;  
    default :  
        // Code
```

{

Var can be an integer, character or String in Java.

A Switch can occur within another but in practice this is rarely done

(("if true") instead of simple
("if true")) which is wrong

whole a variable two but at meadow a class
XOA but assume the final but is simple
and at variable does in XOA want to have
which is a string of two strings of meadow
result will work

at working not bind not more simple
but want to do with the same terminating

but	101	bold simple
XO	102 - 120	
XOS	1001 + 102	
XOE	1001 simple	

dot 101 worked not on so much last work
so many many many things like as follows

not too brief at meadow and as simple
and not I believe with simple but with the
no of but ... without of

Chapter 5 - Loop Control Instruction

Sometimes we want our programs to execute a few set of instructions over and over again. For example - print 1 to 1000, print multiplication table of 7, etc.

Loops make it easy for us to tell the computer that a given set of instructions need to be executed repeatedly.

Types of Loops

Primarily, there are three types of loops in Java:

1. While loop
2. do-while loop
3. for loop

We will look into these one by one.

While loops

While (boolean condition)
 {

 // Statement
 }

→ This keeps executing as long as the condition is true.

If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

Quick Quiz : Write a program to print natural numbers from 100 to 200.

do - while loop

This loop is similar to a while loop except the fact that it is guaranteed to execute at least once.

do {

// Code

} while (condition); → Note this Semicolon

while → checks the condition & executes the code

do - while → Executes the code & then checks the condition

Quick Quiz : Write a program to print first n natural numbers using do - while loop.

for Loop

The syntax of a for loop looks like this :

for (initialize; check bool expression; update) {

// Code ;

}

A for loop is usually used to execute a piece of code for specific number of times.

Quick Quiz : Write a program to print first n odd numbers using a for loop.

Decrementing for loop

```
for( i= 7 ; i != 0 ; i-- )  
    { System.out.println( i ); }
```

This for loop keeps running until i becomes 0.

Quick Quiz : Write a program to print first n natural numbers in reverse order

break statement

The break statement is used to exit the loop irrespective of whether the condition is true or false.

Whenever a "break" is encountered inside the loop, the control is sent outside the loop.

Continue statement

The continue statement is used to immediately move to the next iteration of the loop.

The control is taken to the next iteration thus skipping everything below "continue" inside the loop for that iteration.

In A Nut Shell...

1. break statement completely exits the loop
2. continue statement skips the particular iteration of the loop.

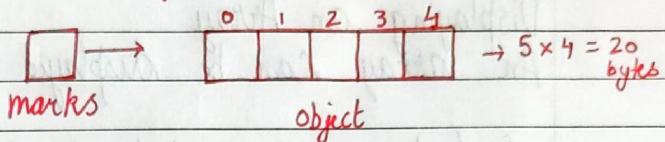
Chapter 6 - Arrays

Array is a collection of similar types of data

Use Case : Storing marks of 5 Students

`int [] marks = new int [5] \Rightarrow [dataType ArrName];`

\downarrow reference \downarrow object



Accessing Array Elements

Array elements can be accessed as follows

`marks[0] = 100`

`marks[1] = 70`

⋮
⋮
⋮

`marks[4] = 98`

\Rightarrow Note that index starts from 0

So in a nut shell, this is how array works :

1. `int [] marks;` \rightarrow Declaration !

`marks = new int[5];` \rightarrow Memory Allocation !

2. `int [] marks = new int[5];` \rightarrow Declaration + Memory Allocation !

3. `int [] marks = { 100, 70, 80, 71, 98 };` \rightarrow Declare + Initialize !

Array indices starts from 0 and goes till $(n-1)$
where n is the size of the array.

Array length

Arrays have a length property which gives the length of the array

`marks.length` → gives 5 if marks is a reference to array with 5 elements

Displaying an Array

An array can be displayed using a for loop:

`for (int i=0 ; i < marks.length ; i++)`

{

`sout(marks[i]);`

}

⇒ Array Traversal

Quick Quiz: Write a Java program to print the elements of an array in reverse order.

for-each loop in Java

Array elements can also be traversed as follows:

`for (int element : Arr) {`

`sout(element);`

}

⇒ Prints all the elements

Multidimensional Arrays

Multidimensional Arrays are Array of Arrays

Each element of a M-D array is an array itself
marks in the previous example was a 1-D array

Multidimensional 2-D Array

A 2-D array can be created as follows:

`int [][] flats = new int [2][3]`

↳ A 2-D array of 2 rows + 3 columns

We can add elements to this array as follows

`flats [0][0] = 100`

`flats [0][1] = 101`

`flats [0][2] = 102`

⋮

& so on!

This 2-D array can be visualised as follows:

	[0]	[1]	[2]
[0]	(0, 1)	(0, 2)	(0, 3)
[1]	(1, 0)	(1, 1)	(1, 2)

Similarly a 3-D array can be created as follows:

`String [[[]]] arr = new String [2][3][4]`

Chapter 7 - Methods in Java

Sometimes our program grows in size and we want to separate the logic of main method to other methods

for instance - If we are calculating average of a number pair 5 times, we can use methods to avoid repeating the logic.

↳ DRY = Don't Repeat Yourself

Syntax of a Method

A method is a function written inside a class.
Since Java is an Object Oriented language, we need to write the method inside some class

```
dataType name () {  
    // Method body  
}
```

Following method returns sum of two numbers

→ Return type
`int mySum(int a, int b) {
 int c = a + b;
 return c;
}` → Return value

Calling a Method

A method can be called by creating an object of the class in which the method exists followed by the method call:

`Calc obj = new Calc();` → Object creation
`obj.mySum(a, b);` → Method call upon an object

The values from the method call (a and b) are copied to the a and b of the function mySum. Thus even if we modify the values a and b inside the method, the values in the main method will not change.

Void return type

When we don't want our method to return anything, we use Void as the return type.

Static keyword

static keyword is used to associate a method of a given class with the class rather than the object. Static method in a class is shared by all the objects.

Process of method invocation in Java

Consider the method sum:

```
int sum (int a, int b)
{
    return a+b;
}
```

The method is called like this:

```
Calc obj = new Calc();
c = obj.sum(2, 3)
```

The values 2 and 3 are copied to a and b and then $a+b = 2+3 = 5$ is returned in c which is an integer.

Note: In case of Arrays, the reference is passed. Same is the case for Object passing to methods.

Method Overloading

Two or more methods can have same name but different parameters. Such methods are called Overloaded methods.

Void foo()

Void foo(int a)

int foo(int a, int b)

⇒ Overloaded function foo

Method overloading cannot be performed by changing the return type of methods

Variable Arguments (Varargs)

A function with Vararg can be created in Java using the following Syntax:

public static Void foo (int ... arr)
{

// arr is available here as int [] arr
}

foo can be called with zero or more arguments like this :

foo(7) foo(7,8,9) foo(1,2,7,8,9)

We can also create a function bar like this

public static void bar (int a, int arr)
{

// code
}

→ Atleast one integer is required now

bar can be called as bar(1), bar(1,2), bar(1,7,9,11) etc.

Recursion

A function in Java can call itself. Such calling of function by itself is called recursion.

Example: Factorial of a number

$$\text{factorial}(n) = n * \text{factorial}(n-1) \quad \forall n \geq 1$$

Quick Quiz: Write a program to calculate (recursion must be used) factorial of a number in Java?

Chapter - 8 : Introduction to OOPs

Object Oriented programming tries to map code instructions with real world making the code short and easier to understand.

What is Object Oriented Programming?
Solving a problem by creating objects is one of the most popular approaches in programming.
This is called Object Oriented Programming.

What is DRY?

DRY stands for - Do not repeat yourself
↳ Focuses on code reusability

Class

A class is a blueprint for creating objects.

JEE
Application
Form

⇒ Filled by an Student ⇒

Application for
that Student

Class

⇒ Object Instantiation ⇒ Object

Contains info to
create a valid
object.

Object

An Object is an instantiation of a class. When a class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

How to model a problem in OOPs
We identify the following:

Noun → Class → Employee
Adjective → Attributes → name, age, Salary
Verb → Methods → getSalary(), increment()

OOPs Terminology

1. Abstraction → Hiding internal details [show only essential info.]



⇒ Use this phone without bothering about how it was made

2. Encapsulation → The act of putting various components together (in a capsule).



⇒ Laptop is a single entity with WiFi + Speaker + Storage in a single box!

In Java, encapsulation simply means that the sensitive data can be hidden from the users.

3. Inheritance → The act of deriving new things from existing things.

Rickshaw ⇒ E-Rickshaw

Phone ⇒ Smart Phone

Implements DRY!

4. Polymorphism → One entity many forms

Smartphone → Phone

Smartphone → Calculator

Writing a Custom Class

We can write a custom class as follows:

```
public class Employee {  
    int id;           → Attribute 1  
    String name;     → Attribute 2  
}
```

Any real world Object = Properties + Behaviour
Object in OOPs = Attributes + Methods

A class with Methods

We can add methods to our class Employee as follows:

```
public class Employee {  
    public int id;  
    public String name;  
  
    public int getSalary() {  
        // Code  
    }  
  
    public void getDetails() {  
        // Code  
    }  
};
```

Chapter 9 - Access Modifiers & Constructors

Access Modifiers

Specifies where a property / method is accessible

There are four types of access modifiers in Java:

- 1> Private
- 2> Default
- 3> Protected
- 4> Public

Getters and Setters

Getter → Returns the Value [accessors]

Setter → Sets/Updates the Value [mutators]

Example :

```
public class Employee {  
    private int id;  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName() {  
        this.name = "Your-name";  
    }  
  
    public void setName(String n) {  
        this.name = n;  
    }  
}
```

Quick Quiz : Use these getters and setters from the main method.

Constructors in Java

A member function used to initialize an object while creating it.

```
Employee harry = new Employee();
harry.setName("Harry Bhai");
```

In order to write our own constructor, we define a method with name same as class name.

```
public Employee() {
    name = "Your Name";
}
```

Constructor Overloading in Java

Constructors can be overloaded just like other methods in Java. We can overload the Employee constructor like below :

```
public Employee(string n) {
    name = n;
}
```

Note : ① Constructors can take parameters without being overloaded.

② There can be more than two overloaded constructors.

Quick Quiz : Overload the Employee constructor to initialize the Salary to Rs 10,000.

Chapter 10 - Inheritance

Inheritance is used to borrow properties & methods from an existing class

Phone → SmartPhone

SuperClass → SubClass Subclass extends SuperClass

Declaring Inheritance in Java

Inheritance in Java is declared using extends keyword

Superclass
↓

Subclass

⇒ Subclass extends the Superclass

More Examples

Vehicle

↓

Car

Animal

↓

Dog

Animal

↓

Cat

Vehicle

↓

Truck

When a class inherits from a superclass, it inherits parts of superclass methods and fields.

Java doesn't support multiple inheritance ie two classes cannot be super classes for a subclass.

Code Example

Inheritance in Java is declared using extends keyword

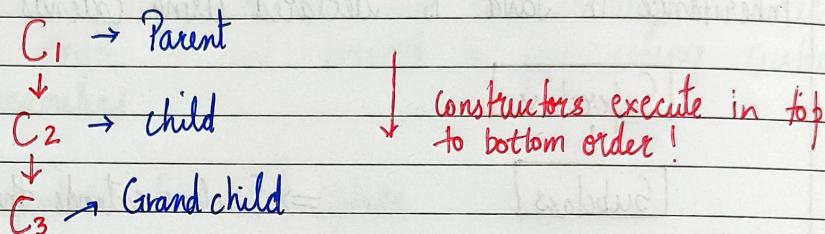
public class Dog extends Animal { → Inheriting Dog from Animal Class!!
 // Code
}

Quick Quiz : Create a class Animal and Derive another class Dog from it.

Constructors in Inheritance

When a Derived class is extended from the Base class, the Constructor of the Base class is executed first followed by the constructor of the derived class.

For the following Inheritance hierarchy, the constructors are executed in the order ① → ② → ③



Constructors during Constructor Overloading

When there are multiple constructors in the parent class, the constructor without any parameters is called from the child class.

If we want to call the constructor with parameters from the parent class, we can use Super keyword.

`Super (a , b) ;` → Calls the constructor from the parent class which takes 2 variables

this keyword

this is a way for us to reference an object of the class which is being created/referenced.

this::area = 2 → this is a reference to current object

Super Keyword

A reference variable used to refer immediate parent class object

- Can be used to refer immediate parent class instance variable
- Can be used to invoke parent class methods.
- Can be used to invoke parent class constructors.

Method Overriding

If the child class implements the same method present in the parent class again, it is known as method overriding

Chapter 11 - Abstract Classes & Interfaces

What does Abstract (class) mean?

Abstract in English means → existing in thought or as an idea without concrete existence

Abstract method

A method that is declared without an implementation

abstract void moveTo (double x, double y)

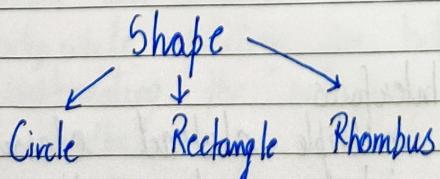
Abstract Class

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class PhoneModel {  
    abstract void switchoff();  
    // more code  
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the methods in parent class. If it doesn't, it must be declared abstract

An Example

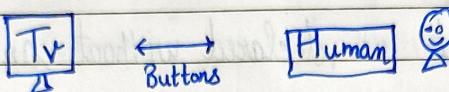


Note - It is possible to create reference of an abstract class
It is not possible to create an object of an abstract class

We can also assign reference of an abstract class to the object of a concrete subclass.

Interfaces in Java

Interface in English is a point where two systems meet and interact



In Java interface is a group of related methods with empty bodies

An Example

```

interface Bicycle {
    void applyBrake(int decrement);
    void speedUp(int increment);
}
    
```

Class AvonCycle implements Bicycle {

int speed = 7;

```

void applyBrake(int decrement) {
    speed = Speed - decrement;
}
    
```

```

void speedUp(int increment) {
}
    
```

speed = Speed + increment;

Abstract class vs Interfaces

We can't extend multiple abstract classes but we can implement multiple interfaces at a time.
Interfaces are meant for dynamic method dispatch

and run time polymorphism

Is multiple inheritance allowed in Java?

Multiple inheritance face problems when there exist methods with same signature in both the super classes.

Due to such problems, Java does not support multiple inheritance directly but the similar concept can be achieved using Interfaces.

A class can implement multiple Interfaces and extend a class at the same time.

- Note :
- ① Interfaces in Java is a bit like the Class but with a significant difference.
 - ② An Interface can only have method signatures, ^{constant} fields and default methods.
 - ③ The class implementing an Interface needs to only declare the methods (not fields).
 - ④ You can create a reference of Interfaces but not the Object.
 - ⑤ Interface methods are public by default.

Default methods

An interface can have static and default methods.

Default methods enable us to add new functionality to existing Interfaces.

This feature was introduced in Java 8 to ensure backward compatibility while updating an Interface.

Classes implementing the interface need not implement the default methods.

Interfaces can also include private methods for default methods to use.

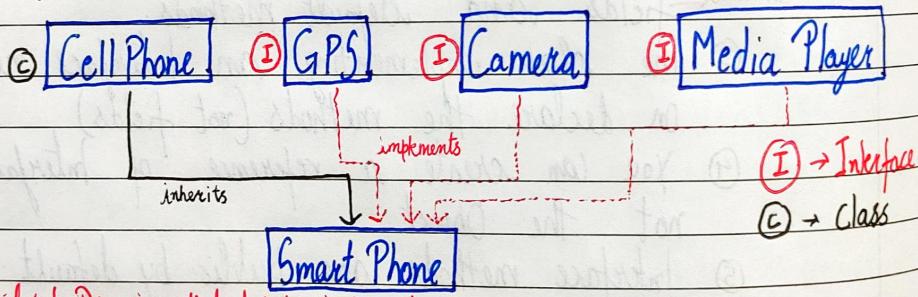
Inheritance in Interfaces
Interfaces can extend another interfaces :

public interface Interface 1 {
 Void meth1();
}

public interface Interface2 extends Interface 1 {
 Void meth2();
}

Remember that interface cannot implement another interface, only classes can do that!

Polymorphism using Interfaces



Similar to Dynamic method dispatch in Inheritance

GPS g = new SmartPhone(); → Can only use GPS methods
SmartPhone s = new SmartPhone(); → Can only use SmartPhone methods

Implementing an Interface forces method implementation.

Chapter 12 - Packages

Interpreter vs Compiler

Interpreter translates one statement at a time into machine code.

Compiler scans the entire program and translates whole of it into machine code.

Interpreter



- * One statement at a time
- * Interpreter is needed everytime
- * Partial execution if error
- * Easy for programmers

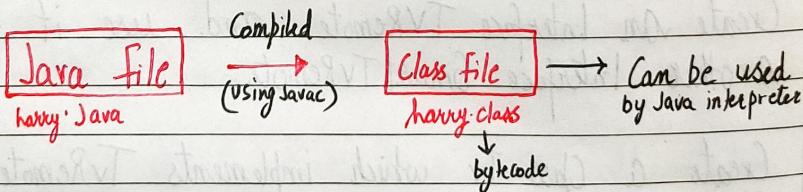
Compiler



- * Entire program at a time
- * Once compiled it is not needed
- * No execution if an error occurs
- * Usually not as easy as interpreted ones

Is Java Compiled or Interpreted?

Java is a hybrid language → both compiled as well as interpreted



- A JVM can be used to interpret this bytecode
- This bytecode can be taken to any platform (Win/Mac/Linux) for execution
- Hence Java is platform independent (write once run everywhere)

Executing a Java Program

Java Harry.java → Compiled
java Harry.class → Interpreted

So far the execution of our program was being managed by IntelliJ Idea.

We can download a source code editor like VS Code to compile & execute our Java programs.

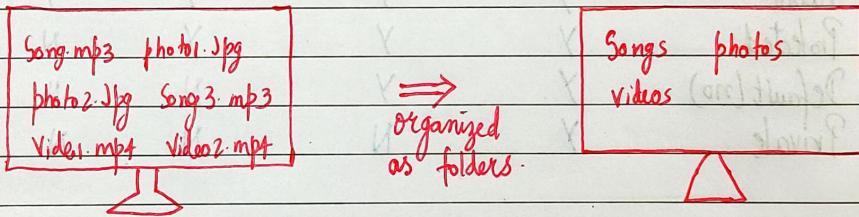
Packages in Java

A package is used to group related classes.

Packages help in avoiding name conflicts.

There are two types of packages:

- * Built in packages → Java API
- * User defined packages → Custom packages



1. class this.java my.mp3
Song.java Harry.java
⇒ organized as packages

Using a Java package

import java.lang.* → import everything from java.lang
import java.lang.String → import String from java.lang
s = new java.lang.String("Harry") → Use without importing

Creating a package

`javac Harry.java` → creates Harry class

`javac -d . Harry.java` → creates a package folder

↳ We can keep adding classes
to a package like this

We can also create inner packages by adding "package inner"
as package name

These packages once created can be used

by other classes.

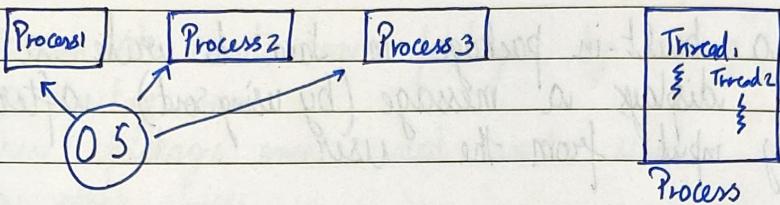
Access Modifiers in Java

Access modifiers determine whether other classes can use
a particular field or invoke a particular method
Can be public, private, protected or default (no modifier)

Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Default (no)	Y	Y	N	N
Private	Y	N	N	N

Chapter 13 - Multithreading

Multiprocessing and multithreading both are used to achieve multitasking



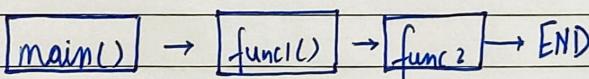
In a nut shell ..

- Threads use shared memory area
- Threads \Rightarrow Faster Context switching
- A Thread is light-weight whereas a process is heavyweight

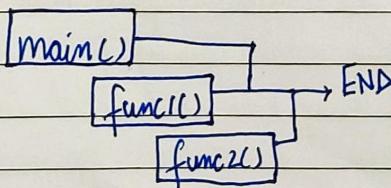
For Example → A word processor can have one thread running in foreground as an editor and another in the background auto saving the document !

Flow of control in Java

1. Without threading :



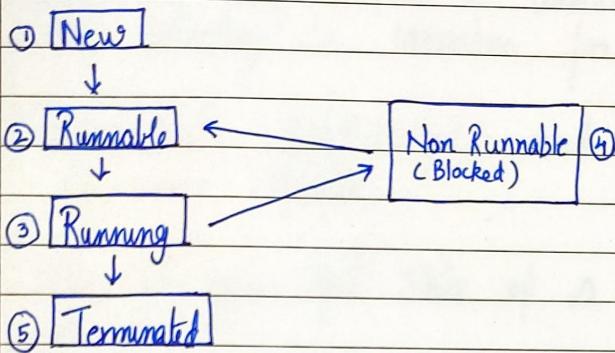
2. With threading :



Creating a Thread

- There are two ways to create a thread in Java.
1. By extending Thread class
 2. By implementing Runnable interface

Life cycle of a Thread



- ① New → Instance of thread created which is not yet started by invoking start()
- ② Runnable → After invocation of start() & before it is selected to be run by the scheduler.
- ③ Running → After thread scheduler has selected it.
- ④ Non Runnable → Thread alive, not eligible to run.
- ⑤ Terminated → run() method has exited

The Thread class

Below are the commonly used constructors of Thread class:

- ① Thread()
- ② Thread(String name)
- ③ Thread(Runnable r)
- ④ Thread(Runnable r, String name)

Methods of Thread class

Thread class offers a lot of methods such as run(), start(), join(), getPriority(), setPriority() etc.
More can be found on visiting Java docs

Chapter 14 - Errors & Exceptions

No matter how smart we are, errors are our constant companions. With practice, we keep getting better at finding & correcting them.

There are three types of errors in Java.

- 1> Syntax errors
- 2> Logical errors
- 3> Runtime errors → Also called Exceptions!

Syntax Errors

When compiler finds something wrong with our program, it throws a syntax error.

`int a = 9` → No semicolon, syntax error!
`a = a + 3;`

`d = 4;` → Variable not declared, Syntax error!

Logical errors

A logical error or a bug occurs when a program compiles and runs but does the wrong thing.

- message delivered wrongly
- Wrong time of chats being displayed
- incorrect redirects!

Runtime Errors

Java may sometimes encounter an error while the program is running. These are also called exceptions!

These are encountered due to circumstances like bad input and/or resource constraints.

Ex:- User supplies '5+8' to a program which adds 2 numbers.

Syntax errors and logical errors are encountered by the programmer whereas Runtime errors are encountered by the users.

Exceptions in Java

An Exception is an event that occurs when a program is executed disrupting the normal flow of instructions.

There are mainly two types of exceptions in Java:

1. Checked Exception → Compile time exceptions (Handled by compiler)
2. Unchecked Exception → Runtime exceptions

Commonly Occurring Exceptions

Following are few commonly occurring exceptions in Java:

1. NullPointerException
2. ArithmeticException
3. ArrayIndexOutOfBoundsException
4. IllegalArgumentException
5. NumberFormatException

try-catch block in Java

In Java, exceptions are managed using try-catch blocks.

Syntax:

```
try {  
    // Code to try  
}  
catch (Exception e) {  
    // Code if exception  
}
```

Handling Specific Exceptions

In Java, we can handle specific exceptions by typing multiple catch blocks.

```
try {  
    // Code  
}
```

```
Catch (IOException e) {  
    // Code  
}
```

→ Handles all Exceptions of type IOException

```
Catch (ArithmaticException e) {  
    // Code  
}
```

→ Handles all Exceptions of type ArithmaticException

```
Catch (Exception e) {  
    // Code  
}
```

→ Handles all other Exceptions

Nested try-catch

We can nest multiple try-catch blocks as follows:

```
try {  
    try {  
        // Code  
    }
```

```
    Catch (Ex. c) {  
        // Code  
    }
```

```
}
```

```
Catch (Ex. c) {  
    // Code  
}
```

⇒ Nested try-catch blocks

Similarly, we can further nest try catch blocks inside the nested try catch blocks.

Quick Quiz : Write a Java program that allows you to keep accessing an array until a valid index is given by the user.

Exception class in Java

We can write our custom Exceptions using Exception class in Java.

```
public class MyException extends Exception {  
    // No overridden methods  
}
```

The Exception class has following important methods :

- (1) String toString() → executed when sout(e) is run
- (2) void printStackTrace() → prints stack trace
- (3) String getMessage() → prints the Exception message

The Throw keyword

The throw keyword is used to throw an exception explicitly by the programmer

```
if (b == 0) {  
    throw new ArithmeticException ("Div by 0");  
}  
else {  
    return a/b;  
}
```

In a similar manner, we can throw user defined exceptions:

```
throw new MyException ("Exception thrown");
```

The throws exception

The Java throws keyword is used to declare an exception. This gives an information to the programmer that there might be an exception so it's better to be prepared with a try catch block!

```
public void calculate(int a, int b) throws IOException {  
    // Code
```

{

Java finally block

finally block contains the code which is always executed whether the exception is handled or not. It is used to execute code containing instructions to release the system resources, close a connection etc.

Advanced Java - 1

Collections Framework

A collection represents a group of objects.
Java Collections provide Classes and Interfaces
for us to be able to write code quickly and
efficiently.

Why do we need Collections

We need Collections for efficient storage and
better manipulation of data in Java.

For ex: We use arrays to store integers but
what if we want to

- Resize this array?
- Insert an element in between?
- Delete an element in Array?
- Apply certain operations to change
this array?

How are collections available

Collections in Java are available as Classes and
Interfaces. Following are few commonly used
Collections in Java:

- * ArrayList → For variable size Collection
- * Set → For distinct collection
- * Stack → A LIFO data structure
- * HashMap → For storing key-value pairs

Collection class is available in `java.util` package
Collection class also provides static methods for
sorting, searching etc.

Date & Time in Java

Java.time → package for Date & time in Java
↳ from Java 8 onwards

Before java 8, java.util package used to hold the date and time classes. Now these classes are deprecated

How java stores a Date?

Date in Java is stored in the form of a long number. This long number holds the number of milliseconds passed since 1 Jan 1970

Java assumes that 1900 is the start year which means it calculates years passed since 1900 whenever we ask it for years passed.

System.currentTimeMillis() returns no of seconds passed. Once no of ms are calculated, we can calculate minutes, seconds & years passed

Quick Quiz : Is it safe to store the no of ms in a variable of type long?

The Date class in Java

```
Date d = new Date();
```

```
System.out.println(d);
```

We can also use constructors provided by the Date class

Java Date class has few methods which can be used.
for ex: getDate(), getDay() etc.

All these methods are deprecated

Calendar Class in Java

Calendar is an abstract class that provides calendar related methods in Java

Calendar.getInstance → returns a Calendar instance based on current time

Calendar a = Calendar.getInstance();
a.getTime() → prints time

Calendar class methods

1. get method is used to get year, date, min, second

a.get(Calendar.SECOND)
a.get(Calendar.MINUTE)
a.get(Calendar.DATE)
a.get(Calendar.YEAR)

2. getTime method returns a Date object

3. Other methods can be looked up from the Java docs!

Gregorian Calendar Class

This class is used to create an instance of gregorian calendar

We can change the year month & date using Set method!

Time Zone

TimeZone class is used to create Time Zones in Java. Some of the important methods of TimeZone class are :

- getAvailableIDs() → get all the available IDs supported
- getDefault() → get the default timezone
- getID() → get the ID of a TimeZone

Java time package

- Available from Java 8 onwards
- Capable of storing even nano seconds

Following are some of the most commonly used classes from java.time package.

LocalDate → Represents a Date

LocalTime → Represents a Time

LocalDateTime → Represents a Date + Time

Date Time Formatter → Formatter for displaying & parsing date-time objects

Advanced Java - 2

Creating our own Java Documentation

Java documentation is great!

If helps us get info about which method / class / Entity to use when.

We can create our own package's Documentation in Java.

Javadoc tool

javadoc command allows us to create documentation in HTML format for our own package.

Java provides tags for class or package to assist with the java doc generation

Tags for class or a package

- 1> @ author : Adds the author name
- 2> @ Version : Adds the version.
- 3> @ Since : To add when was this version written
- 4> @ See : Adds a see also heading with a link

Tags for methods

Javadoc provides following tags for methods :

- 1> @ param → for describing parameters of a method
- 2> @ return → for describing about the return value
- 3> @ throws → for describing exception thrown
- 4> @ deprecated → for describing deprecation status

Description can be added at the start of javadoc comment

Annotations in Java

Used to provide extra information about a program
Annotations provides metadata to class/methods

Following are some common annotations built into Java

1. `@Override` → Used to mark overridden elements in the child classes
2. `@SuppressWarnings` → Used to suppress the generated warnings by the compiler
3. `@Deprecated` → Used to mark deprecated methods
4. `@FunctionalInterface` → Used to ensure an interface is a functional interface

Lambda Expressions

Added in Java 8

Lambda expressions let us express instances of single method classes more compactly.

Anonymous classes are used to implement a base class without giving it a name.

For classes with a single method, even anonymous classes get slightly excessive & cumbersome

Java Generics

Introduced from JDK 5.0 onwards

Very similar to C++ Templates (but not the same)

If we write

`ArrayList a = new ArrayList();`

a.add(75)

int anum = a.get(0) → We can't do this

int anum = (int) a.get(0) → We will have to do this

Hence generics aim to reduce bugs & enhance type safety.

Note: Type parameter in java generics cannot be a primitive data type.

File Handling in Java

Reading from & writing to files is an important aspect of any programming language

We can use the File class in Java to create a file object

- Create NewFile() method → creates a file object
- For reading files we can use the same Scanner class and supply it a file object
- To delete a file in Java we can use file object's delete() method.

Ultimate Java Quick Reference - CodeWithHarry

1. // [comment]	10. Private	64-bit number with decimals.
Single line comment.	Can only be changed by a method.	
2. /* [comment] */	11. int	32-bit number with decimals.
Multi line comment.	Can store numbers from 2^{-31} to 2^{31} .	
3. public	12. fields are attributes	20. protected
This can be imported publically.		Can only be accessed by other code in the package.
4. import [object].*	13. boolean	21. Scanner
Imports everything in object.	Can have true or false as the value.	This lets you get user input.
5. static	14. {}	22. new [object constructor]
Going to be shared by every [object].	These are used to start and end a function, class, etc.	This will let you create a new object.
6. final	15. byte	23. System.in
Cannot be changed; common to be defined with all uppercase.	These can store from -127 - 128.	This lets you get data from the keyboard.
7. double	16. long	24. public [class]()
Integer with numbers that can have decimals.	Can store numbers from 2^{127} to 2^{-127} .	This will be the constructor, you use it to create new objects.
8. ;	17. char	25. super()
Put after every command.	Just lets you put in one character.	This will create the superclass (the class it's inheriting).
9. String	18. double	
Just a string of characters.		

Ultimate Java Quick Reference - CodeWithHarry

26. extends [class]	35. public static void main(String[] args)	44. <
Makes the object a subclass of [object], [object] must be a superclass.	This is your main function and your project will start in here.	This means less than.
27. ++	36. System.out.print([text])	45. >
Will increment the amount.	This prints stuff but there is no line break. (/n)	This means greater than.
28. --	37. \n	46. >=
Will decrement the amount.	Called a line break; will print a new line.	This means greater than or equal to.
29. += [amount]	38. \t	47. [inputVarHere].hasNextLine()
Increment by [amount]	This will print a tab.	This will return if there is a next line in the input.
30. -= [amount]	39. if ([condition])	48. this
Decrement by [amount]	This will make it so if [condition] is true then it'll keep going.	Refer to the class that you are in.
31. *= [amount]	40. &&	49. [caller].next[datatype]()
Multiply by [amount]	This means and.	This will get the [datatype] that you somehow inputted.
32. /= [amount]	41. !	50. Create getters and setters
Divide by [amount]	This means not.	This will create the get methods and set methods for every checked variable.
33. System.out.println([text])	42.	51. [caller].hasNext[datatype]()
Will print something to the output console.	This means or.	
34. +	43. ==	
Can be used for concatenation. (ex. "6" + [var_here])	This means equal to.	

Ultimate Java Quick Reference - CodeWithHarry

This will return if it has the correct datatype within the input.

52. overloading

If you have different parameters you can call them whatever way you want.

53. parameters

These are the inputs of your function.

54. ([datatype])[variable]

This will convert [variable] into [datatype]. Also known as casting.

55. Math.random()

Generate an extremely precise string of numbers between 0 and 1.

56. Primitives

Just the basic data types which are not objects.

57. [x].toString()

Will convert [x] into a string.

58.
[number].parse[number type][string])

This will parse [number] into the [numbertype] with [string].

59. ^

Return true if there is one true and one false.

60. !=

Not equal too. (NEQ)

61. ([condition]) ? [amount] : [var]

This will be like a shortcut way to an if statement.

62. switch([variable])

This will do stuff with specific cases. (e.g. switch(hi){ case 2: (do stuff)})

63. case [value]:

This will do stuff if the case is the case.

64. break

Put that when you want to leave the loop/switch; should be at end of case.

65. default [value]:

This will do stuff if none of the cases in the switch statement was made.

66. for ([number]; [condition]; [operation])

This will start at [number] and then do [operation] until [condition] is met.

67. continue

This will just go back to the enclosing loop before reaching other code.

68. while ([condition])

This will basically do something while [condition] is true.

69. void

This means no return type.

70. return

This will return something when you call it to where it was called from .

71. do { } while ([condition])

Guarantees it will execute once even if [condition] isn't met.

72. printf("%[type] stuff here bah bla", [variable here])

This will let you use [variable here] with %s being where.

Ultimate Java Quick Reference - CodeWithHarry

73. `System.out.printf([text])`

Another way to print? //
didn't quite get but ok then

This will get how long
something is, text, amount
of indexes in array, etc.

74. `[type] [returntype]
[name]([parameters]) {`

This is a way to create a
method.

80. `Arrays.copyOf([array],
indexes);`

This will copy the array and
how many indexes into
another array.

75. `[type][[indexes]]`

This will create an array
with [indexes] amount of
indexes; default infinite.

81. `Arrays.toString([array])`

Convert the whole array
into one huge string.

76. `int[] something = new
int[20];`

This will just make an array
of ints with 20 ints in it.

82.
`Arrays.binarySearch([array],
[object])`

This will search for [object]
in [array].

77. `for ([object]
[nameOfObject] :
[arrayOfObject]) {`

This will iterate through all
of the arrayOfObject with
object in use incrementing
by 1 until done.

78. `[object][[1]][[2]][[3]]
[name] = {[value] [value]
[value] \n [value] [value]
[value]}`

[1] is how many down in
array, [2] how many accross
in array, [3] how many
groups

79. `.length`