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 Goslin
↳ 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

Source Code → (Compiled) → Byte Code → (Interpreted for a given machine) → Machine Code

JAVA Installation
Go to Google & type "Install JDK" ⇒ Installs JAVA JDK
Go to Google & type "Install IntelliJ Idea" ⇒ Installs JAVA IDE

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

JRE → JAVA Runtime Environment = Helps in executing programs developed in JAVA

Basic Structure of a Java Program

```
package com.company;
```
→ Groups classes!

Entrypoint into the application

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

Naming Conventions

→ For classes, we use PascalConvention. First and subsequent characters from a word are capital letters (uppercase)

Example:

Main, MyScanner, MyEmployee, CodeWithHarry

→ For functions and variables, we use camelCaseConvention. 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

Primitive Data Types
- Integral (int)
  - byte
  - short
  - int
  - long (L)
- Floats (Decimal)
  - float (f or F)
  - double (D or d)
- Char ('')
- Boolean (true/false)

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

URBAN EDGE

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).

URBAN EDGE

Chapter 1 - Practice Set

1. Write a program to sum three numbers in Java

2. Write a program to calculate CGPA using marks of three subjects (out of 100).

3. Write a Java program which asks the user to enter his/her name and greets them with "Hello <name>, have a good day" text.

4. Write a Java program to convert Kilometers to miles

5. Write a Java program to detect whether a number entered by the user is integer or not.

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?

$\frac{x-y}{2}$, $\frac{b^2-4ac}{2a}$, 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 + s → int
R = s + i → int
R = l + f → float
R = i + f → float
R = c + i → int
R = c + s → int
R = l + d → double
R = f + d → double

b → byte f → float
s → short d → double
i → integer c → character
l → long

Increment and Decrement Operators
a++ , ++a → Increment operators → Data type
a-- , --a → Decrement operators → remains same

These will operate on all data types except booleans

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

a++ → first use the value and then increment
++a → 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'

URBAN EDGE

Chapter 2 - Practice Set

1. What will be the result of the following expression

float a = 7/4 * 9/2

2. Write a java program to encrypt a grade by adding 8 to it. Decrypt it to show the correct grade.

3. Use comparison operators to find out whether a given number is greater than the user entered number or not.

4. Write the following expression in a java program:

$$\frac{V^2 - u^2}{2as}$$

5. find the value of the following expression:

int x = 7
int a = 7*49/7 + 35/7

Value of a ?

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 lowercase 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!
String

9. name.endsWith("ry") → returns true if name ends with string "ry". true in this case.
String

10. name.charAt(2) → returns character at a given index position. r in this case!
int

11. name.indexOf(s) → returns the index of the given string.
str
For ex: name.indexOf("ar") returns 1 which is the first occurence 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

URBAN EDGE

Chapter 3 - Practice Set

1. Write a Java program to convert a String to lowercase

2. Write a Java program to replace spaces with underscores.

3. Write a Java program to fill in a letter template which looks like below:

   letter = "Dear <|name|>, Thans a lot"

   Replace <|name|> with a string (some name)

4. Write a Java program to detect double and triple spaces in a string

5. Write a program to format the following letter using escape sequence characters.

   letter = "Dear Harry, This Java Course is nice. Thanks"

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 aloo 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 :

$==$, $>=$, $>$, $<$, $<=$, $!=$

equals, greater than or eq to, 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 → true
Y && N = N	N → 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 → true
Y || N = Y	N → false
N || Y = Y	
N || N = N	

NOT Operator

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

! Y = N	Y → true
! N = Y	N → 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

Chapter 4 - Practice Set

1. What will be the output of this program:

```
int a = 10;
if ( a = 11)
        System.out.println("I am 11");
else
        System.out.println("I am not 11")
```

2. Write a program to find out whether a student is pass or fail; if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.

3. Calculate income tax paid by an employee to the government as per the slabs mentioned below:

Income Slab	Tax
2.5L - 5.0L	5%
5.0L - 10.0L	20%
Above 10.0L	30%

Note that there is no tax below 2.5L. Take input amount as an input from the user.

4. Write a Java program to find out the day of the week given the number [1 for Monday, 2 for Tuesday ... and so on!]

5 Write a Java program to find whether a year entered by the user is a leap year or not.

6 Write a program to find out the type of website from the Url

- Com → Commertial website
- org → organization website
- in → Indian website

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 immidiately 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.

URBAN EDGE

Chapter 5 - Practice Set

1. Write a program to print the following pattern

```
* * * *
* * *
* *
*
```

2. Write a program to sum first n even numbers using while loop.

3. Write a program to print multiplication table of a given number n.

4. Write a program to print multiplication table of 10 in reverse order.

5. Write a program to find factorial of a given number using for loops.

6. Repeat 5 using while loop

7. Repeat 1 using for/while loop

8. What can be done using one type of loop can also be done using the other two types of loops - True or False.

9. Write a program to calculate the sum of the numbers occuring in the multiplication table of 8.

URBAN EDGE

10 A do while loop is executed :
1> At least once
2> At least twice
3> At most once

11 Repeat 2 using for loop.

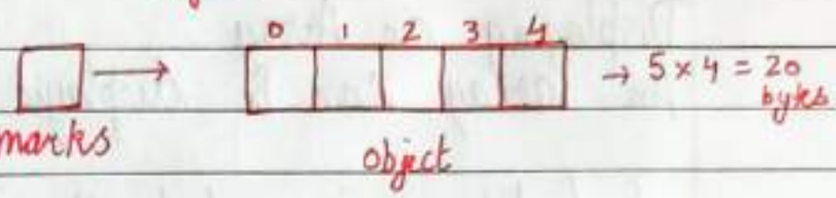
Chapter 6 - Arrays

Array is a collection of similar types of data

Use Case : Storing marks of 5 Students

int [] marks = new int [5] ⇒ [dataType ArrName;]

reference object



Accessing Array Elements

Array elements can be accessed as follows

marks [0] = 100

marks [1] = 70

⋮

marks [4] = 98

⇒ Note that index starts from 0

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

1. int [] marks; → Declaration!
   marks = new int [5]; → Memory Allocation!

2. int [] marks = new int [5]; → Declaration + Memory Allocation!

3. int [] marks = { 100, 70, 80, 71, 98 } → 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] Col 1	[1] Col 2	[2] Col 3
[0] Row 1	(0,0)	(0,1)	(0,2)
[1] Row 2	(1,0)	(1,1)	(1,2)

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

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

Chapter 6 - Practice Set

1. Create an array of 5 floats and calculate their sum.

2. Write a program to find out whether a given integer is present in an array or not.

3. Calculate the average marks from an array containing marks of all students in Physics using for-each loop.

4. Create a Java program to add two matrices of size 2 x 3.

5. Write a Java program to reverse an array

6. Write a Java program to find the maximum element in an array.

7. Write a Java program to find the minimum element in a Java array.

8. Write a Java program to find whether an array is sorted or not.

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 = Dont 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

```
int mySum( int a, int b) {
    int c = a+b;
    return c;
    }
```

→ Return type
→ 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();
obj.mySum(a, b);
```

→ Object Creation
→ 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 dont 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

$$factorial(n) = n * factorial(n-1) \quad \forall\, n \geq 1$$

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

Chapter 7 - Practice Set

1. Write a Java method to print multiplication table of a number n.

2. Write a program using functions to print the following pattern:

```
*
* *
* * *
* * * *
```

3. Write a recursive function to calculate sum of first n natural numbers

4. Write a function to print the following pattern

```
* * * *
* * *
* *
*
```

5. Write a function to print n^{th} term of fiboracci series using recursion.

6. Write a function to find average of a set of numbers passed as arguments

7. Repeat 4 using Recursion.

8. Repeat 2 using Recursion

9 Write a function to convert Celsius temperature into fahrenheit.

10 Repeat 3 using iterative approach.

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 8 - Practice Set

1. Create a class Employee with following properties and methods:
   - Salary (property) (int)
   - getSalary (method returning int)
   - name (property) (String)
   - getName (method returning String)
   - setName (method changing name)

2. Create a class cellphone with methods to print "ringing...", "Vibrating..." etc.

3. Create a class Square with a method to initialize its side, calculating area, perimeter etc.

4. Create a class Rectangle & repeat 3

5. Create a class TommyVecetti for Rockstar Games capable of hitting (print hitting..), running, firing etc.

6. Repeat 4 for a Circle.

Chapter 9 - Access Modifiers & Constructors

Access Modifiers

Specifier 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 9 - Practice Set

1. Create a class Cylinder and use getters and setters to set its radius and height.

2. Use ① to calculate surface area and Volume of the Cylinder.

3. Use a constructor and repeat ①

4. Overload a Constructor used to initialize a rectangle of length 4 and breadth 5 for using custom parameters.

5. Repeat ① for a sphere

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	Animal	Animal	Vehicle
↓	↓	↓	↓
Car	Dog	Cat	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 {
    // Code
}
```

→ Inheriting Dog from Animal Class!!

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

C_1 → Parent
↓
C_2 → child
↓
C_3 → Grand child

↓ constructors execute in top to bottom order!

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

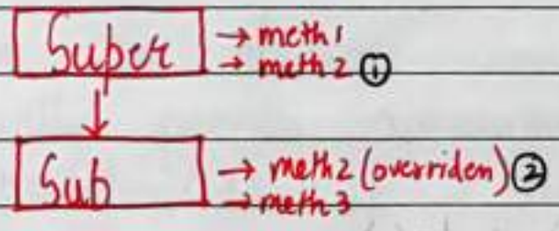
If the child class implements the same method present in the parent class again, it is known as method overriding

↳ Redefining method of super class! (in subclass)

When an object of subclass is created and the overrided method is called, the method which has been implemented in the subclass is called & its code is executed.

Dynamic method dispatch

Consider the following inheritance hierarchy



Scenario 1 → Super obj = new Sub() → Allowed ✓
obj. meth2() → ② is called (Method of object)
obj. meth3() → Not Allowed ✗

Scenario 2 → Sub obj = new Super() → Not Allowed ✗

This is known as Dynamic method dispatch and is used to acheive run time polymorphism in Java.

Chapter 10 - Practice Set

1. Create a class Circle and use inheritance to create another class Cylinder from it.

2. Create a class Rectangle and use inheritance to create another class Cuboid. Try to keep it as close to real world scenario as possible.

3. Create methods for area and Volume in ①

4. Create methods for area & volume in ②. Also create getters and setters

5. What is the order of Constructor execution for the following inheritance hierarchy:

Base
↓
Derived 1
↓
Derived 2

Derived2 obj = new Derived2();
Which constructor(s) will be executed & in what order?

Chapter 11 - Abstract Classes & Interfaces

What does Abstract (class) mean?

Abstract in english means → existing in thought or as an idea without concrete existance

Abstract method

A method that is declared without an implementation

```
abstract void moveTo (double x, double y)
```

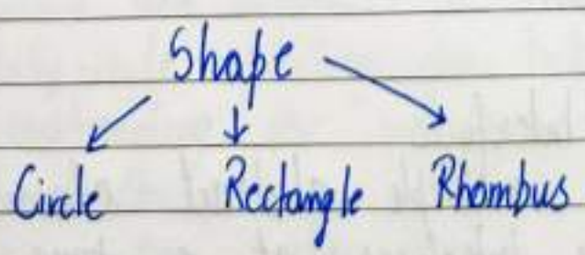
Abstract Class

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 doesnt, it must be declared abstract

An Example

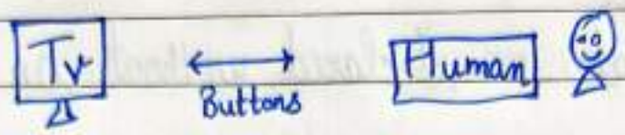


Note - It is possible to create reference of an abstract class
It is not possible to create an object of an abstract class

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 cant 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 acheived 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 on 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 compatibity 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

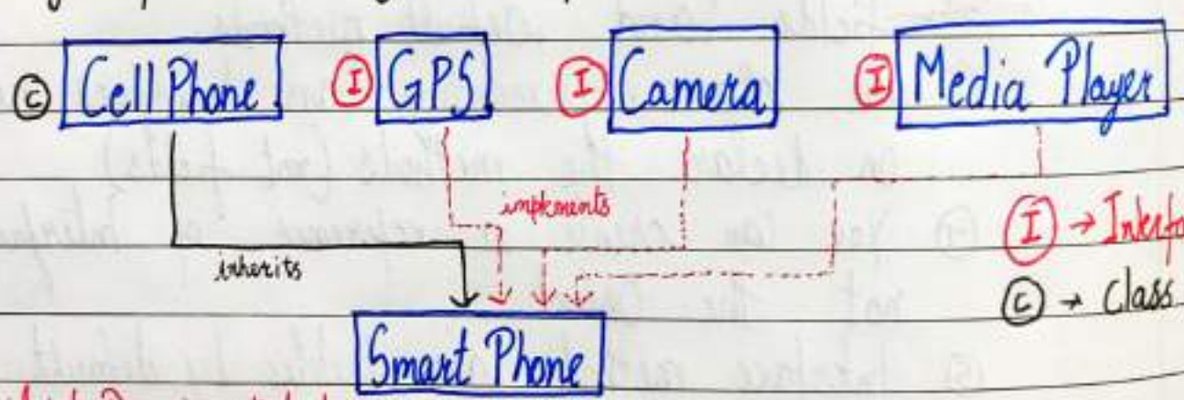
Interfaces can extend anothe interfaces:

```
public interface Interface1 {
        void meth1();
    }

public interface Interface2 extends Interface1 {
        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 11 - Practise Set

1. Create an abstract class Pen with methods write() and refill() as abstract methods

2. Use the Pen Class from Q1 to create a Concrete class FountainPen with additional method changeNib()

3. Create a class Monkey with jump() and bite() methods. Create a class Human which inherits this Monkey class and implements BasicAnimal interface with eat() and sleep methods.

4. Create a class TelePhone with ring(), lift() and disconnect() methods as abstract methods. Create another class SmartTelephone and demonstrate polymorphism

5. Demonstrate polymorphism using monkey class from Ques3.

6. Create an Interface TVRemote and use it to inherit another Interface SmartTvRemote.

7. Create a Class Tv which implements TvRemote interface from Q6

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	Compiler
* One statement at a time	* Entire program at a time
* Interpreter is needed everytime	* Once compiled it is not needed
* Partial execution if error	* No execution if an error occurs
* Easy for programmers	* Usually not as easy as Interpreted ones

Is Java Compiled or Interpreted?

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

Java file (harry.Java) —Compiled (using javac)→ Class file (harry.class → bytecode) → Can be used by Java interpreter

→ A JVM can be used to Interprete 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

Javac Harry.Java → Compiled
Java Harry.class → Interpreted

So fare the execution of our program was being managed by intelly 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

folder subfolder

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)

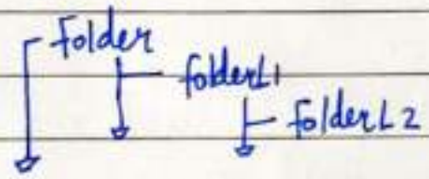
Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Default (no)	Y	Y	N	N
Private	Y	N	N	N

Chapter 12 - Practice Set

1. Create three classes Calculator, ScCalculator and HybridCalculator and group them into a package.

2. Use a built-in package in Java to write a class which displays a message (by using sout) after taking input from the user.

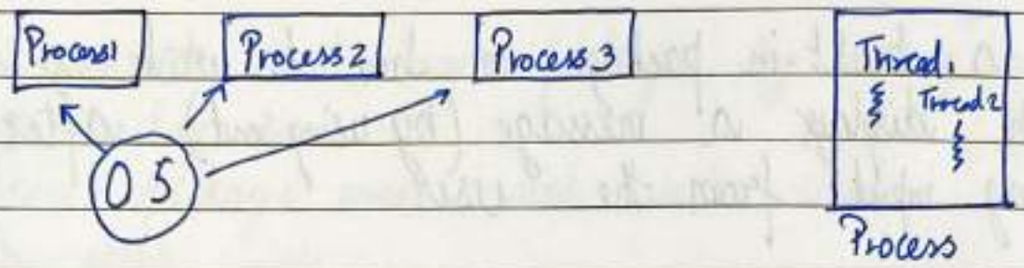
3. Create a package in class with three package levels folder, folderL1, folderL2



4. Prove that you cannot access default property but can access protected property from the subclass.

Chapter 13 - Multithreading

Multiprocessing and multithreading both are used to acheive multitasking



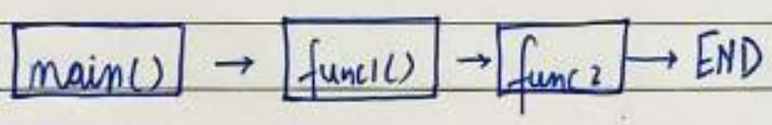
In a nut shell ...

→ Threads use shared memory area.
→ Threads ⇒ Faster Content switching
→ A Thread is light-weight whereas a process is heavyweight

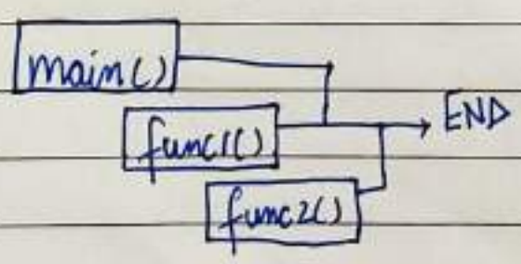
For Example → A word processor can have one thread running in foreground as an editor and another in the background auto saving the document!

Flow of Control in Java

1. Without threading:



2. With threading:

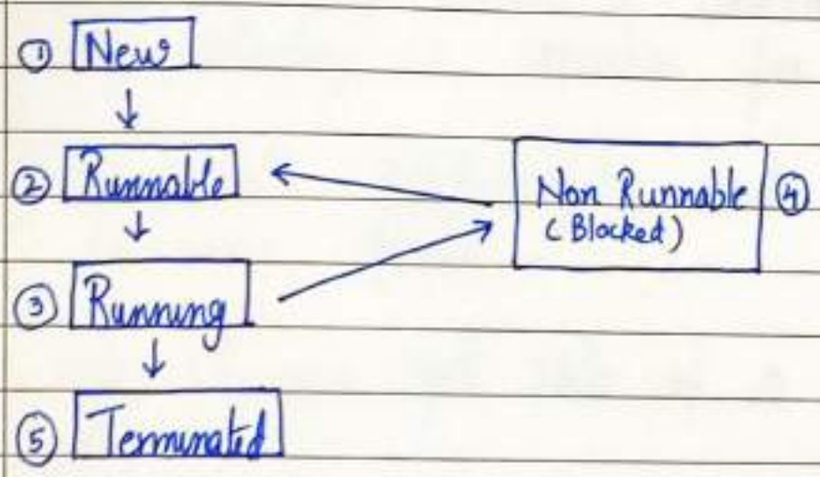


Creating a Thread

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 exitted

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 13 - Practise Set

1. Write a program to print "good morning" and "welcome" Continuously on the screen in Java using Threads.

2. Add a sleep method in welcome thread of question 1 to delay its execution for 200 ms.

3. Demonstrate getPriority() and SetPriority() methods in Java Threads.

4. How do you get state of a given thread in Java?

5. How do you get reference to the current thread in Java?

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

```
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 where as 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 Occuring Exceptions
Following are few commonly occuring exceptions in Java:
1. Null Pointer Exception
2. Arithmetic Exception
3. ArrayIndexOutofBound Exception
4. IllegalArgument Exception
5. NumberFormat Exception

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) {        → Handles all Exceptions of type IOException
    // Code
  }
Catch (ArithmeticException e) {  → Handles all Exceptions of type ArithmeticException
    // Code
  }
catch ( Exception e) {         → Handles all other Exceptions
    // Code
  }
```

Nested try-catch

We can nest multiple try-catch blocks as follows:

```
try {
    try {
      }
    Catch (Ex. e) {
      }
  }
Catch (Ex. e) {                ⟹ 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 {
        //overridden methods
        }
```

The Exception class has following important methods :

1. String toString() → executed when sout(e) is ran
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, its 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.

Chapter 14 - Practice Set

1. Write a Java program to demonstrate syntax, logical & runtime errors.

2. Write a Java program that prints "HaHa" during Arithmetic exception and "HeHe" during an Illegal argument exception

3. Write a program that allows you to keep accessing an array until a valid index is given. If max retries exceed 5 print "Error".

4. Modify program in Q3 to throw a custom Exception if max retries are reached.

5. Wrap the program in Q3 inside a method which throws your custom Exception.

Advanced Java - 1

Collections Framework

A collection represents a group of object.
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.