

Java By CodewithHarry

- ❑ 1. Introduction to Java + Installing Java JDK and IntelliJ IDEA for Java

Free YouTube Video

- ❑ 2. Basic Structure of a Java Program: Understanding our First Java Hello World Program

Free YouTube Video

- ❑ 3. Java Tutorial: Variables and Data Types in Java Programming

Free YouTube Video

- ❑ 4. Java Tutorial: Literals in Java

Free YouTube Video

- ❑ 5. Java Tutorial: Getting User Input in Java

Free YouTube Video

- ❑ 6. Java Programming Exercise 1: CBSE Board Percentage Calculator

Free YouTube Video

- ❑ 7. Java Tutorial: Chapter 1- Practice Set | Java Practice Problems With Solution

Free YouTube Video

- ❑ 8. Java Tutorial: Operators, Types of Operators & Expressions in Java

Free YouTube Video

- ❑ 9. Java Tutorial: Associativity of Operators in Java

Free YouTube Video

- ❑ 10. Java Tutorial: Data Type of Expressions & Increment/Decrement Operators

Free YouTube Video

- ❑ 11. Java Tutorial: Exercise 1 - Solutions + Shoutouts

Free YouTube Video

- ❑ 12. Java Tutorial: Chapter 2 - Practice Set (Java Practice Questions)

Free YouTube Video

- ❑ 13. Java Tutorial: Introduction to Strings

Free YouTube Video

- ❑ 14. Java Tutorial: String Methods in Java


Free YouTube Video

- ❑ 15. Java Practice Questions on Strings: Practice Set on Java Strings (Must Solve!)

Free YouTube Video

- ❑ 16. Java Conditionals: If-else Statement in Java


Free YouTube Video

-  17. Java Tutorial: Relational and Logical Operators in Java


Free YouTube Video

-  18. Java Tutorial: Switch Case Statements in Java


Free YouTube Video

-  19. Java Tutorial: Practice Questions On Conditionals & Switch Case


Free YouTube Video

-  20. Java Programming Exercise 2: Rock, Paper Scissors Game in Java


Free YouTube Video

-  21. Java Tutorial: While Loops in Java


Free YouTube Video

-  22. Java Tutorial: The do-while loop in Java


Free YouTube Video

-  23. Java Tutorial: The for Loop in Java


Free YouTube Video

-  24. Java Tutorial: break and continue in Java


Free YouTube Video

-  25. Java tutorial: Practice Questions on Loops

Free YouTube Video

-  26. Java Tutorial: Introduction to Arrays

Free YouTube Video

-  27. Java Tutorial: For Each Loop in Java


Free YouTube Video

-  28. Java Tutorial: Multidimensional Arrays in Java

Free YouTube Video

-  29. Java Tutorial: Practice Questions on Arrays in Java


Free YouTube Video

-  30. How to Make IntelliJ IDEA look Amazing!


Free YouTube Video

-  31. Java Tutorial: Methods in Java

Free YouTube Video

-  32. Java Tutorial: Method Overloading in Java

Free YouTube Video

-  33. Java Tutorial: Variable Arguments (VarArgs) in Java

Free YouTube Video

-  34. Java Tutorial: Recursion in Java


Free YouTube Video

-  35. Java Tutorial: Practice Questions on Java Methods

Free YouTube Video

-  36. Java Tutorial: Introduction to Object Oriented Programming


Free YouTube Video

-  37. Java Tutorial: Basic Terminologies in Object Oriented Programming


Free YouTube Video

-  38. Java Tutorial: Creating Our Own Java Class

Free YouTube Video

-  39. Java Tutorial: Basic Questions on Object Oriented Programming


Free YouTube Video

-  40. Java Tutorial: Access modifiers, getters & setters in Java


Free YouTube Video

-  41. Java Tutorial: Exercise 2 - Solution and Shoutouts

Free YouTube Video

-  42. Java Tutorial: Constructors in Java

Free YouTube Video

-  43. Java Exercise 3: Guess the Number (OOps Edition)

Free YouTube Video

-  44. Java Tutorial: Exercise on Access Modifiers and Constructors


Free YouTube Video

-  45. Inheritance in Java

Free YouTube Video

-  46. Constructors in Inheritance in Java

Free YouTube Video

-  47. this and super keyword in Java


Free YouTube Video

-  48. Method Overriding in Java


Free YouTube Video

-  49. Dynamic Method Dispatch in Java

Free YouTube Video

-  50. Java Tutorial: Exercise 3 - Solutions & Shoutouts


Free YouTube Video

-  51. Java Tutorial: Exercise 4 - Online Library


Free YouTube Video

-  52. Java Tutorial: Exercise & Practice Questions on Inheritance


Free YouTube Video

-  53. Java Tutorial: Abstract Class & Abstract Methods

Free YouTube Video

-  54. Java Tutorial: Introduction to Interfaces


Free YouTube Video

-  55. Java Tutorial: Abstract Classes Vs Interfaces

Free YouTube Video

-  56. Why multiple inheritance is not supported in java?


Free YouTube Video

-  57. Java Interfaces Example & Default Methods


Free YouTube Video

-  58. Inheritance in Interfaces


Free YouTube Video

-  59. Java Tutorial: Polymorphism in Interfaces


Free YouTube Video

-  60. Java Practice Questions on Abstract Classes & Interfaces

Free YouTube Video

-  61. Java Exercise 4: Solution & Shoutouts!


Free YouTube Video

-  62. Interpreted vs Compiled Languages!


Free YouTube Video

-  63. Is Java interpreted or compiled?

Free YouTube Video

-  64. Packages in Java

Free YouTube Video

-  65. Java Tutorial: Creating Packages in Java

Free YouTube Video

- ☐ 66. Access Modifiers in Java

Free YouTube Video

- ☐ 67. Practice Set on Java Package & Access Modifiers

Free YouTube Video

- ☐ 68. Java Exercise 5: Creating a Custom Package

Free YouTube Video

- ☐ 69. Multithreading in Java

Free YouTube Video

- ☐ 70. Creating a Thread by Extending Thread class

Free YouTube Video

- ☐ 71. Creating a Java Thread Using Runnable Interface

Free YouTube Video

- ☐ 72. Java Thread Life Cycle

Free YouTube Video

- ☐ 73. Constructors from Thread class in Java

Free YouTube Video

- ☐ 74. Java Thread Priorities

Free YouTube Video

- ☐ 75. Java Thread Methods

Free YouTube Video

- ☐ 76. Java Tutorial: Practice Questions on Thread

Free YouTube Video

- ☐ 77. Exercise 5: Solution & Shoutouts!

Free YouTube Video

- ☐ 78. Errors & Exception in Java

Free YouTube Video

- ☐ 79. Syntax Errors, Runtime Errors & Logical Errors in Java (Demo)

Free YouTube Video

- ☐ 80. Exceptions & Try-Catch Block in Java

Free YouTube Video

- ☐ 81. Handling Specific Exceptions in Java

Free YouTube Video

- ☐ 82. Nested Try-Catch in Java

Free YouTube Video

-  83. The Exception class in Java


Free YouTube Video

-  84. Throw vs Throws in Java


Free YouTube Video

-  85. Finally Block in Java & Why is it needed!

Free YouTube Video

-  86. Practice Set on Errors & Exceptions


Free YouTube Video

-  87. Java Exercise 6: Custom Calculator | Java Practice Question


Free YouTube Video

-  88. Java Collections Framework

Free YouTube Video

-  89. Collections Hierarchy in Java

Free YouTube Video

-  90. How to View Java Documentation (Correct Way)


Free YouTube Video

-  91. ArrayList in Java: Demo & Methods


Free YouTube Video

-  92. LinkedList in Java: Demo & Methods


Free YouTube Video

-  93. ArrayDeque in Java


Free YouTube Video

-  94. Hashing in Java

Free YouTube Video

-  95. HashSet in Java

Free YouTube Video

-  96. Date and Time in Java


Free YouTube Video

-  97. The Date Class in Java

Free YouTube Video

-  98. Calendar Class in Java

Free YouTube Video

-  99. GregorianCalendar class & TimeZone in java

Free YouTube Video

-  100. java.time API - Classes & Methods

Free YouTube Video

-  101. DateTimeFormatter in Java


Free YouTube Video

-  102. Advanced Java Practice Set


Free YouTube Video

-  103. Java Exercise 6: Solution | Custom Calculator


Free YouTube Video

-  104. Java Exercise 7: Library Management System in Java

Free YouTube Video

-  105. Generating our own JavaDocs for our Package


Free YouTube Video

-  106. Javadocs: Tags for Documenting Classes


Free YouTube Video

-  107. Javadocs: Method Tags For Generating java Documentation

Free YouTube Video

-  108. Annotations in Java


Free YouTube Video

-  109. Java Anonymous Classes & Lambda Expressions


Free YouTube Video

-  110. Java Generics


Free YouTube Video

-  111. File Handling in Java

Free YouTube Video

-  112. Advanced Java 2 - Practice Set

Free YouTube Video

-  113. Exercise 7: Solutions + Shoutouts

Free YouTube Video

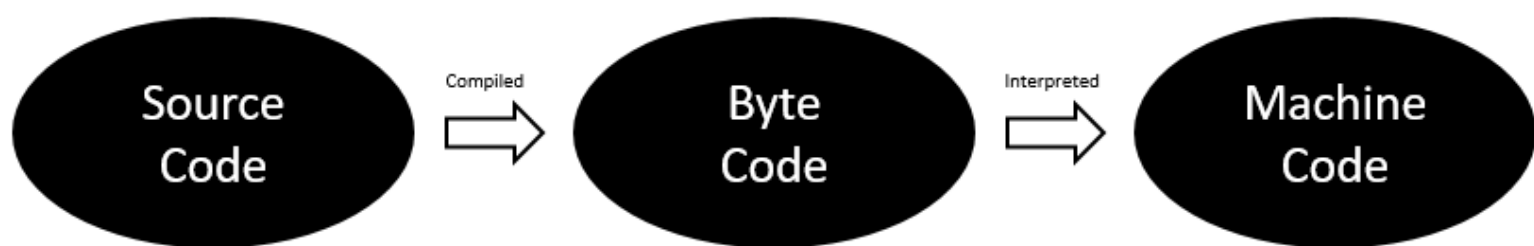
Introduction to Java + Installing Java JDK and IntelliJ IDEA for Java

- Java is one of the most popular programming languages because it is used in various tech fields like app development, web development, client-server applications, etc.
- Java is an object-oriented programming language developed by Sun Microsystems of the USA in 1991.

- It was originally called Oak by James Goslin. He was one of the inventors of Java.
- Java = Purely Object-Oriented.

How Java Works?

- The source code in Java is first compiled into the bytecode.
- Then the Java Virtual Machine(JVM) compiles the bytecode to the machine code.



Java Installation:

Step 1: Downloading JDK

- JDK stands for Java Development Kit. It contains Java Virtual Machine(JVM) and Java Runtime Environment(JRE).
- **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.
- [Click here](#), and you will be redirected to the official download page of JDK.
- Select your operating system and download the file(executable file in case of Windows).

Video tutorial for downloading JDK :

Step 2: Installing JDK

- Once the executable file is downloaded successfully, right-click and open the file.
- The executable file will start executing.
- Keep clicking on the **Next** button to install the JDK in default settings.

Video tutorial for installing JDK :

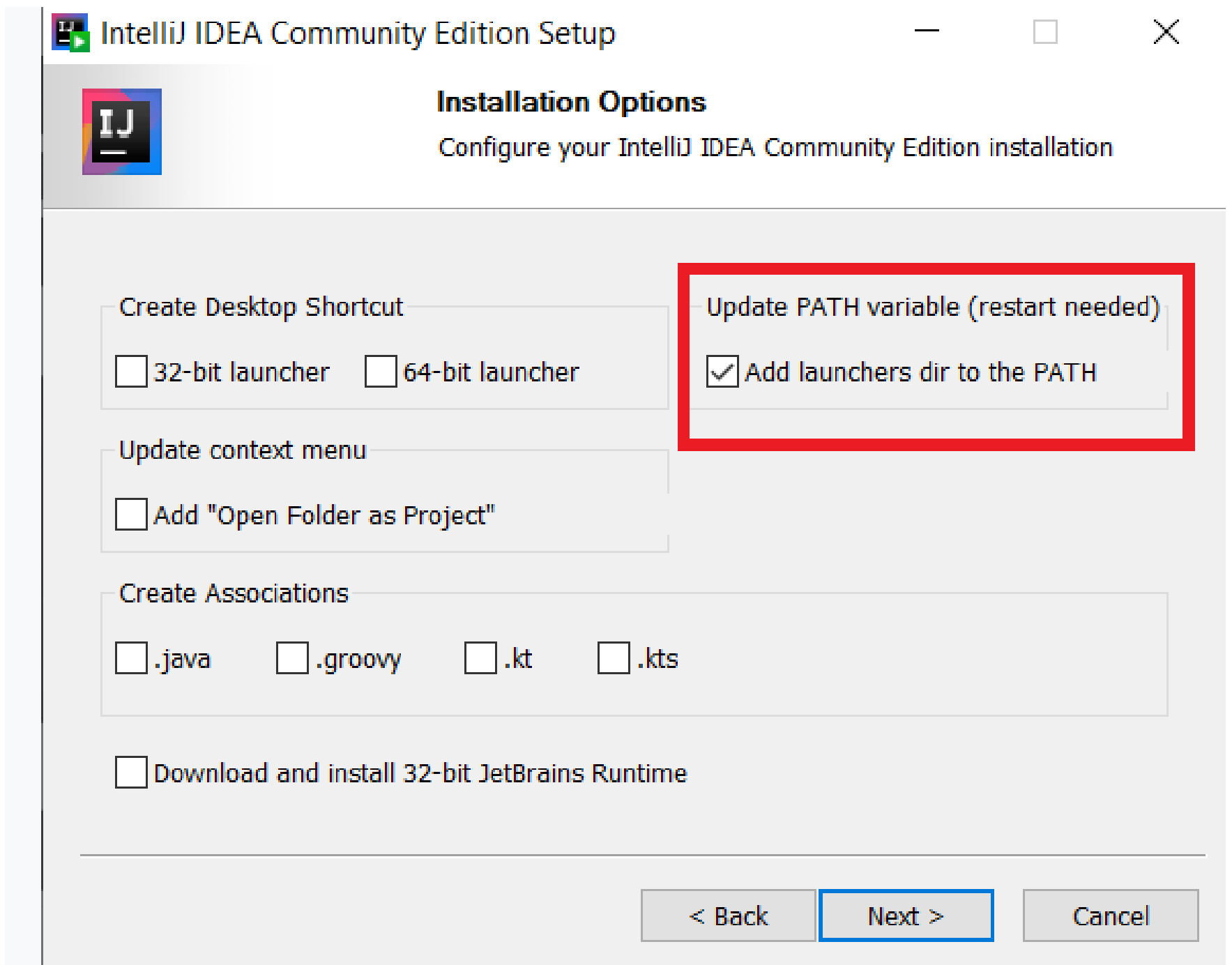
Step 3: Downloading IntelliJ IDEA :

- We need an Integrated Development Environment(IDE) to write and debug our code easily.
- IntelliJ IDEA is the best-suited IDE for writing Java code.
- [Click here](#), and you will be redirected to the official download page of IntelliJ IDEA.
- Download the **Community Version** of the IntelliJ IDEA as it is free to use.

Video tutorial for downloading IntelliJ IDEA:

Step 4: Installing IntelliJ IDEA :

- Once the download completes, open the .exe file, and the installation process will begin.
- Click on the "Next" button to install the IntelliJ IDEA in the default location.
- Do not forget to check "**Add launchers dir to the PATH,**" as shown in the below image.



- After this, click on the "Next" button and then click on the "Install" button.

Video tutorial for installing IntelliJ IDEA:

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Basic Structure of a Java Program: Understanding our First Java Hello World Program

Basic Structure of a Java Program

```
package com.company; // Groups classes
public class Main{    // Entrypoint into the application
    public static void main(String[]args){
        System.out.println("Hello World");
    }
}
```

Copy

Working of the "Hello World" program shown above :

1. package com.company :

- Packages are used to group the related classes.
- The "Package" keyword is used to create packages in Java.
- Here, com.company is the name of our package.

2. public class Main :

- In Java, every program must contain a class.
- The filename and name of the class should be the same.
- Here, we've created a class named "Main".
- It is the entry point to the application.

3. public static void main(String[]args){..} :

- This is the main() method of our Java program.
- Every Java program must contain the main() method.

4. System.out.println("Hello World");

- The above code is used to display the output on the screen.
- Anything passed inside the inverted commas is printed on the screen as plain text.

Naming Conventions

- For classes, we use Pascal Convention. The first and Subsequent characters from a word are capital letters (uppercase).
Example: Main, MyScanner, MyEmployee, CodeWithHarry
- For functions and variables, we use camelCaseConvention. Here the first character is lowercase, and the subsequent characters are uppercase like myScanner, myMarks, CodeWithHarry

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Variables and Data Types in Java Programming

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. This set of these rules is called syntax. It’s like Vocabulary and 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; (Here, int is a data type, the number is the variable name, and 8 is the value it contains/stores).

Rules for declaring a variable name

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

- Must not begin with a digit. (E.g., 1arry is an invalid variable)
- Name is case sensitive. (Harry and harry are different)
- Should not be a keyword (like Void).
- White space is not allowed. (int Code With Harry is invalid)
- 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, i.e., variables must be declared before use. Java supports 8 primitive data types:

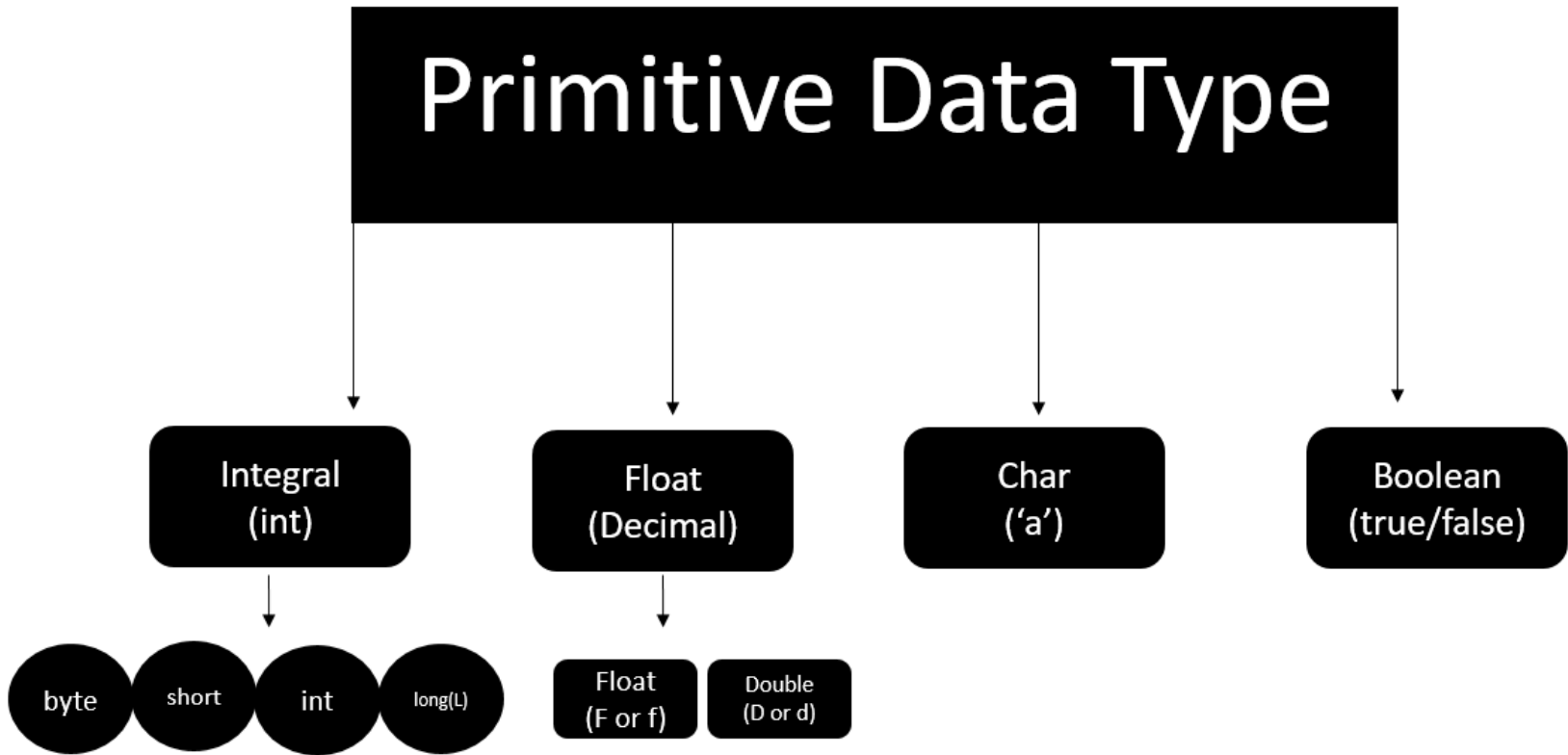
Data Type	Size	Value Range
1. Byte	1 byte	-128 to 127
2. short	1 byte	-32,768 to 32,767
3. int	2 byte	-2,147,483,648 to 2,147,483,647
4. float	4 byte	3.40282347 x 10 ³⁸ to 1.40239846 x 10 ⁻⁴⁵
5. long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
6. double	8 byte	1.7976931348623157 x 10 ³⁰⁸ , 4.9406564584124654 x 10 ⁻³²⁴

7. char	2 byte	0 to 65,535
8. boolean	Depends on JVM	True or 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.



Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Literals in Java

Literals

A constant value that can be assigned to the variable is called 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 that are reserved and used by the Java compiler. They cannot be used as an Identifier.

{You can visit docs.oracle.com for a comprehensive list}

Code as Described in the Video

```
package com.company;

public class CWH_04_literals {
    public static void main(String[] args) {
        byte age = 34;
        int age2 = 56;
        short age3 = 87;
        long ageDino = 5666666666666L;
        char ch = 'A';
    }
}
```

```
float f1 = 5.6f;
double d1 = 4.66;

boolean a = true;
System.out.print(age);
String str = "Harry";
System.out.println(str);

}
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Getting User Input in Java

Reading data from the Keyboard :

- Scanner class of java.util package is used to take input from the user's keyboard. The Scanner class has many methods for taking input from the user depending upon the type of input. To use any of the methods of the Scanner class, first, we need to create an object of the Scanner class as shown in the below example :

- `import java.util.Scanner;` // Importing the Scanner class

```
Scanner sc = new Scanner(System.in); //Creating an object named "sc" of the Scanner class.
```

Copy

Taking an integer input from the keyboard :

```
Scanner S = new Scanner(System.in); //(Read from the keyboard)
int a = S.nextInt(); //(Method to read from the keyboard)
```

Copy

Code as Described in the Video

```
package com.company;
import java.util.Scanner;

public class CWH_05_TakingInpu {
    public static void main(String[] args) {
        System.out.println("Taking Input From the User");
        Scanner sc = new Scanner(System.in);
//        System.out.println("Enter number 1");
//        int a = sc.nextInt();
//        float a = sc.nextFloat();
//        System.out.println("Enter number 2");
//        int b = sc.nextInt();
//        float b = sc.nextFloat();

//        int sum = a +b;
//        float sum = a +b;
//        System.out.println("The sum of these numbers is");
//        System.out.println(sum);
//        boolean b1 = sc.hasNextInt();
//        System.out.println(b1);
//        String str = sc.next();
        String str = sc.nextLine();
    }
}
```

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

    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Programming Exercise 1: CBSE Board Percentage Calculator

Exercise 1.1

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

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Chapter 1- Practice Set | Java Practice Problems With Solution

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 that 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 an integer or not.

Code as Described in the Video

```
package com.company;

import java.util.Scanner;

public class CWH_Ch1_PS {
    public static void main(String[] args) {
//        Question1
//        int a = 4;
//        int b = 17;
//        int c =6;
//        int sum = a + b+c;
//        System.out.println(sum);

//        Question2
//        float subject1 = 45;
//        float subject2 = 95;
//        float subject3 = 48;
//        float cgpa = (subject1 + subject2 +subject3)/30;
//        System.out.println(cgpa);

//        Question 3
//        System.out.println("What is your name");
//        Scanner sc = new Scanner(System.in);
//        String name = sc.next();
//        System.out.println("Hello " + name + " have a good day!");

//        Question 5
```

```
        System.out.println("Enter your number");
        Scanner sc = new Scanner(System.in);
        System.out.println(sc.hasNextInt());
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Operators, Types of Operators & Expressions in Java

- An operator is a symbol that the compiler to perform a specific operation on operands.
- Example : $a + b = c$
- In the above example, 'a' and 'b' are operands on which the '+' operator is applied.

Types of operators :

1. Arithmetic Operators :

- Arithmetic operators are used to perform mathematical operations such as addition, division, etc on expressions.
- Arithmetic operators cannot work with Booleans.
- % operator can work on floats and doubles.
- Let $x=7$ and $y=2$

Operator	Description	Example
+ (Addition)	Used to add two numbers	$x + y = 9$
- (Subtraction)	Used to subtract the right-hand side value from the left-hand side value	$x - y = 5$
* (Multiplication)	Used to multiply two values.	$x * y = 14$
/ (Division)	Used to divide left-hand Value by right-hand value.	$x / y = 3$
% (Modulus)	Used to print the remainder after dividing the left-hand side value from the right-hand side value.	$x \% y = 1$
++ (Increment)	Increases the value of operand by 1.	$x++ = 8$
-- (Decrement)	Decreases the value of operand by 1.	$y-- = 1$

2. Comparison Operators :

- As the name suggests, these operators are used to compare two operands.
- Let $x=7$ and $y=2$

Operator	Description	Example
== (Equal to)	Checks if two operands are equal. Returns a boolean value.	$x == y \rightarrow$ False
!= (Not equal	Checks if two operands are not equal. Returns a boolean value.	$x != y \rightarrow$ True
> (Greater than)	Checks if the left-hand side value is greater than the right-hand side value. Returns a boolean value.	$x > y \rightarrow$ True
< (Less than)	Checks if the left-hand side value is smaller than the right-hand side value. Returns a boolean value.	$x < y \rightarrow$ False
>=(Greater than or equal to)	Checks if the left-hand side value is greater than or equal to the right-hand side value. Returns a boolean value.	$x >= y \rightarrow$ True
<= (Less than or equal to)	Checks if the left-hand side value is less than or equal to the right-hand side value. Returns a boolean value.	$x <= y \rightarrow$ >False

3. Logical Operators :

- These operators determine the logic in an expression containing two or more values or variables.
- Let $x = 8$ and $y =2$

&& (logical and)	Returns true if both operands are true.	$x < y \ \&\& \ x != y \rightarrow$ True
(logical or)	Returns true if any of the operand is true.	$x < y \ \&\& \ x == y \rightarrow$ True
! (logical not)	Returns true if the result of the expression is false and vice-versa	$!(x < y \ \&\& \ x == y) \rightarrow$ False

4. Bitwise Operators :

- These operators perform the operations on every bit of a number.
- Let x =2 and y=3. So 2 in binary is 100, and 3 is 011.

Operator	Description	Example
& (bitwise and)	1&1 =1, 0&1=0,1&0=0,1&1=1, 0&0 =0	(A & B) = (100 & 011) = 000
(bitwise or)	1&0 =1, 0&1=1,1&1=1, 0&0=0	(A B) = (100 011) = 111
^ (bitwise XOR)	1&0 =1, 0&1=1,1&1=0, 0&0=0	(A ^ B) = (100 ^ 011) = 111
<< (left shift)	This operator moves the value left by the number of bits specified.	13<<2 = 52(decimal)
>> (right shift)	This operator moves the value left by the number of bits specified.	13>>2 = 3(decimal)

Precedence of operators

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

In case we like to change this order, we use parenthesis ().

Code as Described in the Video

```
package com.company;

public class CWH_Ch2_Operators {
    public static void main(String[] args) {
        // 1. Arithmetic Operators
        int a = 4;
        // int b = 6 % a; // Modulo Operator
        // 4.8%1.1 --> Returns Decimal Remainder

        // 2. Assignment Operators
        int b = 9;
        b *= 3;
        System.out.println(b);

        // 3. Comparison Operators
        // System.out.println(64<6);

        // 4. Logical Operators
        // System.out.println(64>5 && 64>98);
        System.out.println(64>5 || 64>98);

        // 5. Bitwise Operators
        System.out.println(2&3);
        //      10
        //      11
        //      ----
        //      10
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Associativity of Operators in Java

Associativity

Associativity tells the direction of the execution of operators. It can either be left to right or vice versa.

/ * -> L to R

+ - -> L to R

++, = -> R to L

Here is the precedence and associativity table which makes it easy for you to understand these topics better:

	Operator	Associativity	Precedence
() [] . ->	Function call Array subscript Dot (Member of structure) Arrow (Member of structure)	Left-to-Right	Highest 14
! ~ - ++ -- & * (type) sizeof	Logical NOT One's-complement Unary minus (Negation) Increment Decrement Address-of Indirection Cast Sizeof	Right-to-Left	13
* / %	Multiplication Division Modulus (Remainder)	Left-to-Right	12
+ -	Addition Subtraction	Left-to-Right	11
<< >>	Left-shift Right-shift	Left-to-Right	10
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left-to-Right	8
== !=	Equal to Not equal to	Left-to-Right	8
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, += *=, etc.	Assignment operators	Right-to-Left	1
,	Comma	Left-to-Right	Lowest 0

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

$$\frac{x - y}{2}$$

$$\frac{b^2 - 4ac}{2a}$$

$$V^2 - u^2$$

$$a * b - d$$

Code as Described in the Video

```
package com.company;

public class cwh_09_ch2_op_pre {
    public static void main(String[] args) {
        // Precedence & Associativity

        //int a = 6*5-34/2;
        /*
        Highest precedence goes to * and /. They are then evaluated on the basis
        of left to right associativity
            =30-34/2
            =30-17
        */
    }
}
```



```

        =13
    */
    //int b = 60/5-34*2;
    /*
        = 12-34*2
        =12-68
        =-56
    */

    //System.out.println(a);
    //System.out.println(b);

    // Quick Quiz
    int x =6;
    int y = 1;
    //  int k = x * y/2;

    int b = 0;
    int c = 0;
    int a = 10;
    int k = b*b - (4*a*c)/(2*a);
    System.out.println(k);

}
}

```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Data Type of Expressions & Increment/Decrement Operators

Resulting data type after arithmetic operation

- Result = byte + short -> integer
- Result = short + integer -> integer
- Result = long + float -> float
- Result = integer + float -> float
- Result = character + integer -> integer
- Result = character + short -> integer
- Result = long + double -> double
- Result = float + double -> double

Increment and Decrement operators

- a++, ++a (Increment Operators)
- a--, --a (Decrement Operators)

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).

1. int y=7;
2. int x = ++y*8;
3. value of x?
4. char a = 'B';

5. a++; (a is not 'C')

Code as Described in the Video

```
package com.company;

public class cwh_10_resulting_data_type {
    public static void main(String[] args) {
        /* byte x = 5;
        int y = 6;
        short z = 8;
        int a = y + z;
        float b = 6.54f + x;
        System.out.println(b); */

        // Increment and Decrement Operators
        int i = 56;
        // int b = i++; // first b is assigned i (56) then i is incremented
        int j = 67;
        int c = ++j; // first j is incremented then c is assigned j (68)
        System.out.println(i++);
        System.out.println(i);
        System.out.println(++i);
        System.out.println(i);
        int y = 7;
        System.out.println( ++y *8);
        char ch = 'a';
        System.out.println(++ch);
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise 1 - Solutions + Shoutouts

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

Code Solution:

```
package com.company;
import java.util.Scanner;

public class cwh_11_ex1_sol {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        System.out.println("Enter your Physics marks : ");
        int physics = scan.nextInt();
        System.out.println("Enter your English marks : ");
        int English = scan.nextInt();
        System.out.println("Enter your Chemistry marks : ");
        int chemistry = scan.nextInt();
        System.out.println("Enter your Mathematics marks : ");
```

```

        int mathematics = scan.nextInt();
        System.out.println("Enter your Computer Science marks : ");
        int computer = scan.nextInt();

        float percentage = ((physics + English + chemistry + mathematics + computer)/500.0f)*100;

        System.out.println("percentage : ");
        System.out.println(percentage);

    }
}

```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Chapter 2 - Practice Set (Java Practice Questions)

1. What will be the result of the following expression:

```
float a = 7/4 * 9/2
```

Copy

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:

```
(v^2-u^2)/2as
```

Copy

5. Find the value of 'a' in expression given below :

```
6. int x = 7
```

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

Copy

Code Solution:

```

package com.company;

public class cwh_12_ps2_pr01 {
    public static void main(String[] args) {
        float a = 7/4.0f * 9/2.0f;
        System.out.println(a);
    }
}

```

Copy

```

package com.company;

public class cwh_12_ps2_pr02 {
    public static void main(String[] args) {
        char grade = 'B';
        grade = (char)(grade + 8);
        System.out.println(grade);
        // Decrypting the grade
        grade = (char)(grade - 8);
        System.out.println(grade);
    }
}

```

```
}
```

Copy

```
package com.company;
import java.util.Scanner;

public class cwh_12_ps2_pr_03 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        System.out.println(a>8);
        System.out.println(7*49/7+35/7);
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Introduction to Strings

- A string is a sequence of characters.
- Strings are objects that represent a char array. For example :
- `char[] str = {'H', 'A', 'R', 'R', 'Y'};`

```
String s = new String(str);
```

Copy

is same as :

```
String s = "Harry";
```

Copy

- Strings are immutable and cannot be changed.
- `java.lang.String` class is used to create a String object.
- The string is a class but can be used as a data type.

Syntax of strings in Java :

```
String <String_name> = "<sequence_of_string>";
```

Copy

Example :

```
String str = "CodeWithHarry";
```

Copy

In the above example, str is a reference, and “CodeWithHarrrt” is an object.

Different ways to create a string in Java :

In Java, strings can be created in two ways :

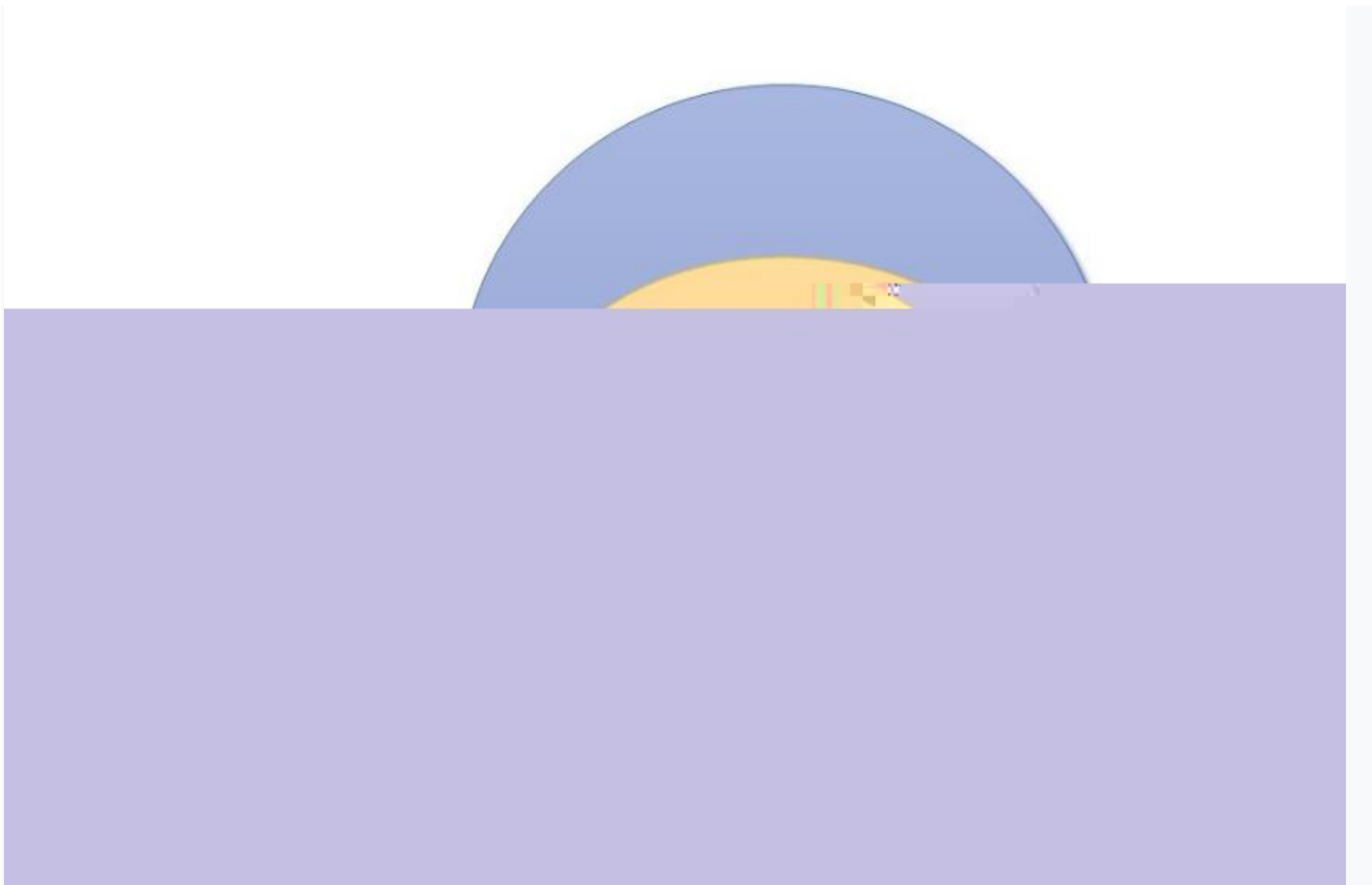
1. By using string literal
2. By using the new

Creating String using String literal :

```
String s1= "String literal"
```

Copy

We use double quotes("") to create string using string literal. Before creating a new string instance, JVM verifies if the same string is already present in the string pool or not. If it is already present, then JVM returns a reference to the pooled instance otherwise, a new string instance is created.



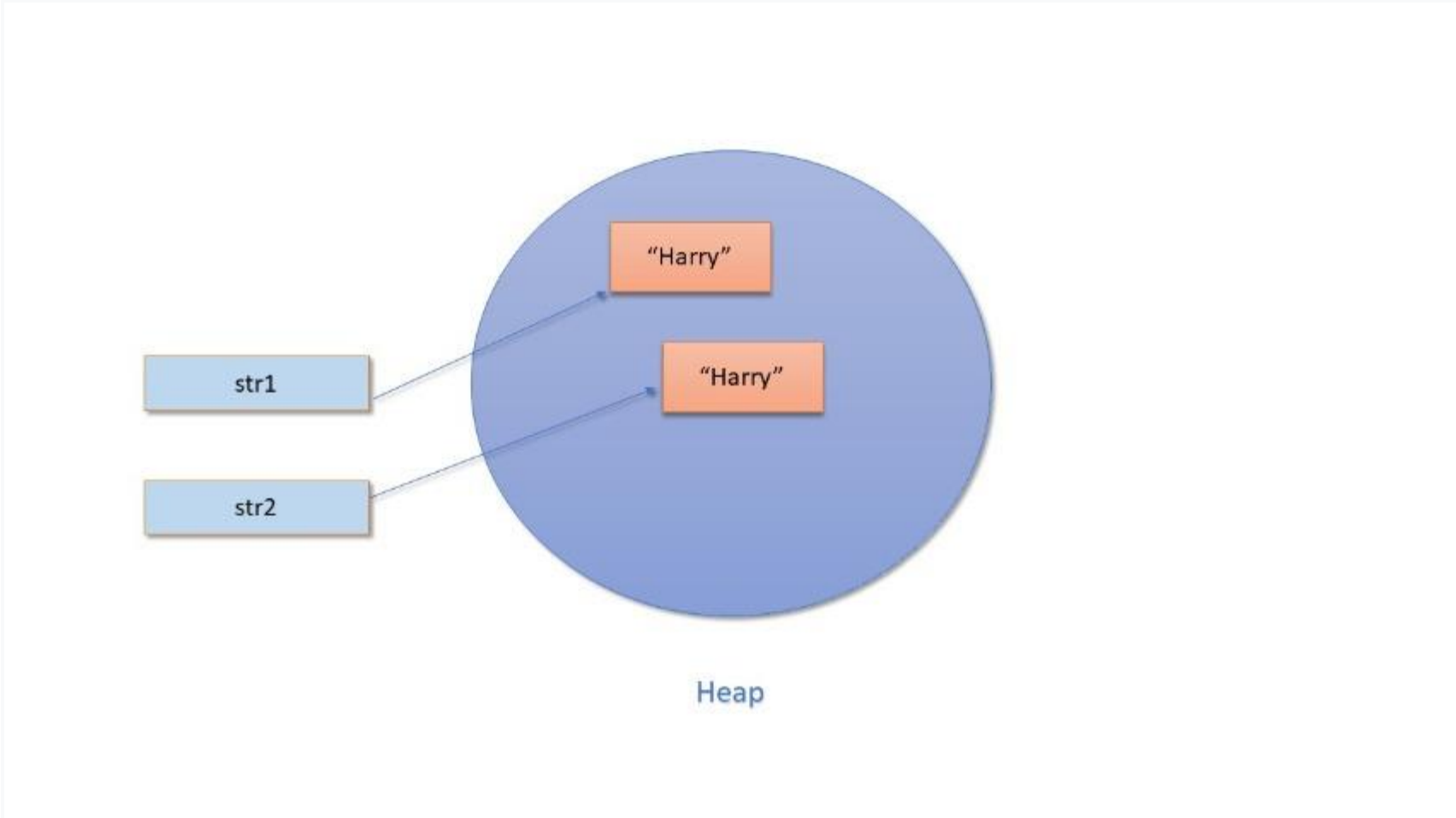
In the above diagram, notice that string "Harry" is already present in the string pool, which is pointed by the str1. When we try to create the same string object using str2, JVM finds that string object with the value "Harry" is already present in the string pool; therefore, instead of creating a new object, a reference to the same object is returned.

Creating String using new :

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

Copy

When we create a string using "new", a new object is always created in the heap memory.



In the above diagram, you can see that although the value of both string objects is the same, i.e., "Harry" still two different objects are created, and they are referred by two different reference variables, i.e., str1 and str2.

See the examples given below to get a better understanding of String literal and String object :

```
String str1 = "CodeWithHarry";
String str2 = "CodeWithHarry"
System.out.println(str1 == str2);
```

Copy

Output :

```
True
```

Copy

Returns true because str1 and str2 are referencing the same object present in the string constant pool. Now, let's see the case of the String object :

```
String str1 = new String("Keep coding");
String str2 = new String("Keep coding");
System.out.println(str1 == str2);
```

Copy

Output :

```
False
```

Copy

Although the value of both the string object is the same, still false is displayed as output because str1 and str2 are two different string objects created in the heap. That's why it is not considered a good practice to compare two strings using the == operator. Always use the equals() method to compare two strings in Java.

Different ways to print in Java :

We can use the following ways to print in Java:

- System.out.print() // No newline at the end
- System.out.println() // Prints a new line at the end
- System.out.printf()
- System.out.format()

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

Copy

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

Code as written in the video

```
package com.company;
import java.util.Scanner;

public class cwh_13_strings {
    public static void main(String[] args) {
        // String name = new String("Harry");
        // String name = "Harry";
        // System.out.print("The name is: ");
        // System.out.print(name);
        int a = 6;
        float b = 5.6454f;
        System.out.printf("The value of a is %d and value of b is %8.2f", a, b);
        //System.out.format("The value of a is %d and value of b is %f", a, b);
        Scanner sc = new Scanner(System.in);
//        String st = sc.next();
//        String st = sc.nextLine();
//        System.out.println(st);
```

```
    }  
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: String Methods in Java

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

Some of the commonly used String methods are:

```
String name = "Harry";
```

Copy

(Indexes of the above string are as follows: 0-H, 1-a, 2-r, 3-r, 4-y)

Method	Description
1. length()	Returns the length of String name. (5 in this case)
2. toLowerCase()	Converts all the characters of the string to the lower case letters.
3. toUpperCase()	Converts all the characters of the string to the upper case letters.
4. trim()	Returns a new String after removing all the leading and trailing spaces from the original string.
5. substring(int start)	Returns a substring from start to the end. Substring(3) returns “ry”. [Note that indexing starts from 0]
6. substring(int start, int end)	Returns a substring from the start index to the end index. The start index is included, and the end is excluded.
7. replace(‘r’, ‘p’)	Returns a new string after replacing r with p. Happy is returned in this case. (This method takes char as argument)
8. startsWith(“Ha”)	Returns true if the name starts with the string “Ha”. (True in this case)
9. endsWith(“ry”)	Returns true if the name ends with the string “ry”. (True in this case)
10. charAt(2)	Returns the character at a given index position. (r in this case)
11. indexOf(“s”)	Returns the index of the first occurrence of the specified character in the given string.
12. lastIndexOf(“r”)	Returns the last index of the specified character from the given string. (3 in this case)
13. equals(“Harry”)	Returns true if the given string is equal to “Harry” false otherwise [Case sensitive]
14.equalsIgnoreCase(“harry”)	Returns true if two strings are equal, ignoring the case of characters.

Escape Sequence Characters :

- The sequence of characters after backslash ‘\’ = Escape Sequence Characters
- Escape Sequence Characters consist of more than one character but represent one character when used within the strings.
- Examples: \n (newline), \t (tab), \' (single quote), \\ (backslash), etc.

Code as described in the video

```
package com.company;  
  
public class cwh_14_string_methods {  
    public static void main(String[] args) {  
        String name = "Harry";  
        // System.out.println(name);  
        int value = name.length();  
        //System.out.println(value);  
  
        //String lstring = name.toLowerCase();  
        //System.out.println(lstring);  
  
        //String ustring = name.toUpperCase();  
        //System.out.println(ustring);  
  
        //String nonTrimmedString = "    Harry    ";
```

```

        //System.out.println(nonTrimmedString);

        //String trimmedString = nonTrimmedString.trim();
        //System.out.println(trimmedString);

        //System.out.println(name.substring(1));
        //System.out.println(name.substring(1,5));

        //System.out.println(name.replace('r', 'p'));
        //System.out.println(name.replace("r", "ier"));

        //System.out.println(name.startsWith("Har"));
        //System.out.println(name.endsWith("dd"));

        //System.out.println(name.charAt(4));

        //String modifiedName = "Harryrrryrry";
        //System.out.println(modifiedName.indexOf("rry"));
        //System.out.println(modifiedName.indexOf("rry", 4));
        //System.out.println(modifiedName.lastIndexOf("rry", 7));

        //System.out.println(name.equals("Harry"));
        System.out.println(name.equalsIgnoreCase("HarRY"));

        System.out.println("I am escape sequence\\tdouble quote");

    }
}

```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Practice Questions on Strings: Practice Set on Java Strings (Must Solve!)

Today we will solve some of the best problems in Java related to strings! Here is the 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|>, Thanks a lot”

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

```

Copy

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”

Code Solution:

```

package com.company;

```



```

public class cwh_15_ps3 {
    public static void main(String[] args) {
        // Problem 1
        //String name = "Jack Parker";
        //name = name.toLowerCase();
        //System.out.println(name);

        // Problem 2
        //String text = "To My      Friend";
        //text = text.replace(" ", "_");
        //System.out.println(text);

        // Problem 3
        String letter = "Dear <|name|>, Thanks a lot!";
        letter = letter.replace("<|name|>", "Sachin");
        System.out.println(letter);

        // Problem 4
        String myString = "This string contains double and  triple spaces";
        System.out.println(myString.indexOf("  "));
        System.out.println(myString.indexOf("   "));

        // Problem 5
        String myLetter = "Dear Harry,\n\tThis Java Course is Nice.\nThanks!";
        System.out.println(myLetter);

    }
}

```

Copy

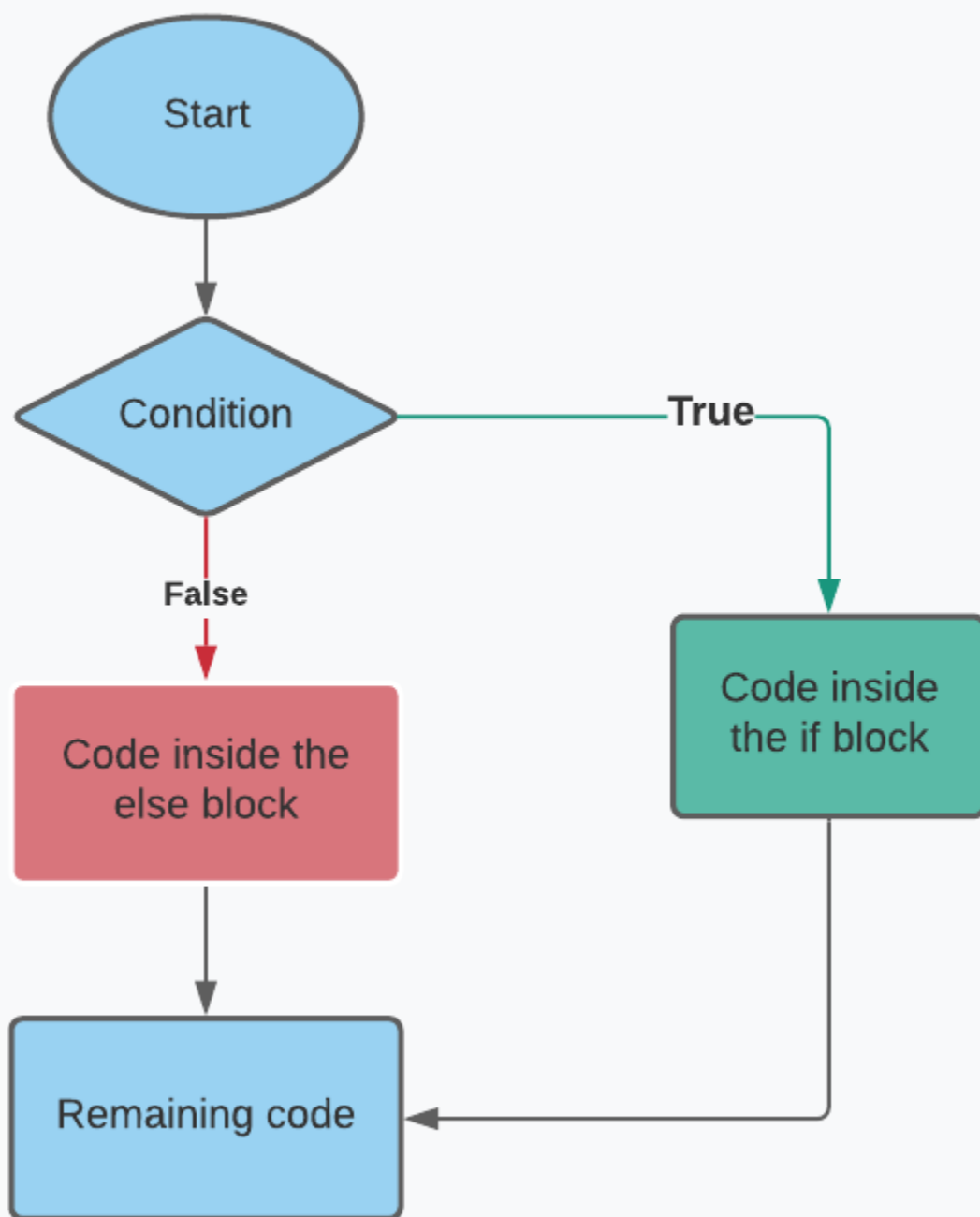
Practice Set: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Conditionals: If-else Statement in Java

Sometimes we want to drink coffee when we feel sleepy. Sometimes, we order junk food if it is our friend's birthday. You might want to buy an umbrella if it's raining. All these decisions depend on a certain condition being met. Similar to real life, we can execute some instructions only when a condition is met in programming also. If-else block is used to check conditions and execute a particular section of code for a specific condition.

Flow control of if-else in Java :



Decision-making instructions in Java

- If-Else Statement
- Switch Statement

If-Else Statement

Syntax of If-else statement in Java :

```
/* if (condition-to-be-checked) {  
    statements-if-condition-true;  
}  
else {  
    statements-if-condition-false;  
} */
```

Copy

Example:

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

Copy

Output:

You can drive

Copy

If-else ladder :

- 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 (condition1) {  
  
    //Statements;  
  
else if {  
  
    // Statements;  
  
}  
  
else {  
  
    //Statements  
  
} */
```

Copy

Note that the else block is optional.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Relational and Logical Operators in Java

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 equals to)
- > (greater than)
- < (less than)
- <= (less than or equals to)
- != (not equals)

Note: '=' is used for an assignment whereas '==' is used for equality check. The condition can be either true or false.

Logical Operators :

- Logical operators are used to provide logic to our Java programs.
- There are three types of logical operators in Java :
- && - AND
- || - OR
- ! – NOT

AND Operator :

Evaluates to true if both the conditions are true.

- Y && Y = Y
- Y && N = N
- N && Y = N

- N && N = N

Convention: # Y – True and N - False

OR Operator :

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

- Y || Y = Y
- Y || N = Y
- N || Y = Y
- N || N = N

Convention: # Y – True and N - False

NOT Operator :

Negates the given logic (true becomes false and vice-versa)

- !Y = N
- !N = Y

Code as Described in the Video

```
package com.company;

public class cwh_17_logical {
    public static void main(String[] args) {
        System.out.println("For Logical AND...");
        boolean a = true;
        boolean b = false;
//        if (a && b){
//            System.out.println("Y");
//        }
//        else{
//            System.out.println("N");
//        }

        System.out.println("For Logical OR...");

//        if (a || b){
//            System.out.println("Y");
//        }
//        else{
//            System.out.println("N");
//        }

        System.out.println("For Logical NOT");
        System.out.print("Not(a) is ");
        System.out.println(!a);
        System.out.print("Not(b) is ");
        System.out.println(!b);
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Switch Case Statements in Java

Switch Case-Control Instruction

- Switch-Case is used when we have to make a choice between the number of alternatives for a given variable.
- Var can be an integer, character, or string in Java.
- Every switch case must contain a default case. The default case is executed when all the other cases are false.
- Never forget to include the break statement after every switch case otherwise the switch case will not terminate.

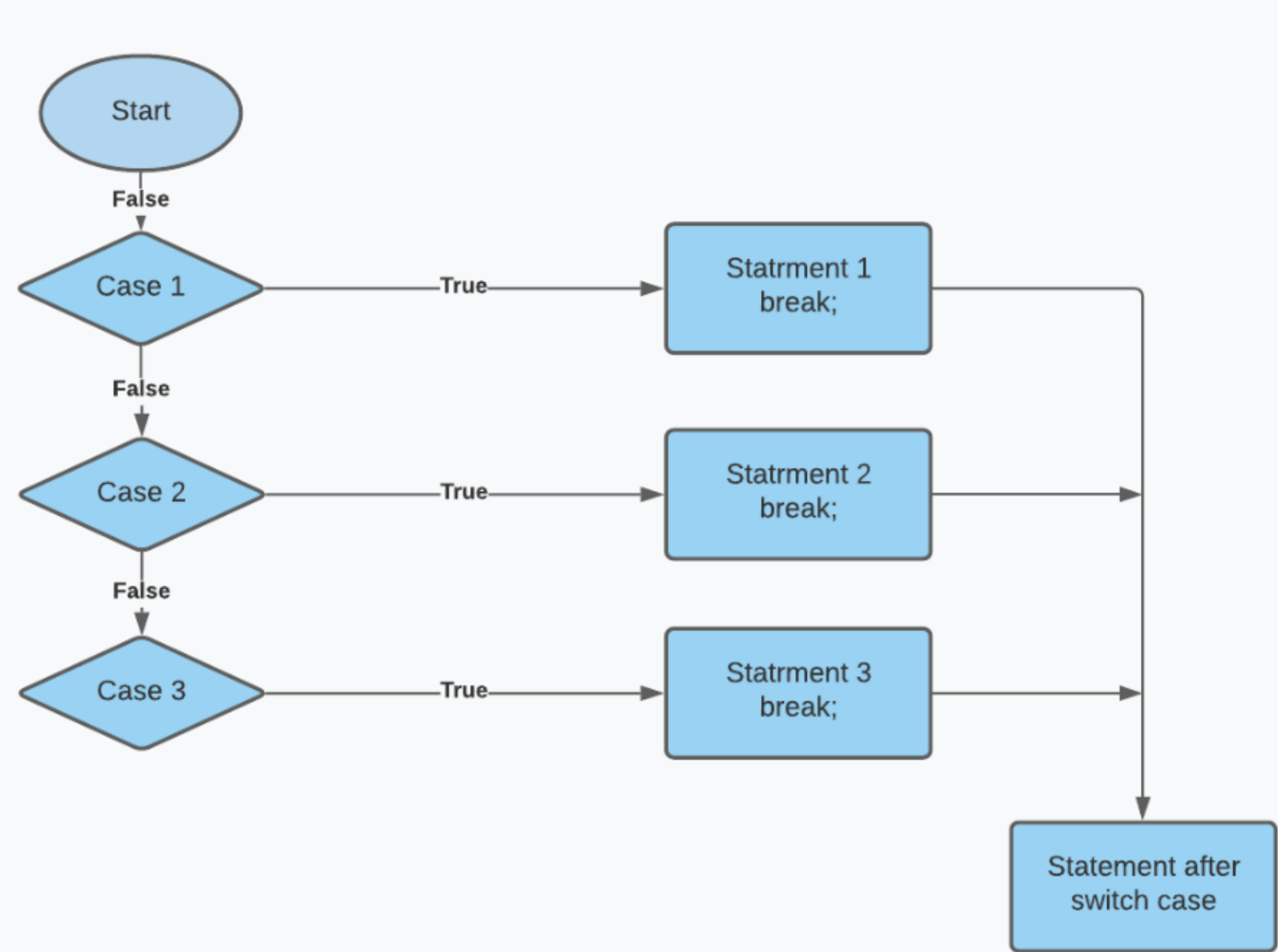
Syntax :

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

Copy

- A switch can occur within another but in practice, this is rarely done.

Flow control of switch case in Java :



Code as Described in the Video

```

package com.company;
import java.util.Scanner;

public class cwh_18_elseif {
    public static void main(String[] args) {
        String var = "Saurabh";

        switch (var) {
            case "Shubham" -> {
                System.out.println("You are going to become an Adult!");
                System.out.println("You are going to become an Adult!");
                System.out.println("You are going to become an Adult!");
            }
            case "Saurabh" -> System.out.println("You are going to join a Job!");
            case "Vishaka" -> System.out.println("You are going to get retired!");
            default -> System.out.println("Enjoy Your life!");
        }
        System.out.println("Thanks for using my Java Code!");

        /*
        int age;
        System.out.println("Enter Your Age");
        Scanner sc = new Scanner(System.in);
        age = sc.nextInt();
        if (age>56){
            System.out.println("You are experienced!");
        }
        else if(age>46){
            System.out.println("You are semi-experienced!");
        }
        else if(age>36){
            System.out.println("You are semi-semi-experienced!");
        }
        else{
            System.out.println("You are not experienced");
        }
        if(age>2){
            System.out.println("You are not a baby!");
        }
        */

    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Practice Questions On Conditionals & Switch Case

Chapter 4– Practice Set

Question 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");
```

Copy

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

Question 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 not tax below 2.5L. Take the input amount as input from the user.

Question 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!]

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

Question 6:Write a program to find out the type of website from the URL:

- .com – commercial website
- .org – organization website
- .in – Indian website

```
package com.company;
import java.util.Scanner;
import java.util.Random;

public class cwh_19_ch4_ps {
    public static void main(String[] args) {
//        Question 1:
//        int a = 11;
//        if(a=11){
//
//        }

        // Question 2
//        byte m1, m2, m3;
//        Scanner sc = new Scanner(System.in);
//        System.out.println("Enter your marks in Physics");
//        m1 = sc.nextByte();
//
//        System.out.println("Enter your marks in Chemistry");
//        m2= sc.nextByte();
//
//        System.out.println("Enter your marks in Mathematics");
//        m3 = sc.nextByte();
//        float avg = (m1+m2+m3)/3.0f;
//        System.out.println("Your Overall percentage is: " + avg);
//        if(avg>=40 && m1>=33 && m2>=33 && m3>=33){
//            System.out.println("Congratulations, You have been promoted");
//        }
//        else{
//            System.out.println("Sorry, You have not been promoted! Please try again.");
```

```

//      }

// Question 3
//      Scanner sc = new Scanner(System.in);
//      System.out.println("Enter your income in Lakhs per annum");
//      float tax = 0;
//      float income = sc.nextFloat();
//      if(income<=2.5){
//          tax = tax + 0;
//      }
//      else if(income>2.5f && income <= 5f){
//          tax = tax + 0.05f * (income - 2.5f);
//      }
//      else if(income>5f && income <= 10.0f){
//          tax = tax + 0.05f * (5.0f - 2.5f);
//          tax = tax + 0.2f * (income - 5f);
//      }
//      else if(income>10.0f){
//          tax = tax + 0.05f * (5.0f - 2.5f);
//          tax = tax + 0.2f * (10.0f - 5f);
//          tax = tax + 0.3f * (income - 10.0f);
//      }
//
//      System.out.println("The total tax paid by the employee is: " + tax);

// Question 4:

//      Scanner sc = new Scanner(System.in);
//      int day = sc.nextInt();
//
//      switch (day){
//          case 1 -> System.out.println("Monday");
//          case 2 -> System.out.println("Tuesday");
//          case 3 -> System.out.println("Wednesday");
//          case 4 -> System.out.println("Thursday");
//          case 5 -> System.out.println("Friday");
//          case 6 -> System.out.println("Saturday");
//          case 7 -> System.out.println("Sunday");
//      }

// Question 6
//      Scanner sc = new Scanner(System.in);
//      String website = sc.next();
//      if(website.endsWith(".org")){
//          System.out.println("This is an organizational website");
//      }
//      else if(website.endsWith(".com")){
//          System.out.println("This is a Commercial website");
//      }
//      else if(website.endsWith(".in")){
//          System.out.println("This is an Indian website");
//      }
//
Random r = new Random();
int a = r.nextInt();

```



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

    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Programming Exercise 2: Rock, Paper Scissors Game in Java

Exercise 2

Create a simple Rock, Paper Scissors game in Java. (#Use Conditional Statements)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: While Loops in Java

- In programming languages, loops are used to execute a particular statement/set of instructions again and again.
- The execution of the loop starts when some conditions become true.
- 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

Let's look into these, one by one.

While loops :

- The while loop in Java is used when we need to execute a block of code again and again based on a given boolean condition.
- Use a while loop if the exact number of iterations is not known.
- If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

```
/*
while (Boolean condition)

{

    // Statements    -> This keeps executing as long as the condition is true.

}
*/
```

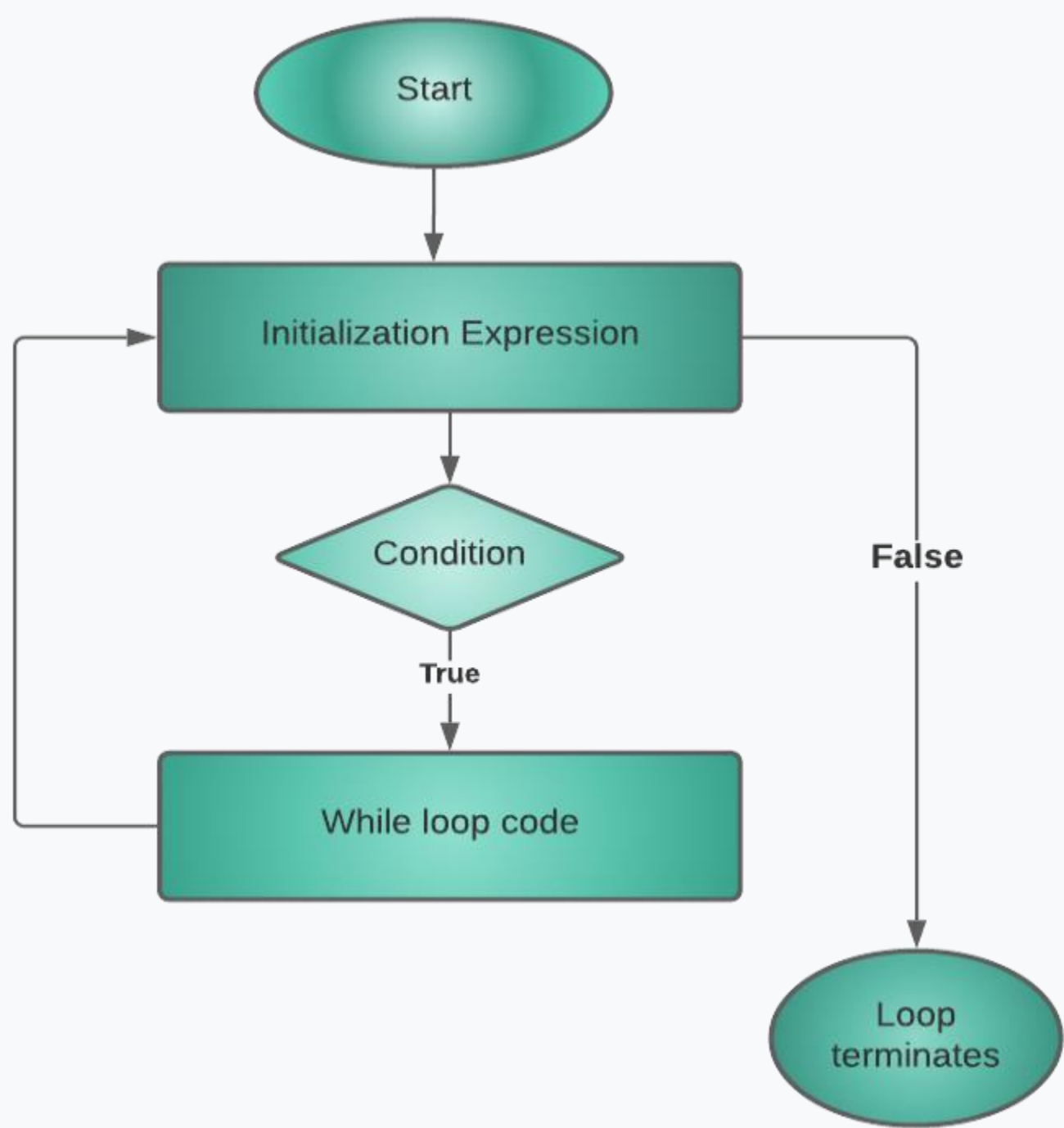
Copy

Example :

```
int i=10;
while(i>0){
System.out.println(i);
i--;
}
```

Copy

Flow control of while loop :



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

Code as described in the video :

```
package com.company;

public class cwh_21_ch5_loops {
    public static void main(String[] args) {
        System.out.println(1);
        System.out.println(2);
        System.out.println(3);

        System.out.println("Using Loops:");
        int i = 100;
        while(i<=200){
            System.out.println(i);
            i++;
        }
        System.out.println("Finish Running While Loop!");

        // while(true){
        //     System.out.println("I am an infinite while loop!");
        // }

    }
}
```

Copy

This is all for this tutorial, and we will discuss the do-while loop in the next tutorial.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: The do-while loop in Java

Do-while loop:

- Do- while loop is similar to a while loop except for the fact that it is guaranteed to execute at least once.
- Use a do-while loop when the exact number of iterations is unknown, but you need to execute a code block at least once.
- After executing a part of a program for once, the rest of the code gets executed on the basis of a given boolean condition.

Syntax :

```
/* do {  
  
    //code  
  
} while (condition);           //Note this semicolon */
```

Copy

Example :

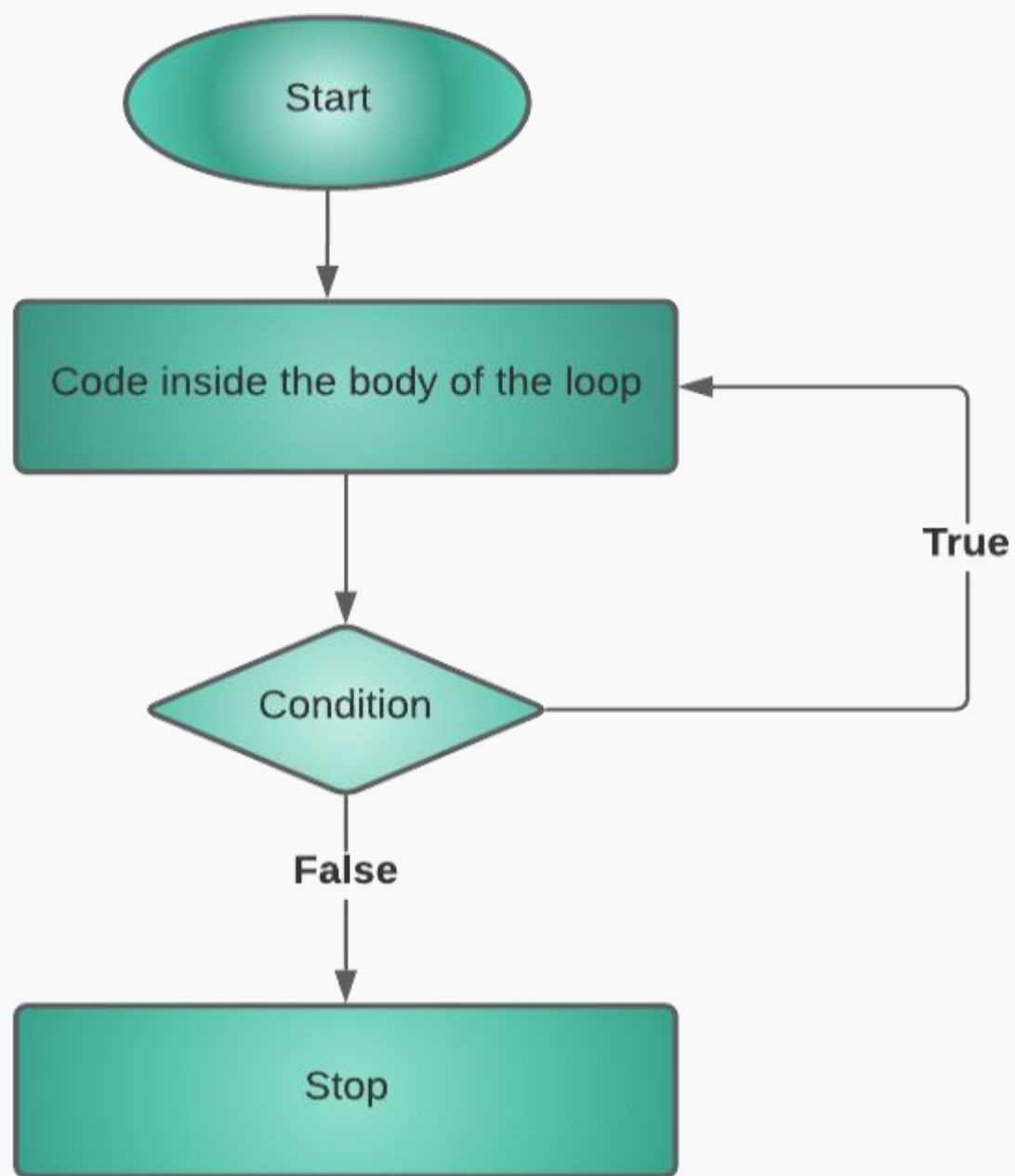
```
int i=1;  
do{  
System.out.println(i);  
i++;  
}while(i<=10);
```

Copy

Difference Between while loop and do-while loop :

- while – checks the condition & executes the code.
- do-while – executes the code at least once and then checks the condition. Because of this reason, the code in the do-while loop executes at least once, even if the condition fails.

Flow control of do-while loop :



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

Code as described in the video :

```
package com.company;

public class cwh_22_ch4_do_while {
    public static void main(String[] args) {
//        int a = 0;
//        while(a<5){
//            System.out.println(a);
//            a++;
//        }
        int b = 10;
        do {
            System.out.println(b);
            b++;
        }while(b<5);

        int c = 1;
        do{
            System.out.println(c);
            c++;
        }while(c<=45);

    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: The for Loop in Java

For loop:

- For loop in java is used to iterate a block of code multiple times.
- Use for loop only when the exact number of iterations needed is already known to you.

Syntax :

```
/* for (initialize; check_bool_expression; update){  
  
    //code;  
  
} */
```

Copy

- **Initializer:** Initializes the value of a variable. This part is executed only once.
- **check_bool_expression:** The code inside the for loop is executed only when this condition returns true.
- **update:** Updates the value of the initial variable.

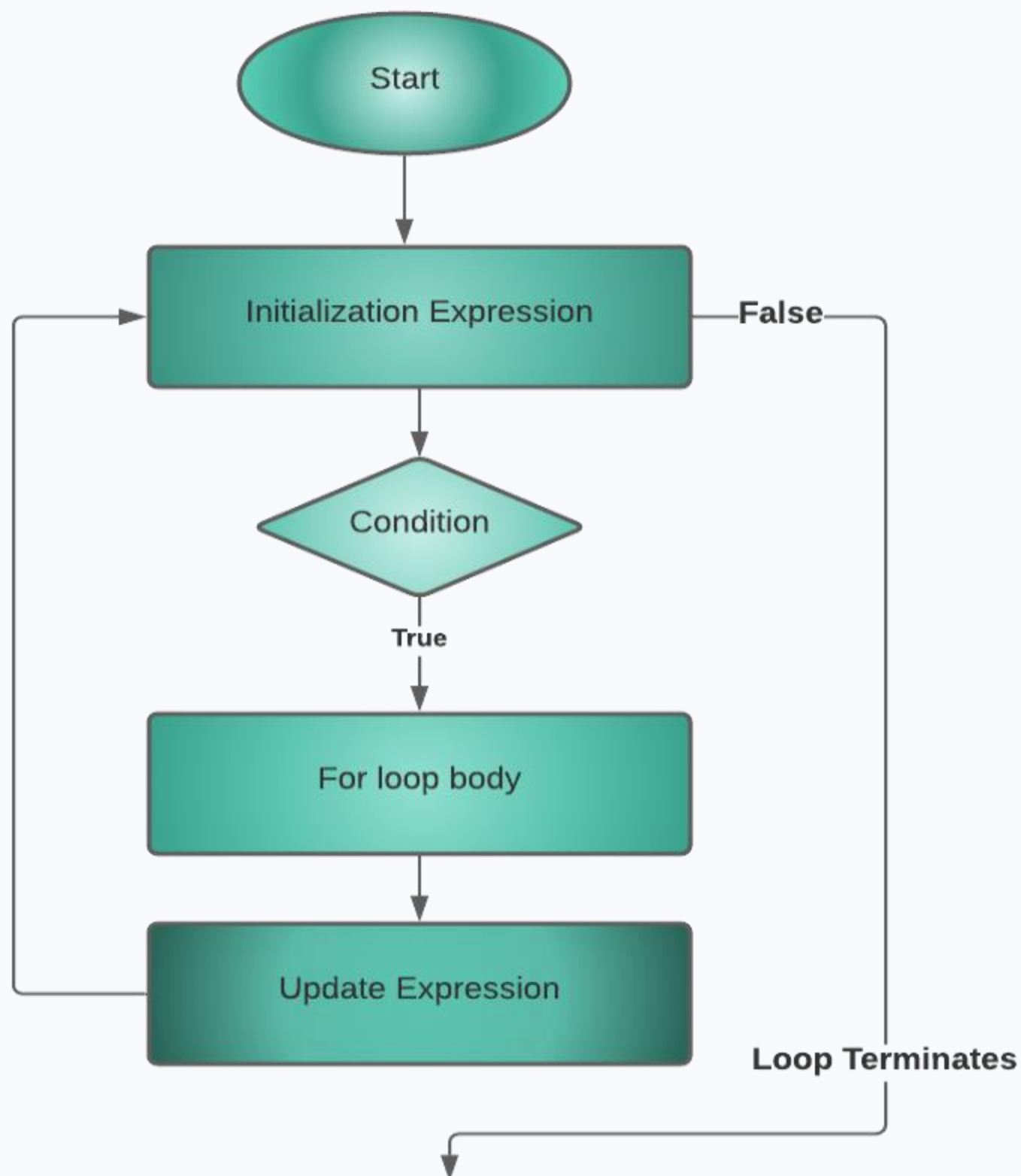
Example :

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

Copy

The above for loop initializes the value of i=7 and keeps printing as well as decrementing the value of i till i do not get equals to 0.

Flow control of for loop :



Quick Quiz 1: Write a program to print first n odd numbers using a for loop.

Quick Quiz 2: Write a program to print first n natural numbers in reverse order.

Code as described in the video :

```
package com.company;

public class cwh_23_for_loop {
    public static void main(String[] args) {
//        for (int i=1; i<=10; i++){
//            System.out.println(i);
//        }
//        // 2i = Even Numbers = 0, 2, 4, 6, 8
//        // 2i+1 = Odd Numbers = 1, 3, 5, 7, 9
//        //int n = 3;
//        //for (int i =0; i<n; i++){
//            //    System.out.println(2*i+1);
//        }

        for(int i=5; i!=0; i--){
            System.out.println(i);
        }
    }
}
```

```
}  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: break and continue in Java

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.

Syntax :

```
break;
```

Copy

Example to demonstrate the use of break inside a for loop :

```
public class CWH_break {  
public static void main(String[] args) {  
    //using for loop  
    for(int i=10;i>0;i--){  
        if(i==7){  
            break;    //break the loop  
        }  
        System.out.println(i);  
    }  
}  
}
```

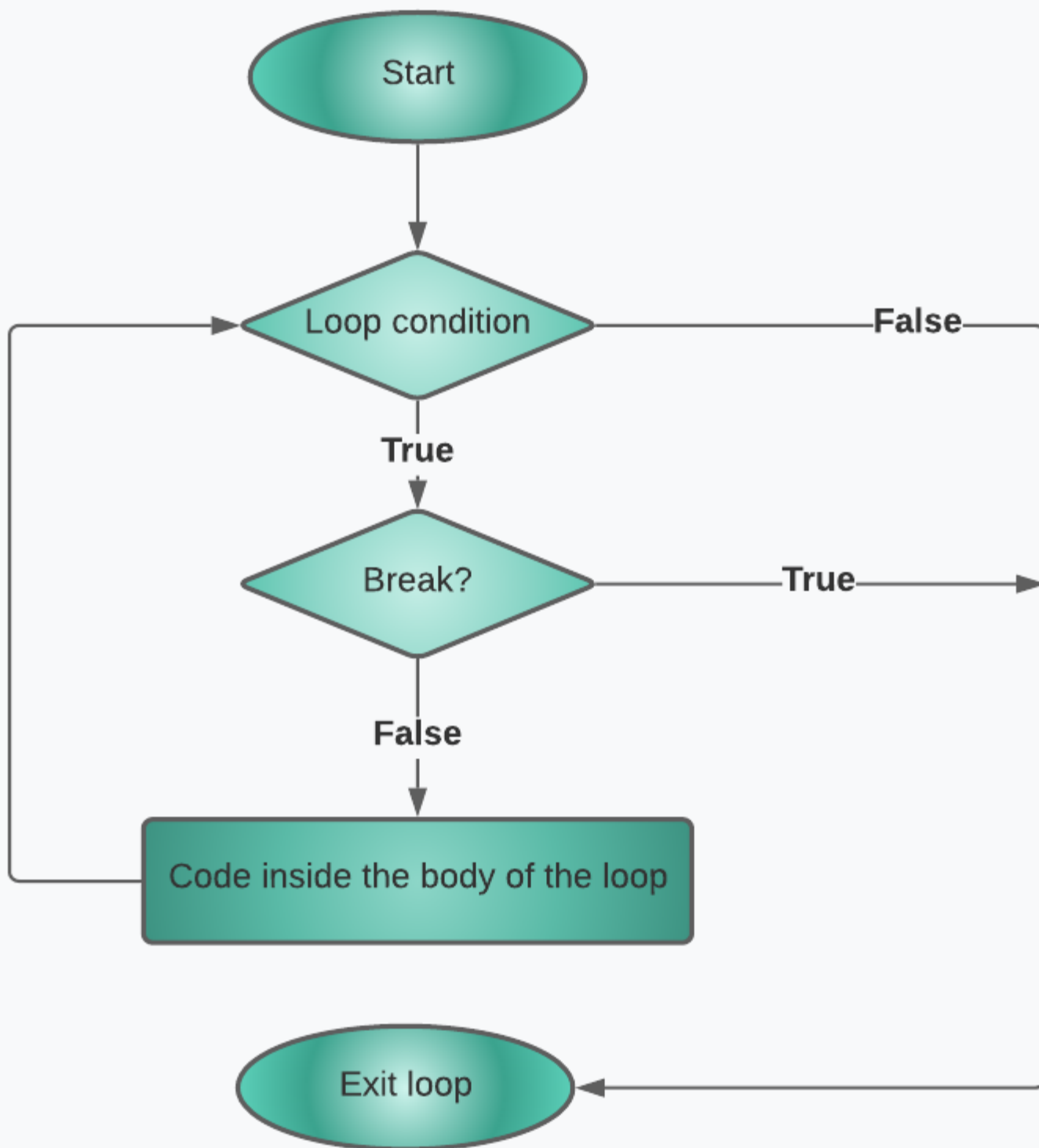
Copy

Output :

```
10  
9  
8
```

Copy

Flow control of break statement :



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.

Syntax :

```
continue;
```

Copy

Example to demonstrate the use of continue statement inside a for loop :

```
public class CWH_continue {
public static void main(String[] args) {

    for(int i=7;i>0;i--){
        if(i==3){
            continue;//continue skips the rest statement
        }
        System.out.println(i);
    }
}
```

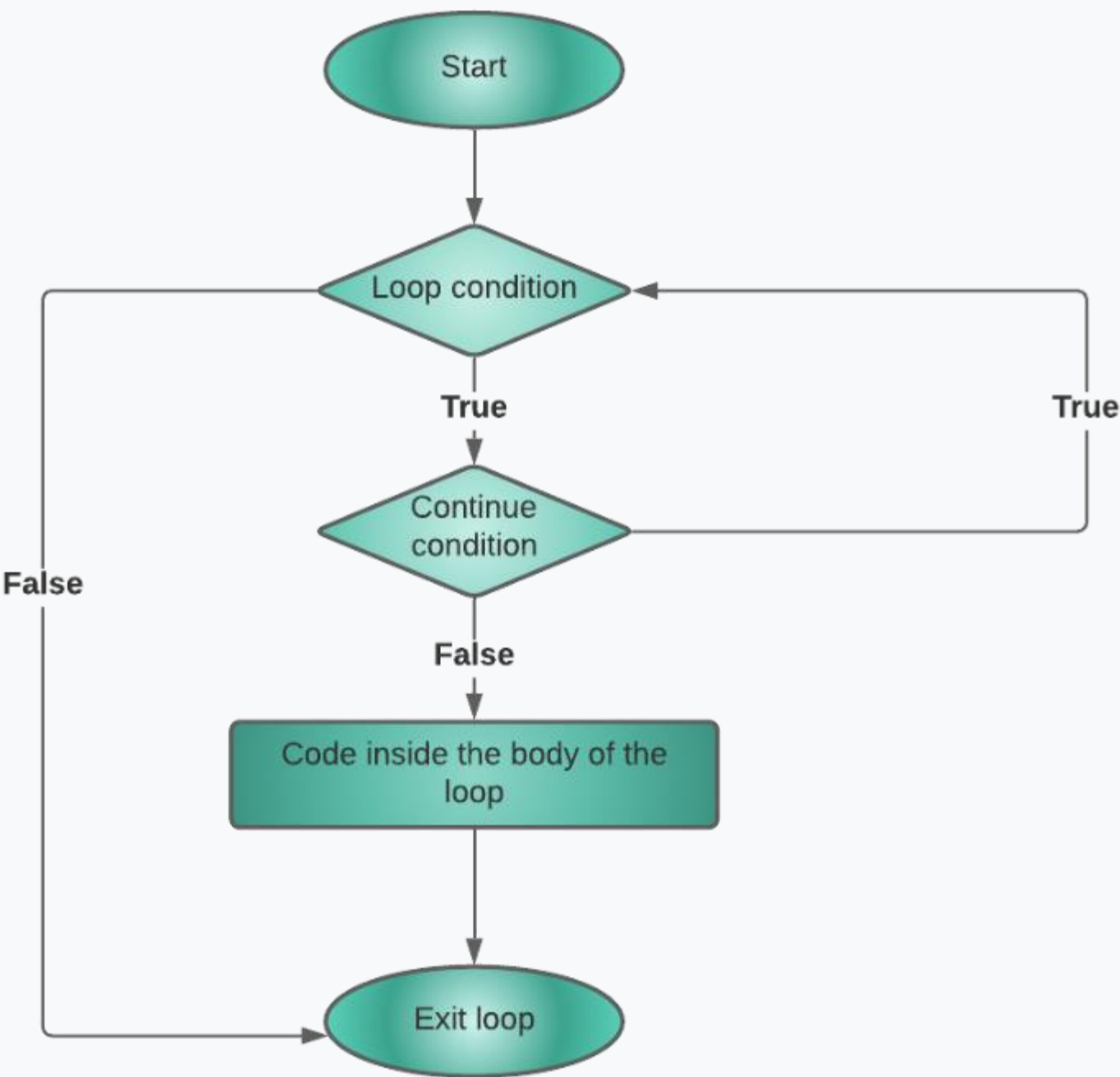
Copy

Output :

7
6
5
4
2
1

Copy

Flow control of continue statement :



In a Nut Shell ...

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

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Code as Described in the Video

```
package com.company;

public class cwh_24_break_and_continue {
    public static void main(String[] args) {
        // Break and continue using loops!
        // for (int i=0;i<50;i++){
        //     System.out.println(i);
```

```
//      System.out.println("Java is great");
//      if(i==2){
//          System.out.println("Ending the loop");
//          break;
//      }
//  }
//  int i=0;
//  do{
//      System.out.println(i);
//      System.out.println("Java is great");
//      if(i==2){
//          System.out.println("Ending the loop");
//          break;
//      }
//      i++;
//  }while(i<5);
//  System.out.println("Loop ends here");

//      for(int i=0;i<50;i++){
//          if(i==2){
//              System.out.println("Ending the loop");
//              continue;
//          }
//          System.out.println(i);
//          System.out.println("Java is great");
//      }
int i=0;
do{
    i++;
    if(i==2){
        System.out.println("Ending the loop");
        continue;
    }
    System.out.println(i);
    System.out.println("Java is great");

}while(i<5);
System.out.println("Loop ends here");
}
}
```

ava tutorial: Practice Questions on Loops

Question 1: Write a program to print the following pattern :

```
****

***

**

*
```

Copy

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

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

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

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

Question 6: Repeat problem 5 using a while loop.

Question 7: Repeat problem 1 using for/while loop.

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

Question 9: Write a program to calculate the sum of the numbers occurring in the multiplication table of 8.

Question 10 :A do-while loop is executed:

- At least once
- At least twice
- At most once

Question 11: Repeat problem 2 using for loop.

Code as Described in the Video

```
package com.company;

public class cwh_25_practice_set_5 {
    public static void main(String[] args) {
        // Practice Problem 1
        //      int n = 4;
        //      for (int i=n; i>0; i--){
        //          for(int j=0;j<i;j++){
        //              System.out.print("*");
        //          }
        //          System.out.print("\n");
        //      }

        // Practice Problem 2
        //      int sum=0;
        //      int n=4;
        //      for(int i=0;i<n;i++){
        //          sum = sum + (2*i);
        //      }
        //      System.out.print("Sum of even numbers is: ");
        //      System.out.println(sum);
        //      // First 4 even numbers are - 0 2 4 6

        // Practice Problem 3
        //      int n = 5;
        //      //for(int i=0; i<10; i++) - Goes from i=0 to i=9
        //      for(int i=1;i<=10;i++){
        //          System.out.printf("%d X %d = %d\n", n, i, n*i);
        //      }

        // Practice Problem 4
        //      int n = 10;
        //      //for(int i=0; i<10; i++) - Goes from i=0 to i=9
```

```
//      for(int i=10;i>=1;i--){
//          System.out.printf("%d X %d = %d\n", n, i, n*i);
//      }

// Practice Problem 6
//      int n = 5;
//      // What is factorial n = n * n-1 * n-2 ..... 1
//      // 5! = 5*4*3*2*1 = 120
//      int i = 1;
//      int factorial = 1;
//      while(i<=n){
//          factorial *= i;
//          i++;
//      }
//      System.out.println(factorial);

// Practice Problem 9
//      int n = 8;
//      int sum = 0;
//      //for(int i=0; i<10; i++) - Goes from i=0 to i=9
//      for(int i=1;i<=10;i++){
//          sum += n*i;
//      }
//      System.out.println(sum);

    }
}
```

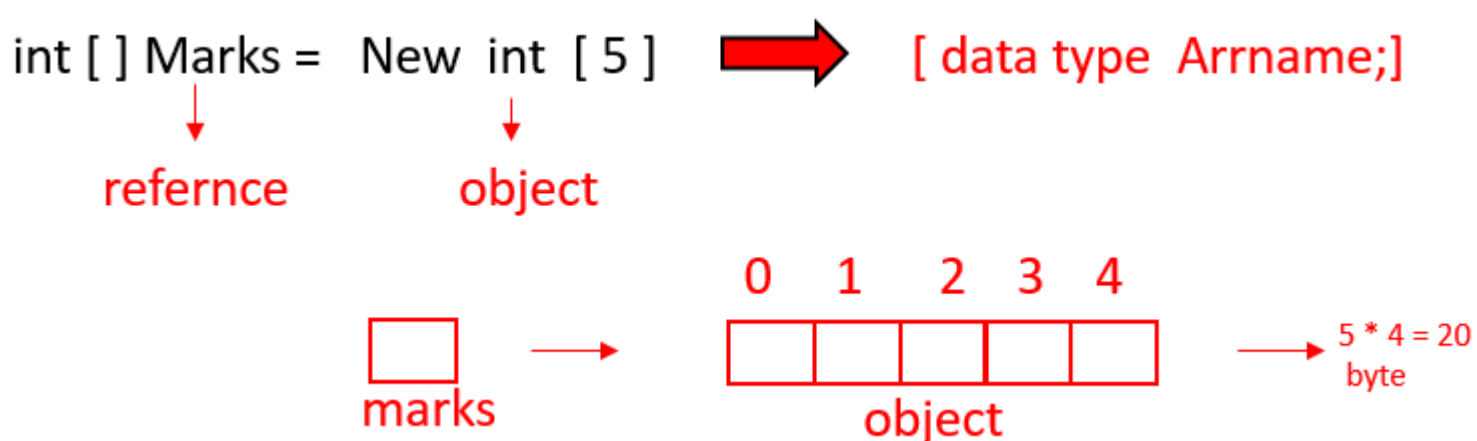
Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Introduction to Arrays

- An array is a collection of similar types of data having contiguous memory allocation.
- The indexing of the array starts from 0., i.e 1st element will be stored at the 0th index, 2nd element at 1st index, 3rd at 2nd index, and so on.
- The size of the array can not be increased at run time therefore we can store only a fixed size of elements in array.
- Use Case: Storing marks of 5 students



Accessing Array Elements :

Array elements can be accessed as follows,

```
/* marks[0] = 100      //Note that index starts from 0
marks[1] = 70
.
```

```
.*
marks[4] = 98 */
```

Copy

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

1. `int[] marks; //Declaration!`
2. `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!`

Note : Array indices start from 0 and go till (n-1) where n is the size of the array.

Array length :

Unlike C/C++, we don't need to use the `sizeof()` operator to get the length of arrays in Java because arrays are objects in Java therefore we can use the `length` property.

```
marks.length //Gives 5 if marks is a reference to an array with 5 elements
```

Copy

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
}
```

Copy

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

Code as Described in the Video

```
package com.company;

public class cwh_26_arrays {
    public static void main(String[] args) {
        /* Classroom of 500 students - You have to store marks of these 500 students
        You have 2 options:
        1. Create 500 variables
        2. Use Arrays (recommended)
        */
        // There are three main ways to create an array in Java
        // 1. Declaration and memory allocation
        // int [] marks = new int[5];

        // 2. Declaration and then memory allocation
        // int [] marks;
        // marks = new int[5];
        // Initialization
        // marks[0] = 100;
        // marks[1] = 60;
        // marks[2] = 70;
        // marks[3] = 90;
        // marks[4] = 86;

        // 3. Declaration, memory allocation and initialization together
        int [] marks = {98, 45, 79, 99, 80};

        // marks[5] = 96; - throws an error
```

```
        System.out.println(marks[4]);
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: For Each Loop in Java

- For each loop is an enhanced version of for loop.
- It travels each element of the data structure one by one.
- Note that you can not skip any element in for loop and it is also not possible to traverse elements in reverse order with the help of for each loop.
- It increases the readability of the code.
- If you just want to simply traverse an array from start to end then it is recommended to use for each loop.

Syntax :

```
/* for (int element:Arr) {
        Sout(element);    //Prints all the elements
    } */
```

Copy

Example :

```
class CWH_forEachLoop{
    public static void main(String args[]){
        //declaring an array
        int arr[]={1,2,3,3,4,5};
        //traversing the array with for-each loop
        for(int i:arr){
            System.out.println(i);
        }
    }
}
```

Copy

Output :

```
1
2
3
4
5
```

Copy

Code as Described in the Video

```
package com.company;

public class cwh_27_arrays {
    public static void main(String[] args) {

        /*
        float [] marks = {98.5f, 45.5f, 79.5f, 99.5f, 80.5f};
        String [] students ={"Harry", "Rohan", "Shubham", "Lovish"};
        System.out.println(students.length);
        System.out.println(students[2]);
        */
    }
}
```

```

    */

    int [] marks = {98, 45, 79, 99, 80};
    // System.out.println(marks.length);

    // Displaying the Array (Naive way)
    System.out.println("Printing using Naive way");
    System.out.println(marks[0]);
    System.out.println(marks[1]);
    System.out.println(marks[2]);
    System.out.println(marks[3]);
    System.out.println(marks[4]);

    // Displaying the Array (for loop)
    System.out.println("Printing using for loop");
    for(int i=0;i<marks.length;i++){
        System.out.println(marks[i]);
    }

    // Quick Quiz: Displaying the Array in Reverse order (for loop)
    System.out.println("Printing using for loop in reverse order");
    for(int i=marks.length -1;i>=0;i--){
        System.out.println(marks[i]);
    }

    // Quick Quiz: Displaying the Array (for-each loop)
    System.out.println("Printing using for-each loop");
    for(int element: marks){
        System.out.println(element);
    }

}
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Multidimensional Arrays in Java

Multidimensional Arrays are an Array of Arrays. Each elements of an 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
```

Copy

We can add elements to this array as follows

```
flats[0][0] = 100
flats[0][1] = 101
flats[0][2] = 102
// ... & so on!
```

Copy

This 2-D array can be visualized as follows:

	[0]	[1]	[2]
	Col 1	Col 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]
```

Copy

```
package com.company;

public class cwh_28_multi_dim_arrays {
    public static void main(String[] args) {
        int [] marks; // A 1-D Array
        int [][] flats; // A 2-D Array
        flats = new int [2][3];
        flats[0][0] = 101;
        flats[0][1] = 102;
        flats[0][2] = 103;
        flats[1][0] = 201;
        flats[1][1] = 202;
        flats[1][2] = 203;

        // Displaying the 2-D Array (for loop)
        System.out.println("Printing a 2-D array using for loop");
        for(int i=0;i<flats.length;i++){
            for(int j=0;j<flats[i].length;j++) {
                System.out.print(flats[i][j]);
                System.out.print(" ");
            }
            System.out.println("");
        }
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Practice Questions on Arrays in Java

- 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 a for-each loop.
- 4. Create a Java program to add two matrices of size 2x3.
- 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 maximum element in a Java array.
- 8. Write a Java program to find whether an array is sorted or not.

Code Solution:

```
package com.company;
```



```
public class cwh_29_Practice_Set_6 {
    public static void main(String[] args) {
        // Practice Problem 1
        /* float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};
        float sum = 0;
        for(float element:marks){
            sum = sum + element;
        }
        System.out.println("The value of sum is " + sum);

        // Practice Problem 2
        float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};
        float num = 45.57f;
        boolean isInArray = false;
        for(float element:marks){
            if(num==element){
                isInArray = true;
                break;
            }
        }
        if(isInArray){
            System.out.println("The value is present in the array");
        }
        else{
            System.out.println("The value is not present in the array");
        }

        // Practice Problem 3

        float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};
        float sum = 0;
        for(float element:marks){
            sum = sum + element;
        }
        System.out.println("The value of average marks is " + sum/marks.length);

        // Practice Problem 4
        int [][] mat1 = {{1, 2, 3},
                        {4, 5, 6}};
        int [][] mat2 = {{2, 6, 13},
                        {3, 7, 1}};
        int [][] result = {{0, 0, 0},
                           {0, 0, 0}};

        for (int i=0;i<mat1.length;i++){ // row number of times
            for (int j=0;j<mat1[i].length;j++) { // column number of time
                System.out.format(" Setting value for i=%d and j=%d\n", i, j);
                result[i][j] = mat1[i][j] + mat2[i][j];
            }
        }
    }
}
```

```

// Printing the elements of a 2-D Array
for (int i=0;i<mat1.length;i++){ // row number of times
    for (int j=0;j<mat1[i].length;j++) { // column number of time
        System.out.print(result[i][j] + " ");
        result[i][j] = mat1[i][j] + mat2[i][j];
    }
    System.out.println(""); // Prints a new line
}

// Practice Problem 5
int [] arr = {1, 21, 3, 4, 5, 34, 67};
int l = arr.length;
int n = Math.floorDiv(l, 2);
int temp;

for(int i=0; i<n; i++){
    // Swap a[i] and a[l-1-i]
    // a    b    temp
    // |4|  |3|  ||
    temp = arr[i];
    arr[i] = arr[l-i-1];
    arr[l-i-1] = temp;
}

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

// Practice Problem 6
int [] arr = {1, 2100, 3, 455, 5, 34, 67};
int max = Integer.MIN_VALUE;
for(int e: arr){
    if(e>max){
        max = e;
    }
}
System.out.println("the value of the maximum element in this array is: "+ max);

// Practice Problem 6
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE);
*/

// Practice Problem 7
boolean isSorted = true;
int [] arr = {1, 12, 3, 4, 5, 34, 67};
for(int i=0;i<arr.length-1;i++){
    if(arr[i] > arr[i+1]){
        isSorted = false;
        break;
    }
}
if(isSorted){
    System.out.println("The Array is sorted");
}

```

```
    }
    else{
        System.out.println("The Array is not sorted");
    }

}

}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

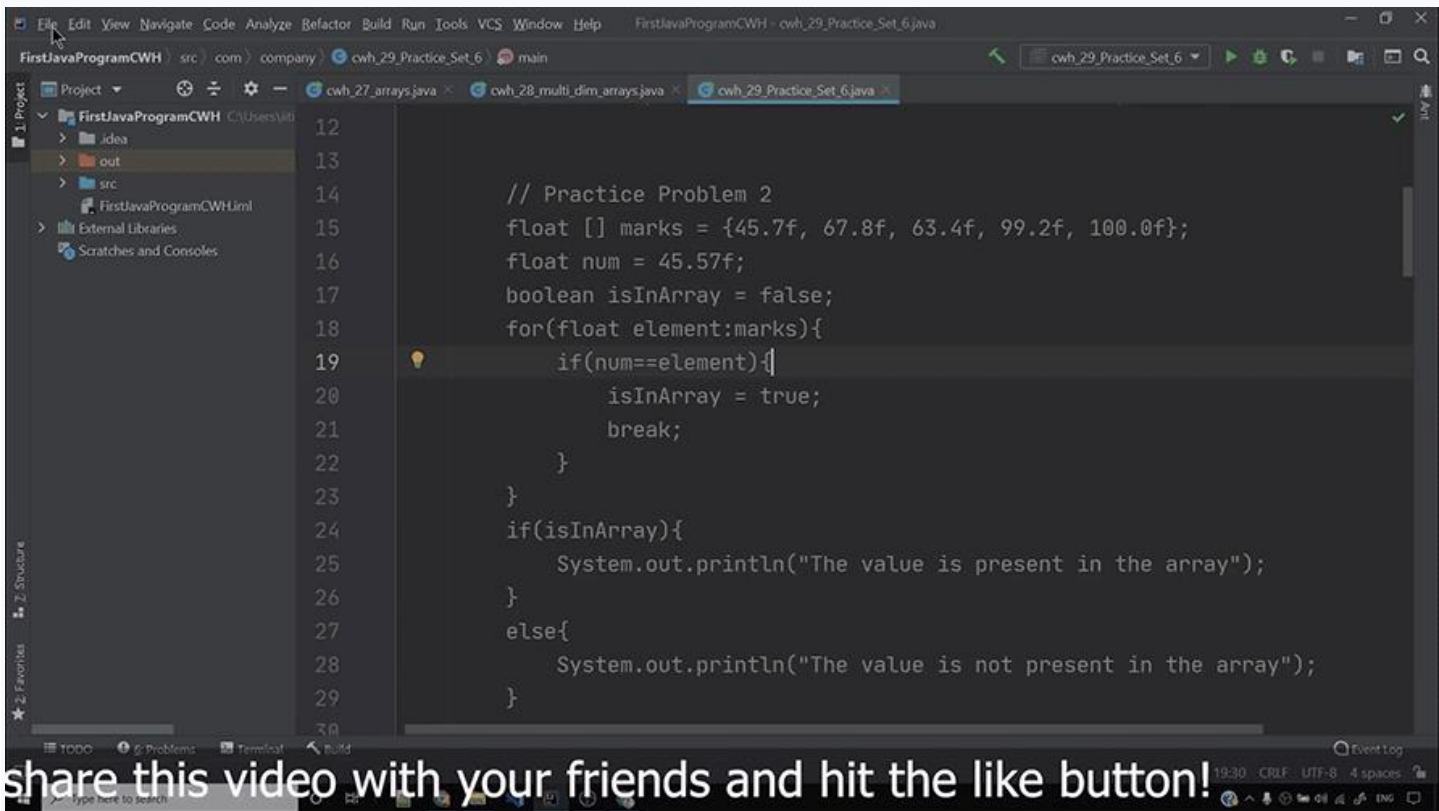
How to Make IntelliJ IDEA look Amazing!

We all know that programming is a fascinating thing. When it comes to writing code, most coders prefer to customize their IDE's on their own to feel motivated, and it brings positive vibes from inside.

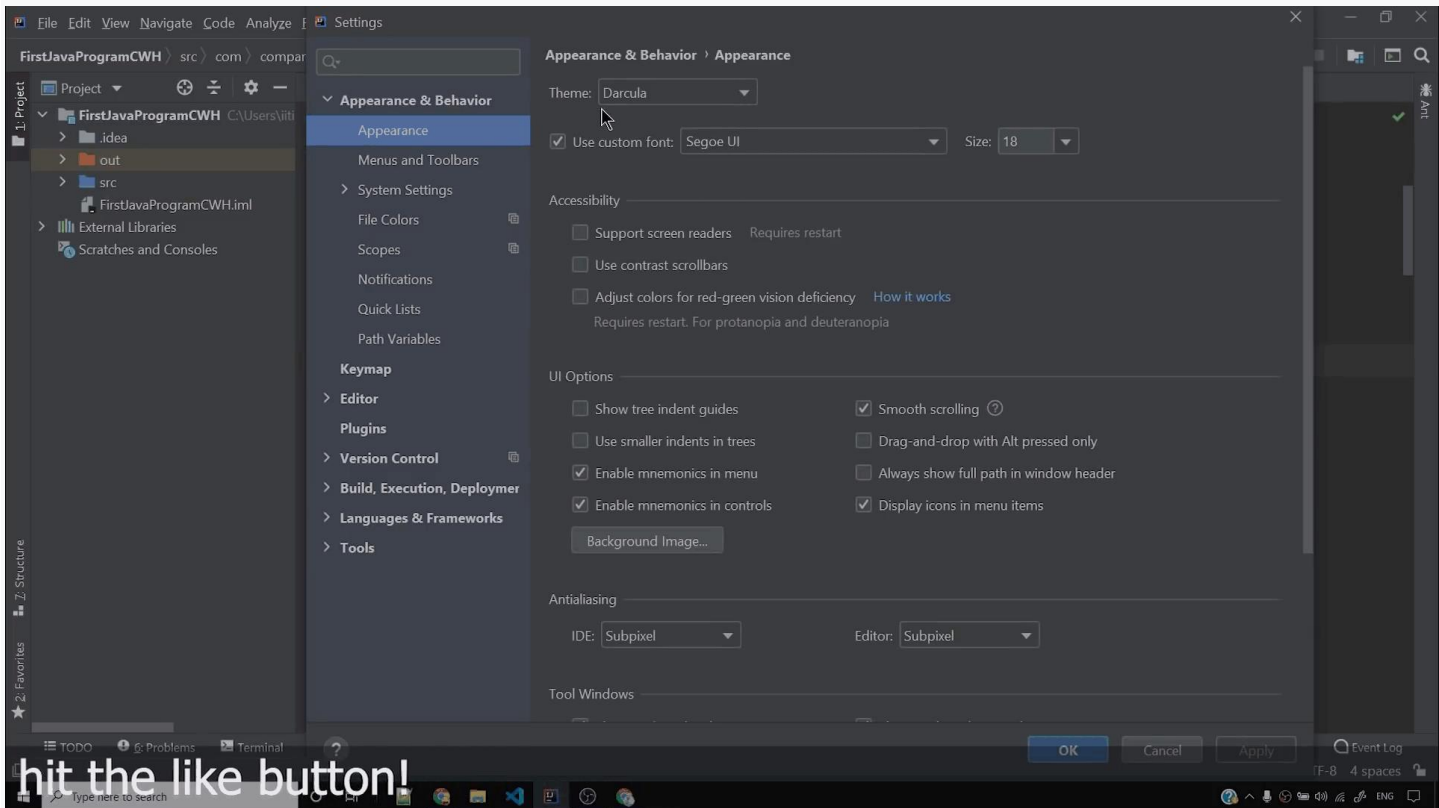
So, in this tutorial, I'm going to help you guys to customize IntelliJ IDEA IDE.

Follow the steps given below to customize your IDE:

Step 1: Go to the File option in the menu and then choose the settings option.

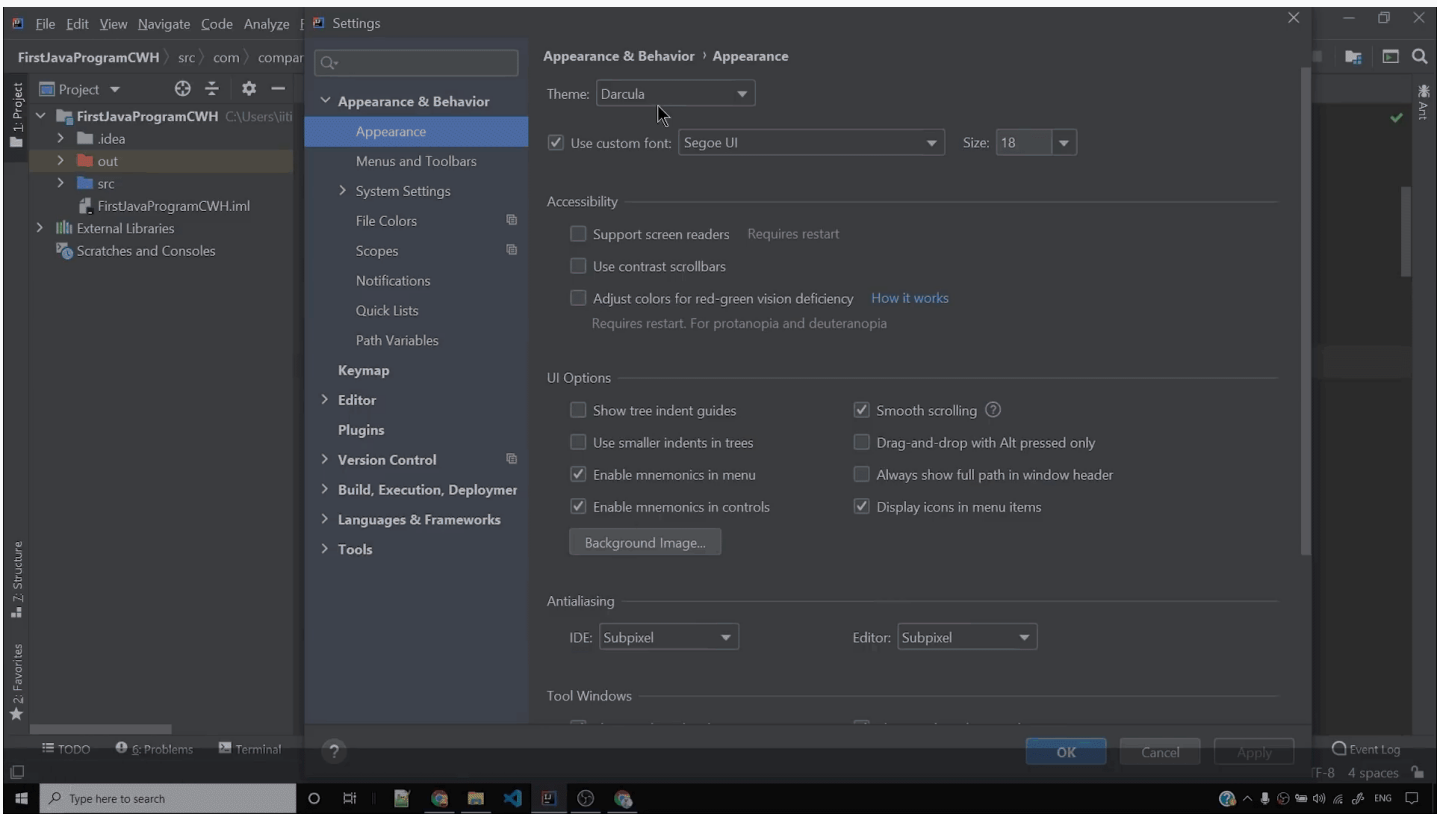


In settings, you'll find many options to customize your IntelliJ IDEA.



To Change Theme of IDE:

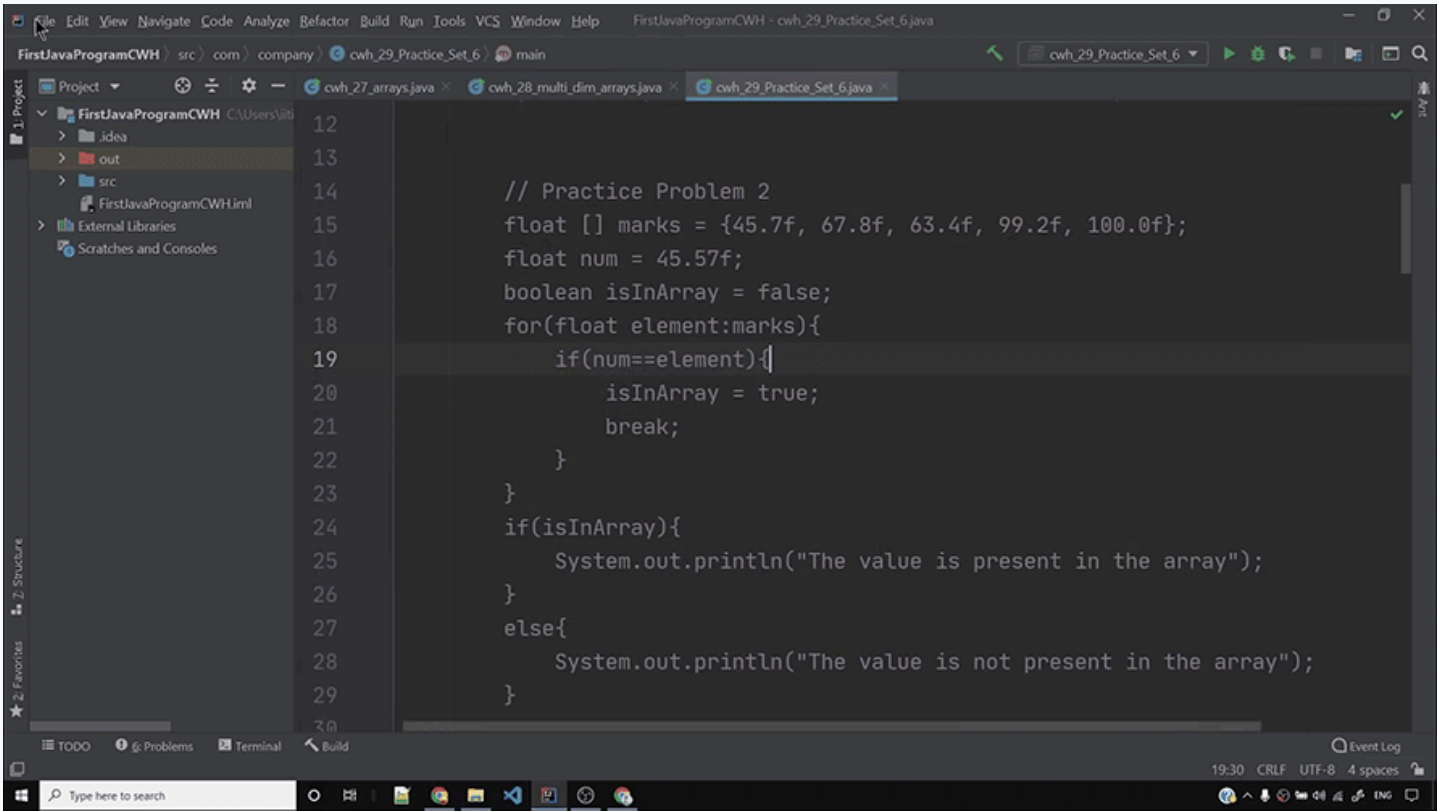
Choose the theme option in the appearance part of the settings. You can select any one of the four themes and can apply it.



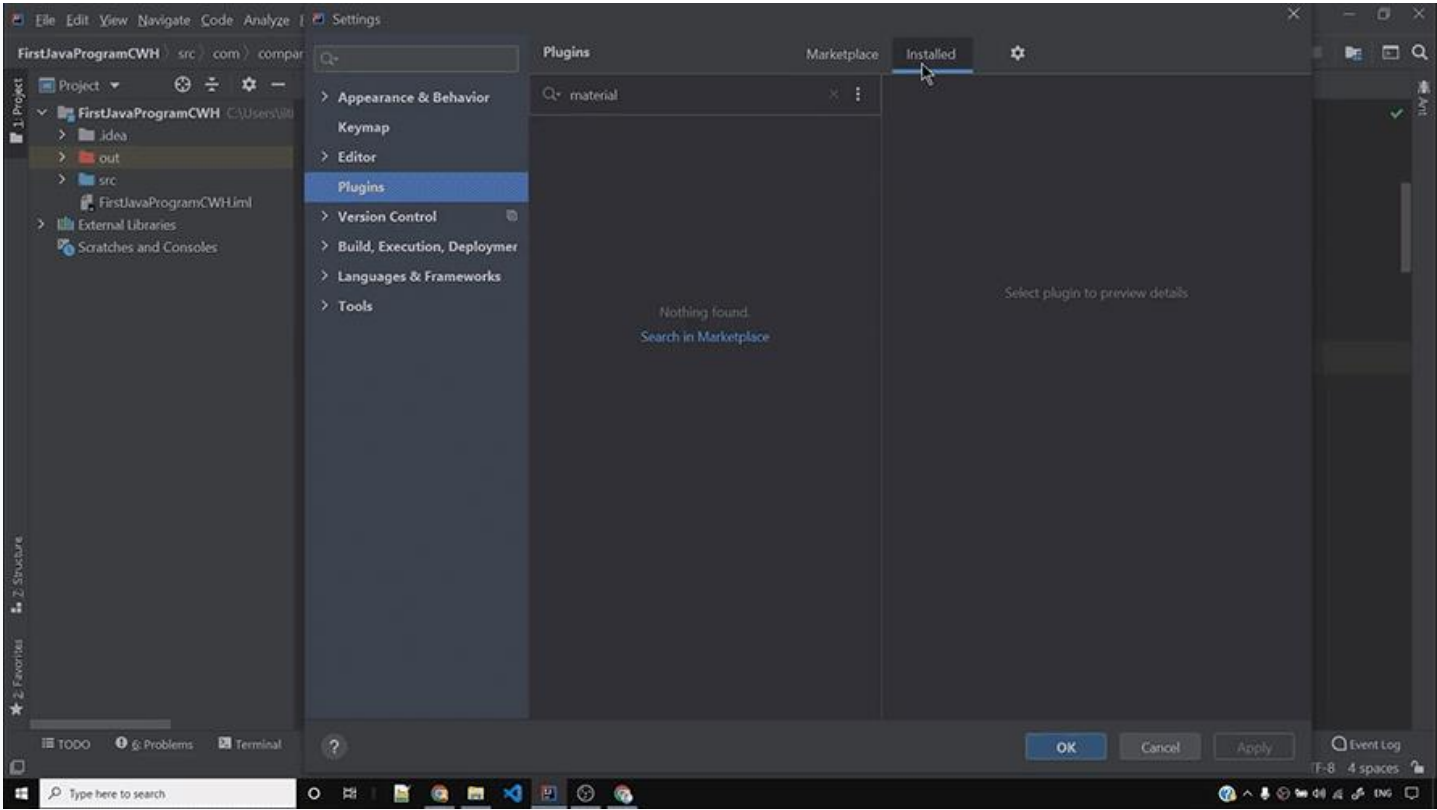
There are many ways to customize the IDE. But for this tutorial purpose, I'll tell you the most efficient and best way to make the IDE look professional and premium by using the Material Theme UI plugin.

We will use the Material Theme UI plugin, and the steps to use the material UI are given below:

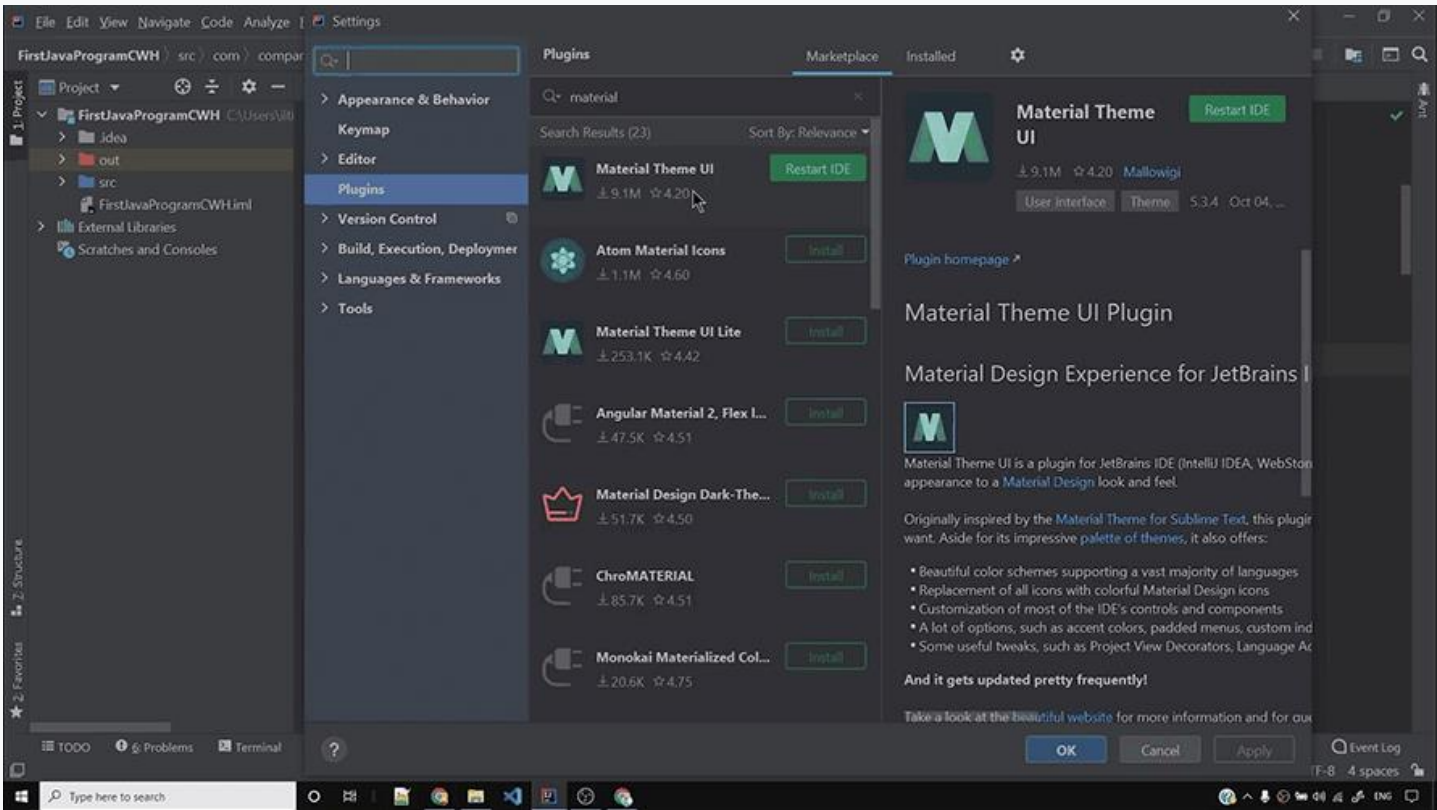
Step 1: Go to File>Settings and then choose the plugins option.



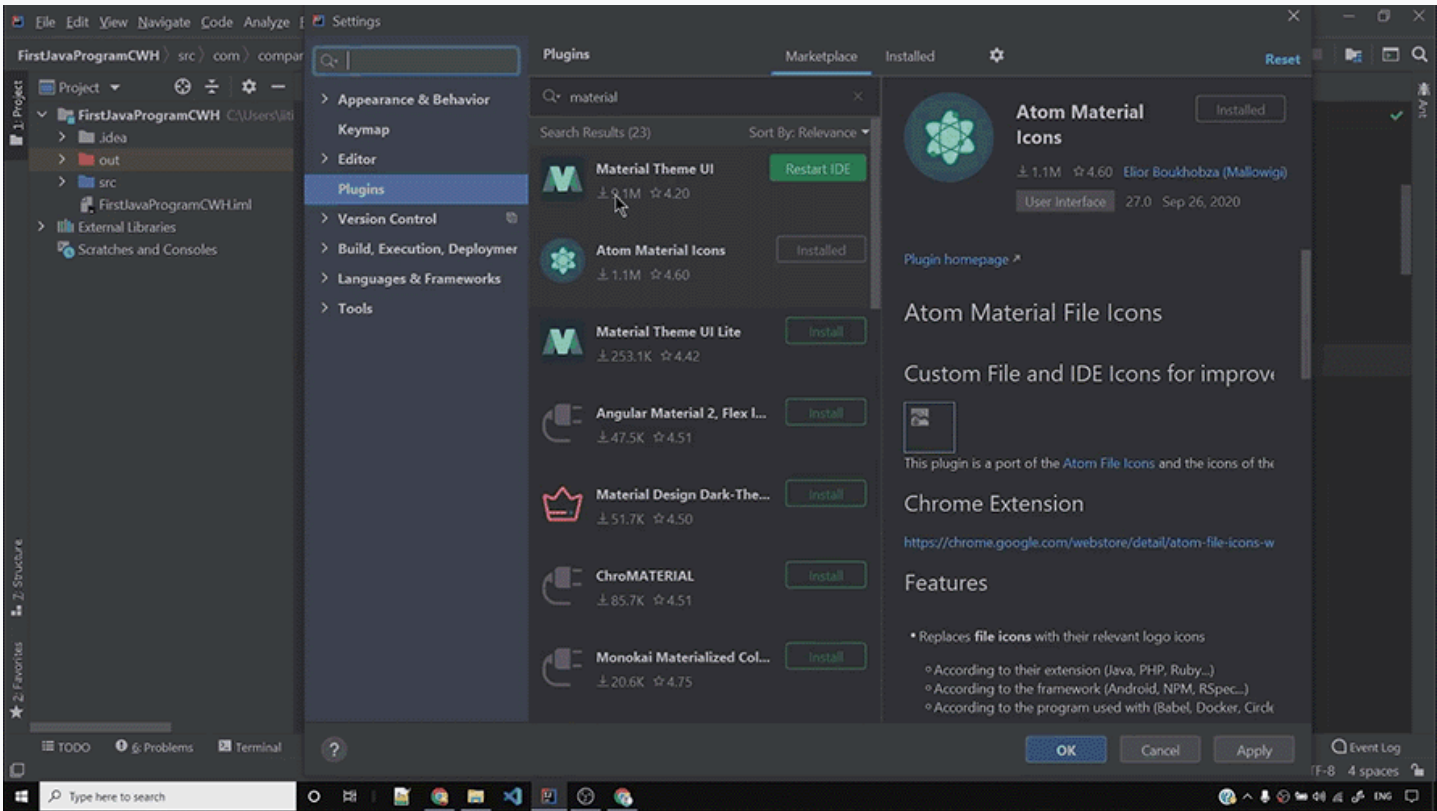
Step 2: Go to the Marketplace in plugins and search for "material" and the first result will be "Material Theme UI". Click on the install option to install it.



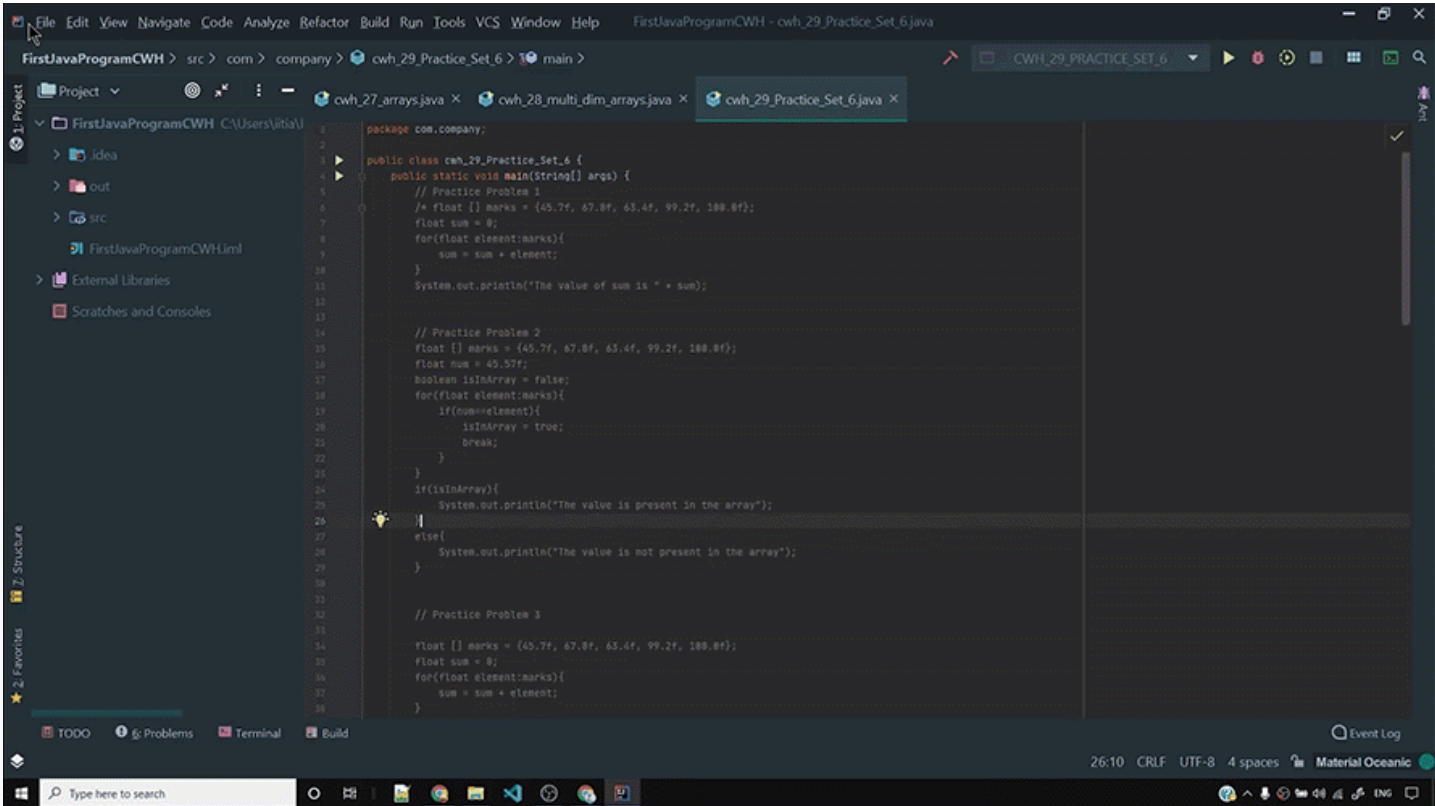
Step 3: Install "Atom Material Icons" to make the IDE icons look attractive and cool.



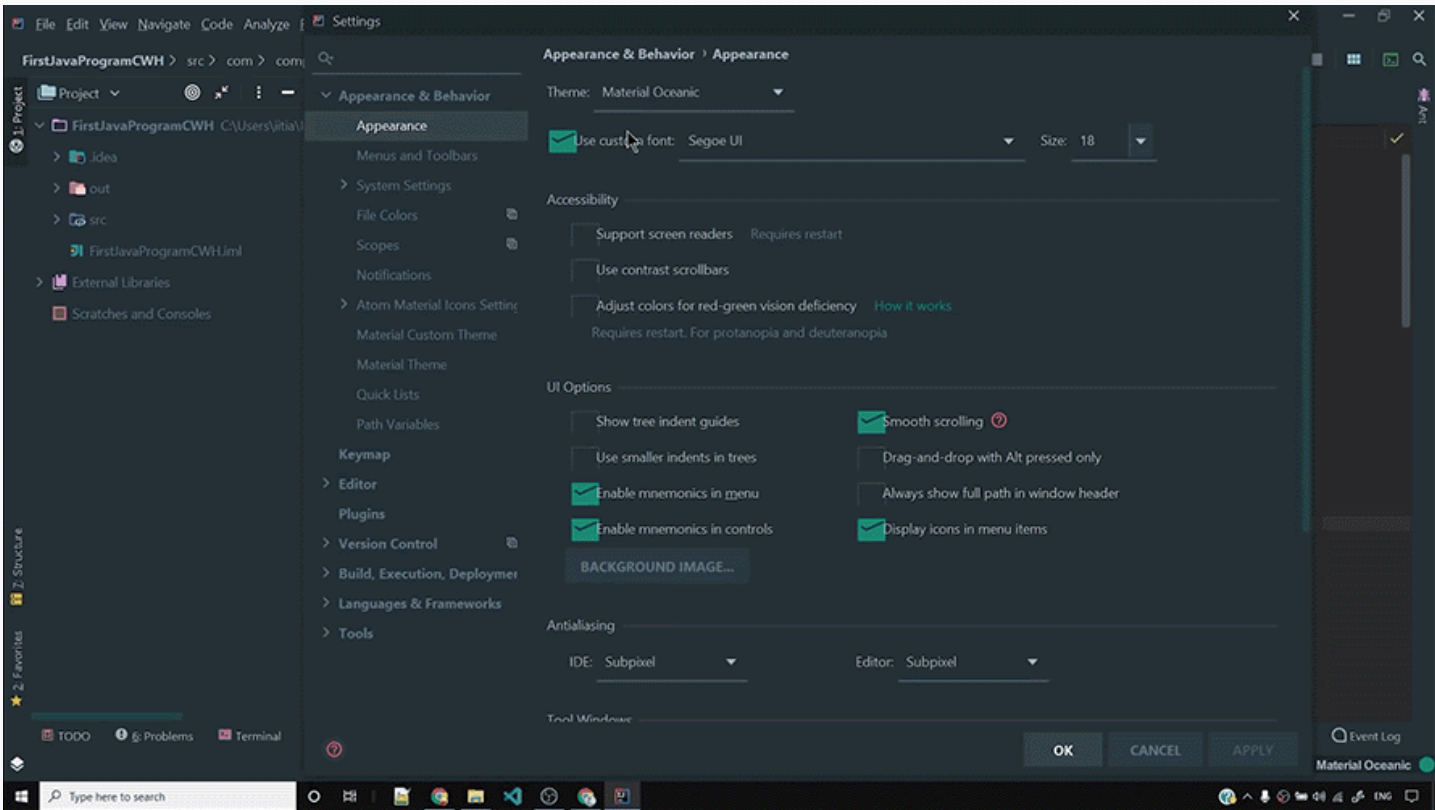
After installing both the plugins, click on the restart IDE option to restart the IDE so that all the changes can take effect.



Now go to File > Settings > Appearance & Behaviour > Appearance, and there you'll get many of the new themes that you can use to make your IDE look professional.



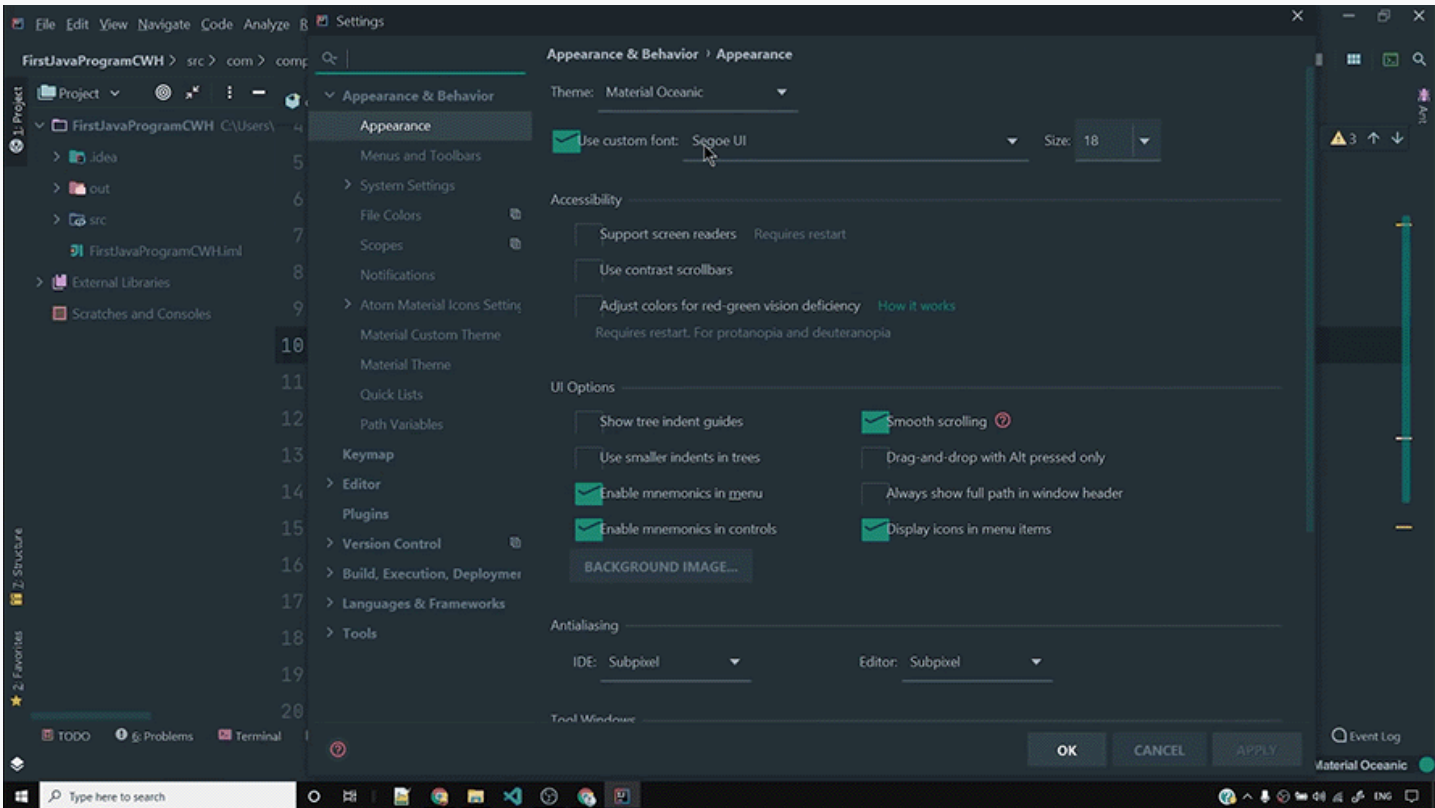
You have to choose any one of the themes and then click on the apply option to see its effect.



To Change the Font:

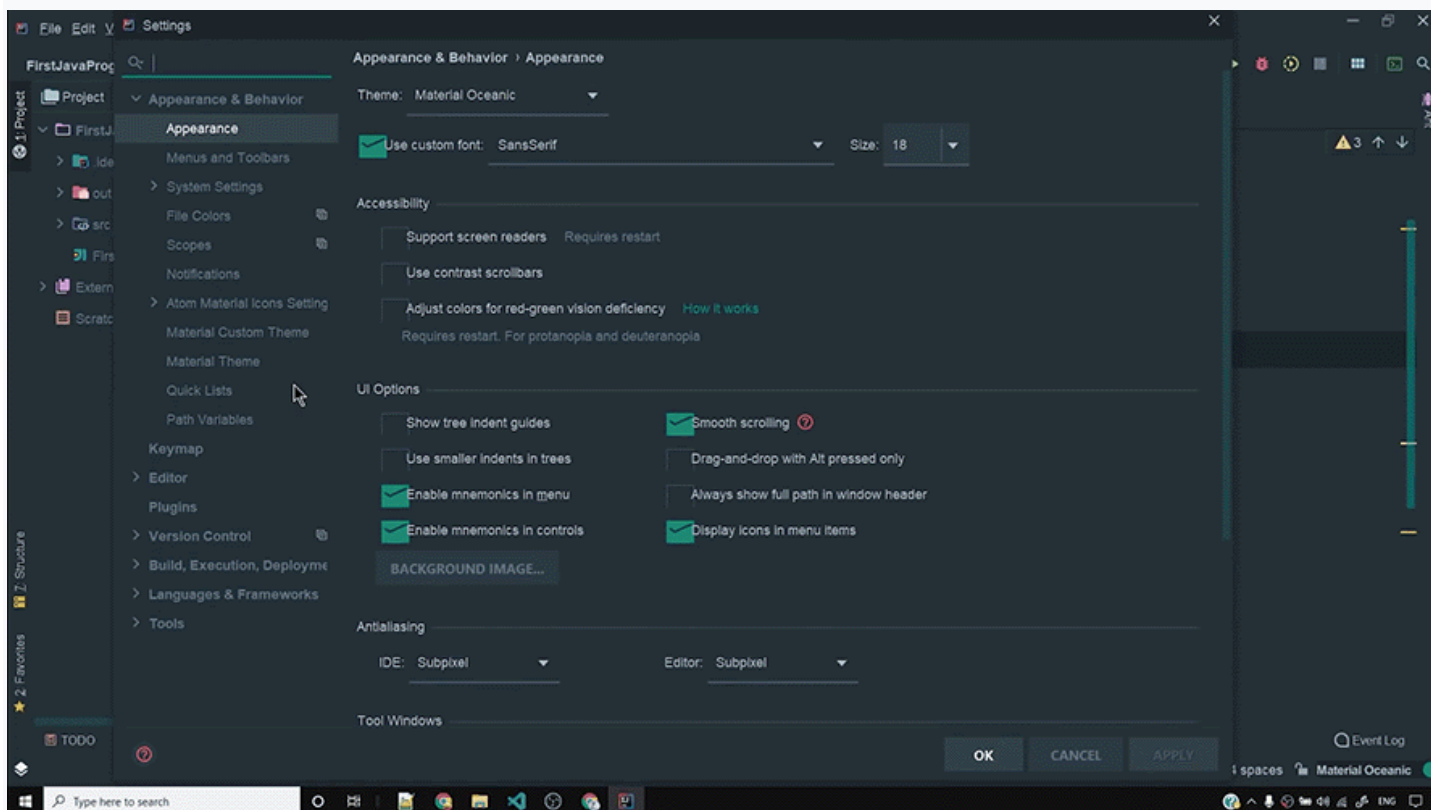
You can even change the font type and size of IDE according to your choice.

In Appearance, you will get the option to use custom font. And there, you can choose any font and can adjust its size accordingly.



To Change Icons:

In Appearance, you'll get the option of "Atom Material Icons Setting", and there you can change the icons of different things of IDE.



Using the methods mentioned above, you can change the look of the IDE the way you want.

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Methods in Java

- Sometimes our program grows in size, and we want to separate the logic of the main method from the other methods.
- For instance, if we calculate the 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.

Syntax of a method :

```
returnType nameOfMethod() {  
    //Method body  
}
```

Copy

The following method returns the sum of two numbers

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

Copy

- In the above method, int is the return data type of the mySum function.
- mySum takes two parameters: int a and int b.
- The sum of two values integer values(a and b) is stored in another integer value named 'c'.
- mySum returns c.

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
```

Copy

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 :

- The static keyword is used to associate a method of a given class with the class rather than the object.
- You can call a static method without creating an instance of the class.
- In Java, the main() method is static, so that JVM can call the main() method directly without allocating any extra memory for object creation.
- All the objects share the static method in a class.

Process of method invocation in Java :

Consider the method Sum of the calculate class as given in the below code :

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

Copy

The method is called like this:

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

public static void main(String[] args) {

    calculate obj = new calculate();
    int c = obj.sum(5,4);
    System.out.println(c);
}
}
```

Copy

Output :

```
9
```

Copy

- Inside the main() method, we've created an object of the calculate class.
- obj is the name of the calculate class.
- Then, we've invoked the sum method and passed 5 and 4 as arguments.

Note: In the case of Arrays, the reference is passed. The same is the case for object passing to methods.

Source code as described in the video:

```
package com.company;

public class cwh_31_methods {

    static int logic(int x, int y){
```



```

    int z;
    if(x>y){
        z = x+y;
    }
    else {
        z = (x +y) * 5;
    }
    x = 566;
    return z;
}

```

```

public static void main(String[] args) {
    int a = 5;
    int b = 7;
    int c;
    // Method invocation using Object creation
    //cwh_31_methods obj = new cwh_31_methods();
    //c = obj.logic(a, b);
    c = logic(a, b);
    System.out.println(a + " "+ b);
    int a1 = 2;
    int b1 = 1;
    int c1;
    c1 = logic(a1, b1);
    System.out.println(c);
    System.out.println(c1);
}
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Method Overloading in Java

- In Java, it is possible for a class to contain two or more methods with the same name but with different parameters. Such methods are called Overloaded methods.
- Method overloading is used to increase the readability of the program.

```

void foo()
void foo(int a)      //Overloaded function foo
int foo(int a, int b)

```

Copy

Ways to perform method overloading :

In Java, method overloading can be performed by two ways listed below :

1. By changing the return type of the different methods
2. By changing the number of arguments accepted by the method

Now, let's have an example to understand the above ways of method overloading :

1. By changing the return type :
 - In the below example, we've created a class named calculate.
 - In the calculate class, we've two methods with the same name i.e. multiply
 - These two methods are overloaded because they have the same name but their return is different.

- The return type of 1st method is int while the return type of the other method is double.

```
class calculate{
    int multiply(int a,int b){
        return a*b;
    }
    double multiply(double a,double b){
        return a*b;
    }

    public static void main(String[] args) {

        calculate obj = new calculate();
        int c = obj.multiply(5,4);
        double d = obj.multiply(5.1,4.2);
        System.out.println("Mutiply method : returns integer : " + c);
        System.out.println("Mutiply method : returns double : " + d);

    }
}
```

Copy

Output :

```
Mutiply method : returns integer : 20
Mutiply method : returns double : 21.419999999999998
```

Copy

2.

3. By changing the number of arguments passed :

- Again, we've created two methods with the same name i.e., multiply
- The return type of both the methods is int.
- But, the first method 2 arguments and the other method accepts 3 arguments.

Example :

```
class calculate{
    int multiply(int a,int b){
        return a*b;
    }
    int multiply(int a,int b,int c){
        return a*b*c;
    }

    public static void main(String[] args) {

        calculate obj = new calculate();
        int c = obj.multiply(5,4);
        int d = obj.multiply(5,4,3);
        System.out.println(c);
        System.out.println(d);

    }
}
```

Copy

Output :

20

60

Copy

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

Source code as described in the video:

```
package com.company;

public class cwh_32_method_overloading {
    static void foo(){
        System.out.println("Good Morning bro!");
    }

    static void foo(int a){
        System.out.println("Good morning " + a + " bro!");
    }

    static void foo(int a, int b){
        System.out.println("Good morning " + a + " bro!");
        System.out.println("Good morning " + b + " bro!");
    }

    static void foo(int a, int b, int c){
        System.out.println("Good morning " + a + " bro!");
        System.out.println("Good morning " + b + " bro!");
    }

    static void change(int a){
        a = 98;
    }

    static void change2(int [] arr){
        arr[0] = 98;
    }
    static void tellJoke(){
        System.out.println("I invented a new word!\n" +
            "Plagiarism!");
    }

    public static void main(String[] args) {
        // tellJoke();

        // Case 1: Changing the Integer
        //int x = 45;
        //change(x);
        //System.out.println("The value of x after running change is: " + x);

        // Case 1: Changing the Array
        // int [] marks = {52, 73, 77, 89, 98, 94};
        // change2(marks);
        // System.out.println("The value of x after running change is: " + marks[0]);
    }
}
```

```
// Method Overloading
foo();
foo(3000);
foo(3000, 4000);
// Arguments are actual!

}
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Variable Arguments (VarArgs) in Java

- In the previous tutorial, we discussed how we can [overload the methods in Java](#).
- Now, let's suppose you want to overload an "add" method. The "add" method will accept one argument for the first time and every time the number of arguments passed will be incremented by 1 till the number of arguments is equaled to 10.
- One approach to solve this problem is to overload the "add" method 10 times. But is it the optimal approach? What if I say that the number of arguments passed will be incremented by 1 till the number of arguments is equaled to 1000. Do you think that it is good practice to overload a method 1000 times?
- To solve this problem of method overloading, Variable Arguments(Varargs) were introduced with the release of JDK 5.
- With the help of Varargs, we do not need to overload the methods.

Syntax :

```
/*
public static void foo(int ... arr)
{
// arr is available here as int[] arr
}
*/
```

Copy

- foo can be called with zero or more arguments like this:
 - foo(7)
 - foo(7,8,9)
 - foo(1,2,7,8,9)

Example of Varargs In Java :

```
class calculate {

    static int add(int ...arr){
        int result = 0;
        for (int a : arr){
            result = result + a;
        }
        return result;
    }

}

public static void main(String[] args){
    System.out.println(add(1,2));
    System.out.println(add(2,3,4));
    System.out.println(add(4,5,6));
}
```

```
}  
}
```

Copy

Output :

```
3  
9  
15
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Recursion in Java

One does not simply understand RECURSION without understanding RECURSION.

- In programming, recursion is a technique through which a function calls itself.
- With the help of recursion, we can break down complex problems into simple problems.
- **Example:** Factorial of a number

```
//factorial(n) = n*factorial(n-1) [n >= 1]
```

Copy

Now, let's see an example to see the beauty of recursion in programming. First, we will print numbers from 1 to n and then n to 1 using recursion.

Program for printing 1 to n :

```
class recursion {  
    static void fun2(int n){  
        if(n>0){  
            fun2(n-1);  
            System.out.println(n);  
        }  
    }  
  
    public static void main(String[] args){  
        int n = 3;  
        fun2(n);  
    }  
}
```

Copy

Output :

```
1  
2  
3
```

Copy

In the above code, the print statement is getting executed at returning time. Watch the video given below to get the proper understanding of the recursive tree for the above program :

Program for printing n to 1 :

```
class recursion {  
    static void fun1(int n){  
        if(n>0){  
            System.out.println(n);  
            fun1(n-1);  
        }  
    }  
}
```

```
}  
public static void main(String[] args){  
    int n = 3;  
    fun1(n);  
}  
}
```

Copy

Output :

```
3  
2  
1
```

Copy

In the above recursive code, the print statement is getting executed at the calling time. Before the recursive function is called, printing was done. Watch the video given below to get the proper understanding of the recursive tree for the above program :

Notice that by just changing the order of the print statement, the output of the code is completely reversed. This is the beauty of recursion. The same trick can be used to reverse a linked list.

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

```
package com.company;  
  
public class cwh_34_recursion {  
    // factorial(0) = 1  
    // factorial(n) = n * n-1 *....1  
    // factorial(5) = 5 * 4 * 3 * 2 * 1 = 120  
    // factorial(n) = n * factorial(n-1)  
  
    static int factorial(int n){  
        if(n==0 || n==1){  
            return 1;  
        }  
        else{  
            return n * factorial(n-1);  
        }  
    }  
  
    static int factorial_iterative(int n){  
        if(n==0 || n==1){  
            return 1;  
        }  
        else{  
            int product = 1;  
            for (int i=1;i<=n;i++){ // 1 to n  
                product *= i;  
            }  
            return product;  
        }  
    }  
  
    public static void main(String[] args) {  
        int x = 0;  
        System.out.println("The value of factorial x is: " + factorial(x));  
        System.out.println("The value of factorial x is: " + factorial_iterative(x));  
    }  
}
```

Copy
Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Practice Questions on Java Methods

1. Write a Java method to print the multiplication table of a number n.
2. Write a program using functions to print the following pattern:

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

3. Write a recursive function to calculate the sum of first n natural numbers.
4. Write a function to print the following pattern:

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

5. Write a function to print the n^{th} term of the Fibonacci series using recursion.
6. Write a function to find the average of a set of numbers passed as arguments.
7. Repeat problem 4 using Recursion.
8. Repeat problem 2 using Recursion.
9. Write a function to convert Celsius temperature into Fahrenheit.
10. Repeat problem 3 using an iterative approach.

Code Solution:

```
package com.company;  
  
public class cwh_35_practice_set_on_methods {  
    static void multiplication(int n) {  
        for (int i = 1; i <= 10; i++) {  
            System.out.format("%d X %d = %d\n", n, i, n * i);  
        }  
    }  
  
    static void pattern1(int n) {  
        for (int i = 0; i < n; i++) {  
            for (int j = 0; j < i + 1; j++) {  
                System.out.print("*");  
            }  
            System.out.println();  
        }  
    }  
  
    static void pattern1_rec(int n) {  
        if (n > 0) {  
            pattern1_rec(n - 1);  
            for (int i = 0; i < n; i++) {  
                System.out.print("*");  
            }  
            System.out.println();  
        }  
    }  
}
```

```

    }
}
// pattern1_rec(3)
// pattern1_rec(2) + 3 times star and new line
// pattern1_rec(1) + 2 times star and new line + 3 times star and new line
// pattern1_rec(0) + 1 times star and new line + 2 times star and new line + 3 times star and
new line

// sum(n) = 1 + 2 + 3... + n
// sum(n) = 1 + 2 + 3... + n-1 + n
// sum(n) = sum(n-1) + n
// sum(3) = 3 + sum(2)
// sum(3) = 3 + 2 + sum(1)
// sum(3) = 3 + 2 + 1
static int sumRec(int n) {
    // Base condition
    if (n == 1) {
        return 1;
    }
    return n + sumRec(n - 1);
}

static int fib(int n) {
    /* if(n==1){
        return 0;
    }
    else if(n==2){
        return 1;
    } */
    if (n == 1 || n == 2) {
        return n - 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}

public static void main(String[] args) {
    // Problem 1
    // multiplication(7);

    // Problem 2
    // pattern1(9);

    // Problem 3
    // int c = sumRec(4);
    // System.out.println(c);

    // Problem 4
    // fibonacci series - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
    // int result = fib(7);
    // System.out.println(result);

    // Problem 8

```



```
pattern1(9);  
  
}  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

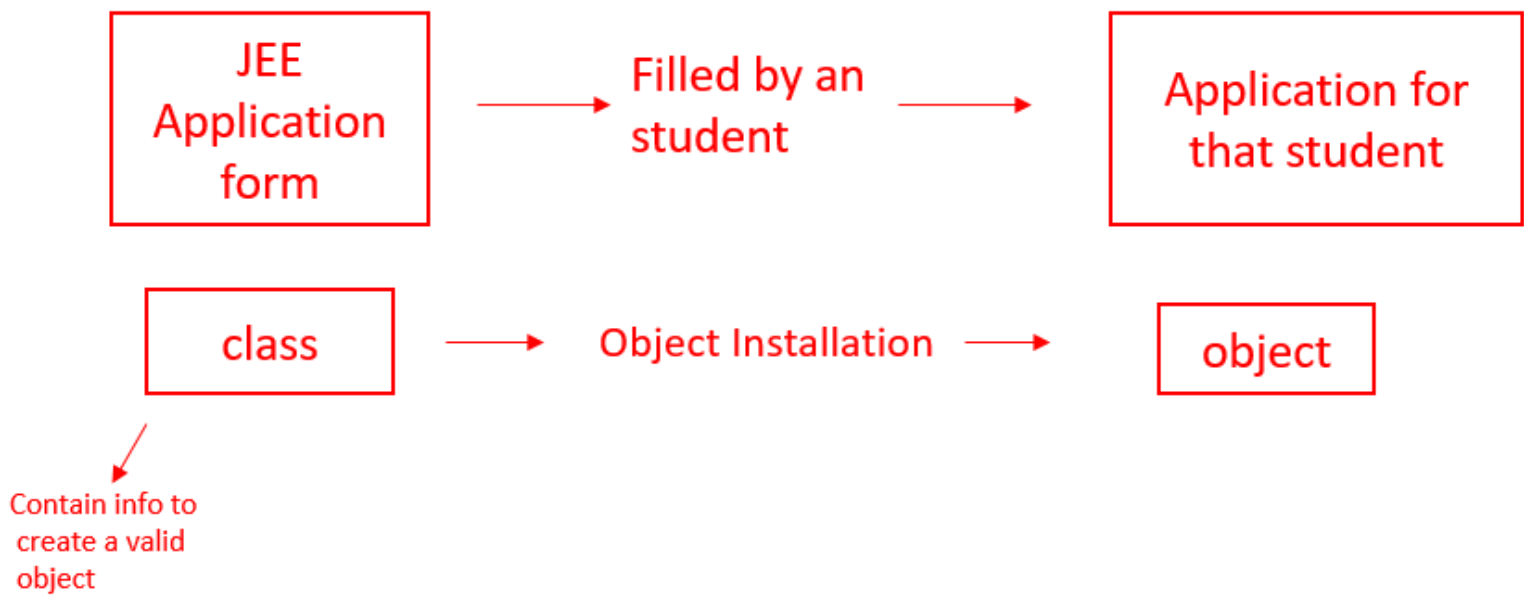
Java Tutorial: Introduction to Object Oriented Programming

- Object-Oriented Programming tries to map code instructions with real-world, making the code short and easier to understand.
- With the help of OOPs, we try to implement real-world entities such as object, inheritance, abstraction, etc.
- OOPs helps us to follow the DRY(Don't Repeat Yourself) approach of programming, which in turn increases the reusability of the code.

Two most important aspects of OOPs - Classes & Objects :

Class :

- A class is a blueprint for creating objects.
- Classes do not consume any space in the memory.
- Objects inherit methods and variables from the class.
- It is a logical component.



Objects :

- An object is an instantiation of a class. When a class is defined, a template (info) is defined.
- Every object has some address, and it occupies some space in the memory.
- It is a physical entity.

Take a look at the below example to get a better understanding of objects and classes :

Class - Fruits



Objects :

1.Mango :



2. Guava



3. Grapes



How to model a problem in OOPs

We identify the following:

- | | | |
|-------------|--------------|----------------------------|
| • Noun | - Class | - Employee |
| • Adjective | - Attributes | - name, age, salary |
| • Verb | - Methods | - getSalary(), increment() |

This is all for this tutorial. We will do a detailed discussion on every aspect of OOPs in further tutorials.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Basic Terminologies in Object Oriented Programming

Four pillars of Object-Oriented-Programming Language :

1. Abstraction :

- Let's suppose you want to turn on the bulb in your room. What do you do to switch on the bulb. You simply press the button and the light bulb turns on. Right? Notice that here you're only concerned with your final result, i.e., turning on the light bulb. You do not care about the circuit of the bulb or how current flows through the bulb. The point here is that you press the switch, the bulb turns on! You don't know how the bulb turned on/how the circuit is made because all these details are hidden from you. This phenomenon is known as abstraction.
- More formally, data abstraction is the way through which only the essential info is shown to the user, and all the internal details remain hidden from the user.

Example :

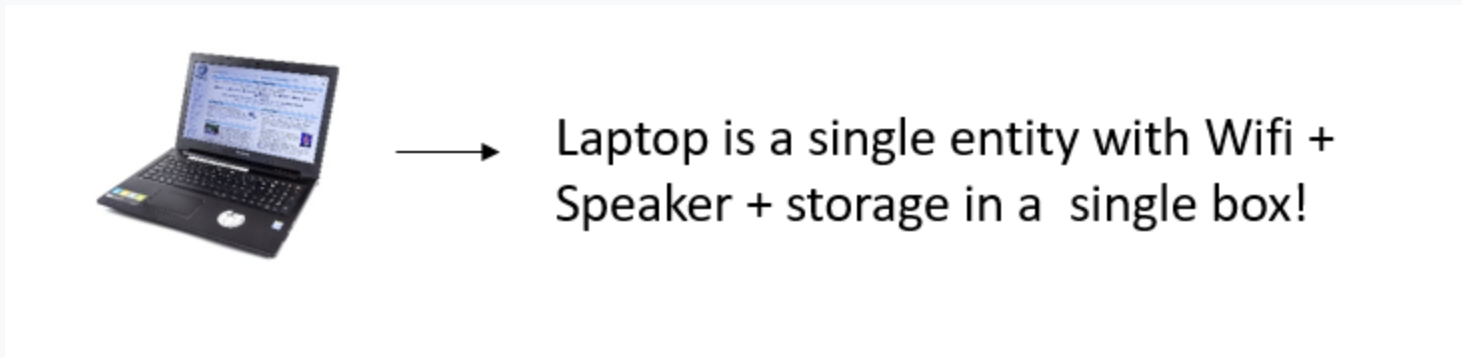


Use this phone without bothering
about how it was made

2. Polymorphism :

- One entity many forms.
- The word polymorphism comprises two words, poly which means many, and morph, which means forms.
- In OOPs, polymorphism is the property that helps to perform a single task in different ways.

- Let us consider a real-life example of polymorphism. A woman at the same time can be a mother, wife, sister, daughter, etc. Here, a woman is an entity having different forms.
- Let's take another example, a smartphone can work like a camera as well as like a calculator. So, you can see the a smartphone is an entity having different forms. Also :



3. Encapsulation :

- The act of putting various components together (in a capsule).
- In java, the variables and methods are the components that are wrapped inside a single unit named class.
- All the methods and variables of a class remain hidden from any other class.
- A automatic cold drink vending machine is an example of encapsulation.
- Cold drinks inside the machine are data that is wrapped inside a single unit cold drink vending machine.

4. Inheritance :

- The act of deriving new things from existing things.
- In Java, one class can acquire all the properties and behaviours of other some other class
- The class which inherits some other class is known as child class or sub class.
- The class which is inherited is known as parent class or super class.
- Inheritance helps us to write more efficient code because it increases the reusability of the code.
- Example :
 - Rickshaw → E-Rickshaw
 - Phone → Smart Phone

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Creating Our Own Java Class

Writing a Custom Class :

Syntax of a custom class :

```
class <class_name>{  
    field;  
    method;  
}
```

Copy

Example :

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

Copy

Note: The first letter of a class should always be capital.

- Any real-world object = Properties + Behavior
- 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
    }
};
*/
```

Copy

Code as Described in the Video

```
package com.company;

class Employee{
    int id;
    int salary;
    String name;
    public void printDetails(){
        System.out.println("My id is " + id);
        System.out.println("and my name is "+ name);
    }

    public int getSalary(){
        return salary;
    }
}

public class cwh_38_custom_class {
    public static void main(String[] args) {
        System.out.println("This is our custom class");
        Employee harry = new Employee(); // Instantiating a new Employee Object
        Employee john = new Employee(); // Instantiating a new Employee Object

        // Setting Attributes for Harry
        harry.id = 12;
        harry.salary = 34;
        harry.name = "CodeWithHarry";

        // Setting Attributes for John
        john.id = 17;
        john.salary = 12;
        john.name = "John Khandelwal";

        // Printing the Attributes
        harry.printDetails();
        john.printDetails();
        int salary = john.getSalary();
        System.out.println(salary);
        // System.out.println(harry.id);
    }
}
```

```
        // System.out.println(harry.name);  
    }  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Basic Questions on Object Oriented Programming

1. Create a class Employee with the 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 & problem 3.
5. Create a class TommyVecetti for Rockstar Games capable of hitting (print hitting...), running, firing, etc.
6. Repeat problem 4 for a circle.

Code as Described in the Video

```
package com.company;  
  
class Employee{  
    int salary;  
    String name;  
  
    public int getSalary(){  
        return salary;  
    }  
    public String getName(){  
        return name;  
    }  
    public void setName(String n){  
        name = n;  
    }  
}  
  
class CellPhone{  
    public void ring(){  
        System.out.println("Ringing...");  
    }  
    public void vibrate(){  
        System.out.println("Vibrating...");  
    }  
    public void callFriend(){  
        System.out.println("Calling Mukul...");  
    }  
}  
  
class Square{  
    int side;
```

```

    public int area(){
        return side*side;
    }
    public int perimeter(){
        return 4*side;
    }
}

class Tommy{
    public void hit(){
        System.out.println("Hitting the enemy");
    }
    public void run(){
        System.out.println("Running from the enemy");
    }
    public void fire(){
        System.out.println("Firing on the enemy");
    }
}

public class cwh_39_ch8ps {
    public static void main(String[] args) {
        /*
        // Problem 1
        Employee harry = new Employee();
        harry.setName("CodeWithHarry");
        harry.salary = 233;
        System.out.println(harry.getSalary());
        System.out.println(harry.getName());

        // Problem 2
        CellPhone asus = new CellPhone();
        asus.callFriend();
        asus.vibrate();
        //asus.ring();

        // Problem 3
        Square sq = new Square();
        sq.side = 3;
        System.out.println(sq.area());
        System.out.println(sq.perimeter());
        */

        // Problem 5
        Tommy player1 = new Tommy();
        player1.fire();
        player1.run();
        player1.hit();

    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Access modifiers, getters & setters in Java

Access Modifiers

Access Modifiers specify where a property/method is accessible. There are four types of access modifiers in java :

- 1. private
- 2. default
- 3. protected
- 4. public

Access Modifier	within class	within package	outside package by subclass only	outside package
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Default	Y	Y	N	N
private	Y	N	N	N

From the above table, notice that the private access modifier can only be accessed within the class. So, let's try to access private modifiers outside the class :

```
class Employee {

    private int id;
    private String name;

}

public class CWH {
    public static void main(String[] args) {
        Employee emp1 = new Employee();
        emp1.id = 3;
        emp1.name = "Shubham";

    }
}
```

Copy

Output :

```
java: id has private access in Employee
```

Copy

You can see that the above code produces an error that we're trying to access a private variable outside the class. So, is there any way by which we can access the private access modifiers outside the class? The answer is Yes! We can access the private access modifiers outside the class with the help of getters and setters.

Getters and Setters :

- Getter ➡ Returns the value [accessors]
- setter ➡ Sets / updates the value [mutators]

In the below code, we've created total 4 methods:

- 1. setName(): The argument passed to this method is assigned to the private variable name.
- 2. getName(): The method returns the value set by the setName() method.
- 3. setId(): The integer argument passed to this method is assigned to the private variable id.
- 4. getId(): This method returns the value set by the setId() method.

```
class Employee {

    private int id;
```

```

    private String name;

    public String getName(){
        return name;
    }
    public void setName(String n){
        name = n;
    }
    public void setId(int i){
        id = i;
    }
    public int getId(){
        return id;
    }
}

public class CWH {
    public static void main(String[] args) {
        Employee emp1 = new Employee();

        emp1.setName("Shubham");
        System.out.println(emp1.getName());
        emp1.setId(1);
        System.out.println(emp1.getId());

    }
}

```

Copy

Output :

```

Shubham
1

```

Copy

As you can see that we've got our expected output. So, that's how we use the getters and setters method to get and set the values of private access modifiers outside the class.

Source code as described in the video:

```

package com.company;

class MyEmployee{
    private int id;
    private String name;

    public String getName(){
        return name;
    }
    public void setName(String n){
        this.name = n;
    }
    public void setId(int i){
        this.id = i;
    }
    public int getId(){
        return id;
    }
}

```



```

    }
}
public class cwh_40_ch9 {
    public static void main(String[] args) {
        MyEmployee harry = new MyEmployee();
        // harry.id = 45;
        // harry.name = "CodeWithHarry"; --> Throws an error due to private access modifier
        harry.setName("CodeWithHarry");
        System.out.println(harry.getName());
        harry.setId(234);
        System.out.println(harry.getId());
    }
}

```

Copy

Quick Quiz: use the getters and setters from the main method

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise 2 - Solution and Shoutouts

Source code as described in the video:

```

package com.company;

import java.util.Random;
import java.util.Scanner;

public class cwh_41_ex2sol {
    public static void main(String[] args) {
        // 0 for Rock
        // 1 for Paper
        // 2 for Scissor

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter 0 for Rock, 1 for Paper, 2 for Scissor ");
        int userInput = sc.nextInt();

        Random random = new Random();
        int computerInput = random.nextInt(3);

        if (userInput == computerInput) {
            System.out.println("Draw");
        }
        else if (userInput == 0 && computerInput == 2 || userInput == 1 && computerInput == 0
                || userInput == 2 && computerInput == 1) {
            System.out.println("You Win!");
        } else {
            System.out.println("Computer Win!");
        }
        // System.out.println("Computer choice: " + computerInput);
        if(computerInput==0){
            System.out.println("Computer choice: Rock");
        }
        else if(computerInput==1){

```

```
        System.out.println("Computer choice: Paper");
    }
    else if(computerInput==2){
        System.out.println("Computer choice: Scissors");
    }
}
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Constructors in Java

Constructors in Java :

- Constructors are similar to methods,, but they are used to initialize an object.
- Constructors do not have any return type(not even void).
- Every time we create an object by using the new() keyword, a constructor is called.
- If we do not create a constructor by ourself, then the default constructor(created by Java compiler) is called.

Rules for creating a Constructor :

1. The class name and constructor name should be the same.
2. It must have no explicit return type.
3. It can not be abstract, static, final, and synchronized.

Types of Constructors in Java :

There are two types of constructors in Java :

1. **Defaut constructor** : A constructor with 0 parameters is known as default constructor.

Syntax :

```
<class_name>(){
//code to be executed on the execution of the constructor
}
```

Copy

Example :

```
class CWH {
    CWH(){
        System.out.println("This is the default constructor of CWH class.");
    }
}

public class CWH_constructors {
    public static void main(String[] args) {
        CWH obj1 = new CWH();

    }
}
```

Copy

Output :

```
This is the default constructor of CWH class.
```

Copy

In the above code, CWH() is the constructor of class CWH The CWH() constructor is invoked automatically with the creation of object ob1.

2. **Parameterized constructor** : A constructor with some specified number of parameters is known as a parameterized constructor.

Syntax :

```
<class-name>(<data-type> param1, <data-type> param2,.....){  
    //code to be executed on the invocation of the constructor  
}
```

Copy

Example :

```
class CWH {  
    CWH(String s, int b){  
  
        System.out.println("This is the " +b+ "th video of " + " " + s);  
    }  
  
}  
public class CWH_constructors {  
    public static void main(String[] args) {  
        CWH obj1 = new CWH("CodeWithHarry Java Playlist",42);  
  
    }  
}
```

Copy

Output :

```
This is the 42th video of  CodeWithHarry Java Playlist
```

Copy

In the above example, CWH() constructor accepts two parameters i.e., string s and int b.

Constructor Overloading in Java :

Just like methods, constructors can also be overloaded in Java. We can overload the Employee constructor like below:

```
public Employee (String n)  
    name = n;  
}
```

Copy

Note:

1. Constructors can take parameters without being overloaded
2. There can be more than two overloaded constructors

Let's take an example to understand the concept of constructor overloading.

Example :

In the below example, the class Employee has a constructor named Employee(). It takes two argument,i.e., string s & int i. The same constructor is overloaded and then it accepts three arguments i.e., string s, int i & int salary.

```
class Employee {  
    // First constructor  
    Employee(String s, int i){
```

```

        System.out.println("The name of the first employee is : " + s);
        System.out.println("The id of the first employee is : " + i);
    }
//    Constructor overloaded
    Employee(String s, int i, int salary){
        System.out.println("The name of the second employee is : " + s);
        System.out.println("The id of the second employee is : " + i);
        System.out.println("The salary of second employee is : " + salary);
    }
}

public class CWH_constructors {
    public static void main(String[] args) {
        Employee shubham = new Employee("Shubham",1);
        Employee harry = new Employee("Harry",2,70000);

    }
}

```

Copy

Output :

```

The name of the first employee is : Shubham
The id of the first employee is : 1
The name of the second employee is : Harry
The id of the second employee is : 2
The salary of second employee is : 70000

```

Copy

Quick quiz: Overloaded the employee constructor to initialize the salary to Rs 10,000

Source code as described in the video:

```

package com.company;

class MyMainEmployee{
    private int id;
    private String name;

    public MyMainEmployee(){
        id = 0;
        name = "Your-Name-Here";
    }
    public MyMainEmployee(String myName, int myId){
        id = myId;
        name = myName;
    }
    public MyMainEmployee(String myName){
        id = 1;
        name = myName;
    }
    public String getName(){
        return name;
    }
}

```

```

    public void setName(String n){
        this.name = n;
    }
    public void setId(int i){
        this.id = i;
    }
    public int getId(){
        return id;
    }
}

public class cwh_42_constructors {
    public static void main(String[] args) {
        //MyMainEmployee harry = new MyMainEmployee("ProgrammingWithHarry", 12);
        MyMainEmployee harry = new MyMainEmployee();
        //harry.setName("CodeWithHarry");
        //harry.setId(34);
        System.out.println(harry.getId());
        System.out.println(harry.getName());
    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Exercise 3: Guess the Number (OOPs Edition)

Create a class Game, which allows a user to play "Guess the Number" game once.

- Game should have the following methods:
- Constructor to generate the random number
- takeUserInput() to take a user input of number
- isCorrectNumber() to detect whether the number entered by the user is true
- getter and setter for noOfGuesses

Use properties such as noOfGuesses(int), etc to get this task done!

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise on Access Modifiers and Constructors

1. create a class cylinder and use getter and setters to set its radius and height
2. use ❶ to calculate surface and volume of the cylinder
3. Use a constructor and repeat ❶
4. Overload a constructor used to initialize a rectangle of length and breath 5 for using custom parameters
5. Repeat ❶ for a sphere

Source code as described in the video:

```

package com.company;

class Cylinder{
    private int radius;
    private int height;

    public Cylinder(int radius, int height) {
        this.radius = radius;
    }
}

```

```

        this.height = height;
    }

    public int getRadius() {
        return radius;
    }

    public void setRadius(int radius) {
        this.radius = radius;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }
    public double surfaceArea(){
        return 2* Math.PI* radius * radius + 2*Math.PI*radius*height;
    }
    public double volume(){
        return Math.PI * radius * radius * height;
    }
}

```

```

class Rectangle{
    private int length;
    private int breadth;

    public Rectangle() {
        this.length = 4;
        this.breadth = 5;
    }

    public Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public int getLength() {
        return length;
    }

    public int getBreadth() {
        return breadth;
    }
}

```

```

public class cwh_44_ps09 {

    public static void main(String[] args) {

```

```

/*
// Problem 1
Cylinder myCylinder = new Cylinder(9, 12);
//myCylinder.setHeight(12);
System.out.println(myCylinder.getHeight());
//myCylinder.setRadius(9);
System.out.println(myCylinder.getRadius());
// Problem 2
System.out.println(myCylinder.surfaceArea());
System.out.println(myCylinder.volume());
*/

// Problem 3
Rectangle r = new Rectangle(12, 56);
System.out.println(r.getLength());
System.out.println(r.getBreadth());

}
}

```

Copy

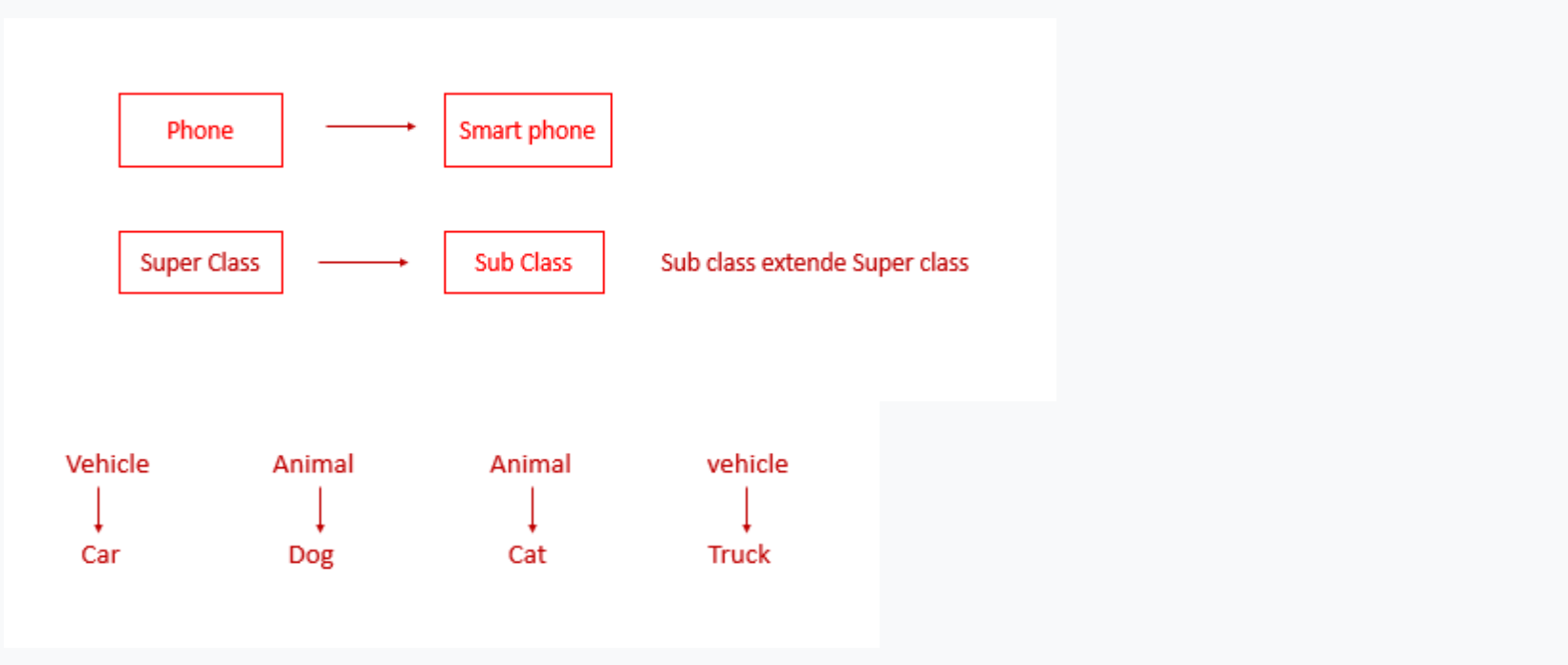
Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Inheritance in Java

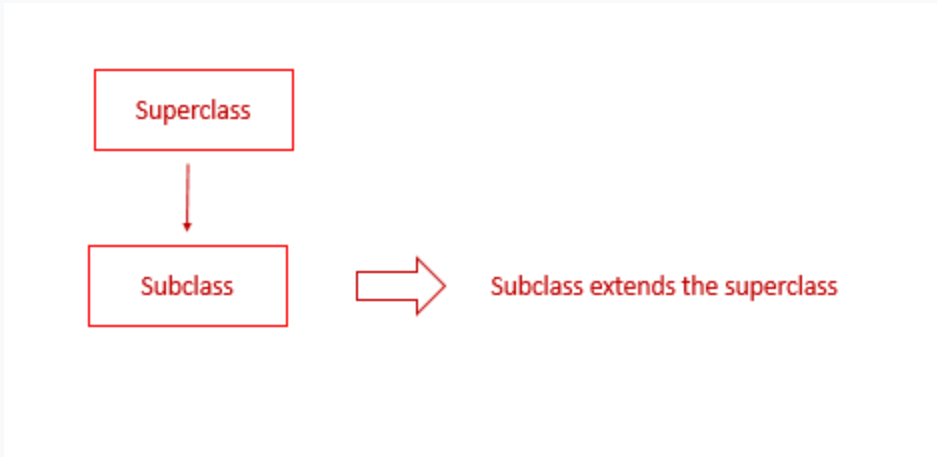
- You might have heard people saying your nose is similar to your father or mother. Or, more formally, we can say that you've inherited the genes from your parents due to which you look similar to them.
- The same phenomenon of inheritance is also valid in programming.
- In Java, one class can easily inherit the attributes and methods from some other class. This mechanism of acquiring objects and properties from some other class is known as inheritance in Java.
- Inheritance is used to borrow properties & methods from an existing class.
- Inheritance helps us create classes based on existing classes, which increases the code's reusability.

Examples :



Important terminologies used in Inheritance :

1. Parent class/superclass: The class from which a class inherits methods and attributes is known as parent class.
2. Child class/sub-class: The class that inherits some other class's methods and attributes is known as child class.



Extends keyword in inheritance :

- The **extends** keyword is used to inherit a subclass from a superclass.

Syntax :

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

Copy

Example :

```
public class dog extends Animal {
    // code
}
```

Copy

Note: [Java doesn't support multiple inheritances](#), i.e., two classes cannot be the superclass for a subclass.

Quick quiz: Create a class Animal and Derive another class dog from it

```
package com.company;
```

```
class Base{
```



```

    public int x;

    public int getX() {
        return x;
    }

    public void setX(int x) {
        System.out.println("I am in base and setting x now");
        this.x = x;
    }

    public void printMe(){
        System.out.println("I am a constructor");
    }
}

class Derived extends Base{
    public int y;

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}

public class cwh_45_inheritance {
    public static void main(String[] args) {
        // Creating an Object of base class
        Base b = new Base();
        b.setX(4);
        System.out.println(b.getX());

        // Creating an object of derived class
        Derived d = new Derived();
        d.setY(43);
        System.out.println(d.getY());
    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Constructors in Inheritance in Java

Constructors in Inheritance:

When a driven 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. C1- Parent
2. C2 - Child
3. C3 - Grandchild

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 the super keyword.
- super(a, b) calls the constructor from the parent class which takes 2 variables

Source code as described in the video:

```
package com.company;

class Base1{
    Base1(){
        System.out.println("I am a constructor");
    }
    Base1(int x){
        System.out.println("I am an overloaded constructor with value of x as: " + x);
    }
}

class Derived1 extends Base1{
    Derived1(){
        //super(0);
        System.out.println("I am a derived class constructor");
    }
    Derived1(int x, int y){
        super(x);
        System.out.println("I am an overloaded constructor of Derived with value of y as: " + y);
    }
}

class ChildOfDerived extends Derived1{
    ChildOfDerived(){
        System.out.println("I am a child of derived constructor");
    }
    ChildOfDerived(int x, int y, int z){
        super(x, y);
        System.out.println("I am an overloaded constructor of Derived with value of z as: " + z);
    }
}

public class cwh_46_constructors_in_inheritance {
    public static void main(String[] args) {
        // Base1 b = new Base1();
        // Derived1 d = new Derived1();
        // Derived1 d = new Derived1(14, 9);
        // ChildOfDerived cd = new ChildOfDerived();
        ChildOfDerived cd = new ChildOfDerived(12, 13, 15);
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

this and super keyword in Java

this keyword in Java :

- this is a way for us to reference an object of the class which is being created/referenced.
- It is used to call the default constructor of the same class.
- **this** keyword eliminates the confusion between the parameters and the class attributes with the same name. Take a look at the example given below :

```
class cwh{
    int x;

    //    getter of x
    public int getX(){
        return x;
    }

    // Constructor with a parameter
    cwh(int x) {
        x = x;
    }

    // Call the constructor
    public static void main(String[] args) {
        cwh obj1 = new cwh(65);
        System.out.println(obj1.getX());
    }
}
```

}

Copy

Output :

```
0
```

Copy

- In the above example, the expected output is 65 because we've passed x=65 to the constructor of the cwh class. But the compiler fails to differentiate between the parameter 'x' & class attribute 'x.' Therefore, it returns 0.
- Now, let's see how we can handle this situation with the help of this keyword. Take a look at the below code :

```
class cwh{
    int x;

    //    getter of x
    public int getX(){
        return x;
    }

    // Constructor with a parameter
    cwh(int x) {
        this.x = x;
    }

    // Call the constructor
    public static void main(String[] args) {
        cwh obj1 = new cwh(65);
        System.out.println(obj1.getX());
    }
}
```

```
}  
}
```

Copy

Output :

```
65
```

Copy

Now, you can see that we've got the desired output

Super keyword

- A reference variable used to refer immediate parent class object.
- It can be used to refer immediate parent class instance variable.
- It can be used to invoke the parent class method.

Source code as described in the video:

```
package com.company;  
import javax.print.Doc;  
class EkClass{  
int a;  
public int getA() {  
return a;  
}  
EkClass(int a){  
this.a = a;  
}  
public int returnnone(){  
return 1;  
}  
}  
class DoClass extends EkClass{ DoClass(int c){ super(c);  
System.out.println("I am a constructor"); }  
}  
public class cwh_47_this_super {  
public static void main(String[] args) {  
EkClass e = new EkClass(65);  
DoClass d = new DoClass(5);  
System.out.println(e.getA()); } }
```

Copy

- **Handwritten Notes:** [Click to Download](#)
- **Ultimate Java Cheatsheet:** [Click To Download](#)

Method Overriding in Java

Method Overriding in Java:

- If the child class implements the same method present in the parent class again, it is know as method overriding.
- Method overriding helps us to classify a behavior that is specific to the child class.
- The subclass can override the method of the parent class only when the method is not declared as final.
- Example :
- In the below code, we've created two classes: class A & class B.
- Class B is inheriting class A.

- In the main() method, we've created one object for both classes. We're running the meth1() method on class A and B objects separately, but the output is the same because the meth1() is defined in the parent class, i.e., class A.

```
class A{
    public void meth1(){
        System.out.println("I am method 1 of class A");
    }
}

class B extends A{
}

public class CWH{
    public static void main(String[] args) {
        A a = new A();
        a.meth1();

        B b = new B();
        b.meth1();
    }
}
```

Copy

Output :

```
I am method 1 of class A
I am method 1 of class A
```

Copy

- Now, let's see how we can override the meth1() for class B :

```
class A{
    public void meth1(){
        System.out.println("I am method 1 of class A");
    }
}

class B extends A{
    @Override
    public void meth1(){
        System.out.println("I am method 1 of class B");
    }
}

public class CWH{
    public static void main(String[] args) {
        A a = new A();
        a.meth1();

        B b = new B();
        b.meth1();
    }
}
```

Copy

Output :

```
I am method 1 of class A
I am method 1 of class B
```

Copy

Source code as described in the video:

```
package com.company;

class A{
    public int a;
    public int harry(){
        return 4;
    }
    public void meth2(){
        System.out.println("I am method 2 of class A");
    }
}

class B extends A{
    @Override
    public void meth2(){
        System.out.println("I am method 2 of class B");
    }
    public void meth3(){
        System.out.println("I am method 3 of class B");
    }
}

public class cwh_48_method_overriding {
    public static void main(String[] args) {
        A a = new A();
        a.meth2();

        B b = new B();
        b.meth2();
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Dynamic Method Dispatch in Java

- Dynamic method dispatch is also known as run time polymorphism.
- It is the process through which a call to an overridden method is resolved at runtime.
- This technique is used to resolve a call to an overridden method at runtime rather than compile time.
- To properly understand Dynamic method dispatch in Java, it is important to understand the concept of upcasting because dynamic method dispatch is based on upcasting.

Upcasting :

- It is a technique in which a superclass reference variable refers to the object of the subclass.

Example :

```
class Animal{}
class Dog extends Animal{}
```

Copy

```
Animal a=new Dog();//upcasting
```

Copy

In the above example, we've created two classes, named Animal(superclass) & Dog(subclass). While creating the object 'a', we've taken the reference variable of the parent class(Animal), and the object created is of child class(Dog).

Example to demonstrate the use of Dynamic method dispatch :

- In the below code, we've created two classes: **Phone & SmartPhone**.
- The **Phone** is the parent class and the **SmartPhone** is the child class.
- The method **on()** of the parent class is overridden inside the child class.
- Inside the main() method, we've created an object **obj** of the **Smartphone()** class by taking the reference of the **Phone()** class.
- When **obj.on()** will be executed, it will call the **on()** method of the **SmartPhone()** class because the reference variable obj is pointing towards the object of class **SmartPhone()**.

```
class Phone{
    public void showTime(){
        System.out.println("Time is 8 am");
    }
    public void on(){
        System.out.println("Turning on Phone...");
    }
}

class SmartPhone extends Phone{
    public void music(){
        System.out.println("Playing music...");
    }
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
}

public class CWH {
    public static void main(String[] args) {

        Phone obj = new SmartPhone(); // Yes it is allowed
        // SmartPhone obj2 = new Phone(); // Not allowed

        obj.showTime();
        obj.on();
        // obj.music(); Not Allowed

    }
}
```

Copy

Output :

```
Time is 8 am
Turning on SmartPhone...
```

Copy

Note: The data members can not achieve the run time polymorphism.

Code as described/written in the video :

```

package com.company;

class Phone{
    public void showTime(){
        System.out.println("Time is 8 am");
    }
    public void on(){
        System.out.println("Turning on Phone...");
    }
}

class SmartPhone extends Phone{
    public void music(){
        System.out.println("Playing music...");
    }
    public void on(){
        System.out.println("Turning on SmartPhone...");
    }
}

public class cwh_49_dynamic_method_dispatch {
    public static void main(String[] args) {
        // Phone obj = new Phone(); // Allowed
        // SmartPhone smobj = new SmartPhone(); // Allowed
        // obj.name();

        Phone obj = new SmartPhone(); // Yes it is allowed
        // SmartPhone obj2 = new Phone(); // Not allowed

        obj.showTime();
        obj.on();
        // obj.music(); Not Allowed

    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise 3 - Solutions & Shoutouts

Create a class Game, which allows a user to play "Guess the Number" game once.

- Game should have the following methods:
- Constructor to generate the random number
- takeUserInput() to take a user input of number
- isCorrectNumber() to detect whether the number entered by the user is true
- getter and setter for noOfGuesses

Use properties such as noOfGuesses(int), etc to get this task done!

```

package com.company;

import java.util.Random;
import java.util.Scanner;

class Game{
    public int number;
    public int inputNumber;
}

```



```

public int noOfGuesses = 0;

public int getNoOfGuesses() {
    return noOfGuesses;
}

public void setNoOfGuesses(int noOfGuesses) {
    this.noOfGuesses = noOfGuesses;
}

Game(){
    Random rand = new Random();
    this.number = rand.nextInt(100);
}

void takeUserInput(){
    System.out.println("Guess the number");
    Scanner sc = new Scanner(System.in);
    inputNumber = sc.nextInt();
}

boolean isCorrectNumber(){
    noOfGuesses++;
    if (inputNumber==number){
        System.out.format("Yes you guessed it right, it was %d\nYou guessed it in %d
attempts", number, noOfGuesses);
        return true;
    }
    else if(inputNumber<number){
        System.out.println("Too low...");
    }
    else if(inputNumber>number){
        System.out.println("Too high...");
    }
    return false;
}

}

public class cwh_50_ex3sol {
    public static void main(String[] args) {
        /*
            Create a class Game, which allows a user to play "Guess the Number"
            game once. Game should have the following methods:
            1. Constructor to generate the random number
            2. takeUserInput() to take a user input of number
            3. isCorrectNumber() to detect whether the number entered by the user is true
            4. getter and setter for noOfGuesses
            Use properties such as noOfGuesses(int), etc to get this task done!
        */

        Game g = new Game();
        boolean b= false;
        while(!b){
            g.takeUserInput();
            b = g.isCorrectNumber();
        }
    }
}

```

```
}  
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise 4 - Online Library

You have to implement a library using Java Class "Library"

- Methods: addBook, issueBook, returnBook, showAvailableBooks
- Properties: Array to store the available books,
- Array to store the issued books

```
package com.company;  
  
public class cwh_51_exercise4 {  
    public static void main(String[] args) {  
        // You have to implement a library using Java Class "Library"  
        // Methods: addBook, issueBook, returnBook, showAvailableBooks  
        // Properties: Array to store the available books,  
        // Array to store the issued books  
        //  
    }  
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Exercise & Practice Questions on Inheritance

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 the real-world scenario as possible
3. Create a method 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

Derived obj = new Derived 2();
Which constructor(s) will be executed & in what order?

```
package com.company;  
  
class Circle{  
    public int radius;  
    Circle(){  
        System.out.println("I am non param of circle");  
    }  
    Circle(int r){  
        System.out.println("I am circle parameterized constructor");  
        this.radius = r;  
    }  
  
    public double area(){  
        return Math.PI*this.radius*this.radius;  
    }  
}
```

```

}

class Cylinder1 extends Circle{
    public int height;
    Cylinder1(int r, int h){
        super(r);
        System.out.println("I am cylinder1 parameterized constructor");
        this.height = h;
    }
    public double volume(){
        return Math.PI*this.radius*this.radius*this.height;
    }
}

public class cwh_52_ch10ps {
    public static void main(String[] args) {
        // Problem 1
        // Circle objC = new Circle(12);
        Cylinder1 obj = new Cylinder1(12, 4);
    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Abstract Class & Abstract Methods

What does Abstract mean?

Abstract in English means existing in through or as an idea without concrete existence.

Abstract class :

- An abstract class cannot be instantiated.
- Java requires us to extend it if we want to access it.
- It can include abstract and non-abstract methods.
- If a class includes abstract methods, then the class itself must be declared abstract, as in:

```

public abstract class phone Model {
    abstract void switch off ();
    || more code
}

```

Copy

- Abstract class are used when we want to achieve security & abstraction(hide certain details & show only necessary details to the user)

Example :

```

abstract class Phone{
    abstract void on();
}

class SmartPhone extends Phone{
void run(){
System.out.println("Turning on...");
}

public static void main(String args[]){
    Phone obj = new SmartPhone();
    obj.on();
}
}

```

```
}
```

Copy

Output :

```
Turning on...
```

Copy

Abstract method :

- A method that is declared without implementation is known as the abstract method.
- An abstract method can only be used inside an abstract class.
- The body of the abstract method is provided by the class that inherits the abstract class in which the abstract method is present.
- In the above example, on() is the abstract method.

Code as described/written in the video :

```
package com.company;

abstract class Parent2{
    public Parent2(){
        System.out.println("Mai base2 ka constructor hoon");
    }
    public void sayHello(){
        System.out.println("Hello");
    }
    abstract public void greet();
    abstract public void greet2();
}

class Child2 extends Parent2{
    @Override
    public void greet(){
        System.out.println("Good morning");
    }
    @Override
    public void greet2(){
        System.out.println("Good afternoon");
    }
}

abstract class Child3 extends Parent2{
    public void th(){
        System.out.println("I am good");
    }
}

public class cwh_53_abstract {
    public static void main(String[] args) {
        //Parent2 p = new Parent2(); -- error
        Child2 c = new Child2();
        //Child3 c3 = new Child3(); -- error
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Introduction to Interfaces

Interfaces in Java :

- Just like a class in java is a collection of the related methods, an interface in java is a collection of abstract methods.
- The interface is one more way to achieve abstraction in Java.
- An interface may also contain constants, default methods, and static methods.
- All the methods inside an interface must have empty bodies except default methods and static methods.
- We use the **interface** keyword to declare an interface.
- There is no need to write **abstract** keyword before declaring methods in an interface because an interface is implicitly abstract.
- An interface cannot contain a constructor (as it cannot be used to create objects)
- In order to implement an interface, java requires a class to use the **implement** keyword.

Example to demonstrate Interface in Java :

```
interface Bicycle {
    void apply brake ( int decrement );
    void speed up ( int increment );
}

class Avon cycle implements Bicycle {
    int speed = 7 ;
    void apply brake ( int decrement ) {
        speed = speed - decrement ;
    }
    void speedup ( int increment ){
        speed = speed + increment ;
    }
}
```

Copy

Code as described/written in the video :

```
package com.company;

interface Bicycle{
    int a = 45;
    void applyBrake(int decrement);
    void speedUp(int increment);
}

interface HornBicycle{
    int x = 45;
    void blowHornK3g();
    void blowHornmhn();
}

class AvonCycle implements Bicycle, HornBicycle{
    //public int x = 5;
    void blowHorn(){
        System.out.println("Pee Pee Poo Poo");
    }
    public void applyBrake(int decrement){
        System.out.println("Applying Brake");
    }
}
```

```
        public void speedUp(int increment){
            System.out.println("Applying SpeedUP");
        }
        public void blowHornK3g(){
            System.out.println("Kabhi khushi kabhi gum pee pee pee pee");
        }
        public void blowHornmhn(){
            System.out.println("Main hoon naa po po po po");
        }
    }
}

public class cwh_54_interfaces {
    public static void main(String[] args) {
        AvonCycle cycleHarry = new AvonCycle();
        cycleHarry.applyBrake(1);
        // You can create properties in Interfaces
        System.out.println(cycleHarry.a);
        System.out.println(cycleHarry.x);

        // You cannot modify the properties in Interfaces as they are final
        // cycleHarry.a = 454;
        //System.out.println(cycleHarry.a);

        cycleHarry.blowHornK3g();
        cycleHarry.blowHornmhn();
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Abstract Classes Vs Interfaces

Abstract class	Interface
1. It can contain abstract and non-abstract method	It can only contain abstract methods. We do not need to use the "abstract" keyword in interface methods because the interface is implicitly abstract.
2. abstract keyword is used to declare an abstract class.	interface keyword is used to declare an interface.
3. A sub-class extends the abstract class by using the "extends" keyword.	The "implements" keyword is used to implement an interface.
4. A abstract class in Java can have class members like private, protected, etc.	Members of a Java interface are public by default.
5. Abstract class doesn't support multiple inheritance.	Multiple inheritance is achieved in Java by using the interface.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Why multiple inheritance is not supported in java?

Is multiple inheritance allowed in Java?

- Multiple inheritance faces problems when there exists a method with the same signature in both the superclasses.
- Due to such a problem, 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.

Some Important points :

1. Interfaces in java are a bit like the class but with a significantly different.
2. An Interface can only have method signatures field and a default method.
3. The class implementing an interface needs to declare the methods (not field)
4. You can create a reference of an interface but not the object
5. Interface methods are public by default

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Interfaces Example & Default Methods

Default methods In Java:

- 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.
- A class implementing the interface need not implement the default methods.
- Interfaces can also include private methods for default methods to use.
- You can easily override a default method like any other method of an interface.

Example :

```
interface Animal{
    // Default method
    default void say(){
        System.out.println("Hello, this is default method");
    }
    // Abstract method
    void bark();
}
public class CWH implements Animal{

    @Override
    public void bark() {
        System.out.println("Dog barks!");
    }
    public static void main(String[] args) {
        CWH obj1 = new CWH();
        obj1.bark();
        obj1.say();

    }

}
```

Copy

Output :

```
Dog barks!
Hello, this is default method
```

Copy

Code as described/written in the video :

```
package com.company;

interface MyCamera{
```

```

    void takeSnap();
    void recordVideo();
    private void greet(){
        System.out.println("Good Morning");
    }
    default void record4KVideo(){
        greet();
        System.out.println("Recording in 4k...");
    }
}

interface MyWifi{
    String[] getNetworks();
    void connectToNetwork(String network);
}

class MyCellPhone{
    void callNumber(int phoneNumber){
        System.out.println("Calling "+ phoneNumber);
    }
    void pickCall(){
        System.out.println("Connecting... ");
    }
}

class MySmartPhone extends MyCellPhone implements MyWifi, MyCamera{
    public void takeSnap(){
        System.out.println("Taking snap");
    }
    public void recordVideo(){
        System.out.println("Taking snap");
    }
    // public void record4KVideo(){
    //     System.out.println("Taking snap and recoding in 4k");
    // }
    public String[] getNetworks(){
        System.out.println("Getting List of Networks");
        String[] networkList = {"Harry", "Prashanth", "Anjali5G"};
        return networkList;
    }
    public void connectToNetwork(String network){
        System.out.println("Connecting to " + network);
    }
}

public class cwh_57_default_methods {
    public static void main(String[] args) {
        MySmartPhone ms = new MySmartPhone();
        ms.record4KVideo();
        // ms.greet(); --> Throws an error!
        String[] ar = ms.getNetworks();
        for (String item: ar) {
            System.out.println(item);
        }
    }
}

```



```
}  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Inheritance in Interfaces

Interfaces can extend other interfaces as shown below :

```
public interface Interface 1 {  
    void meth1 ();  
}  
  
public interface Interface 2 extends Interface 1 {  
    void meth 2( );  
}
```

Copy

Note: Remember that interface cannot implement another interface only classes can do that!

Code as described/written in the video :

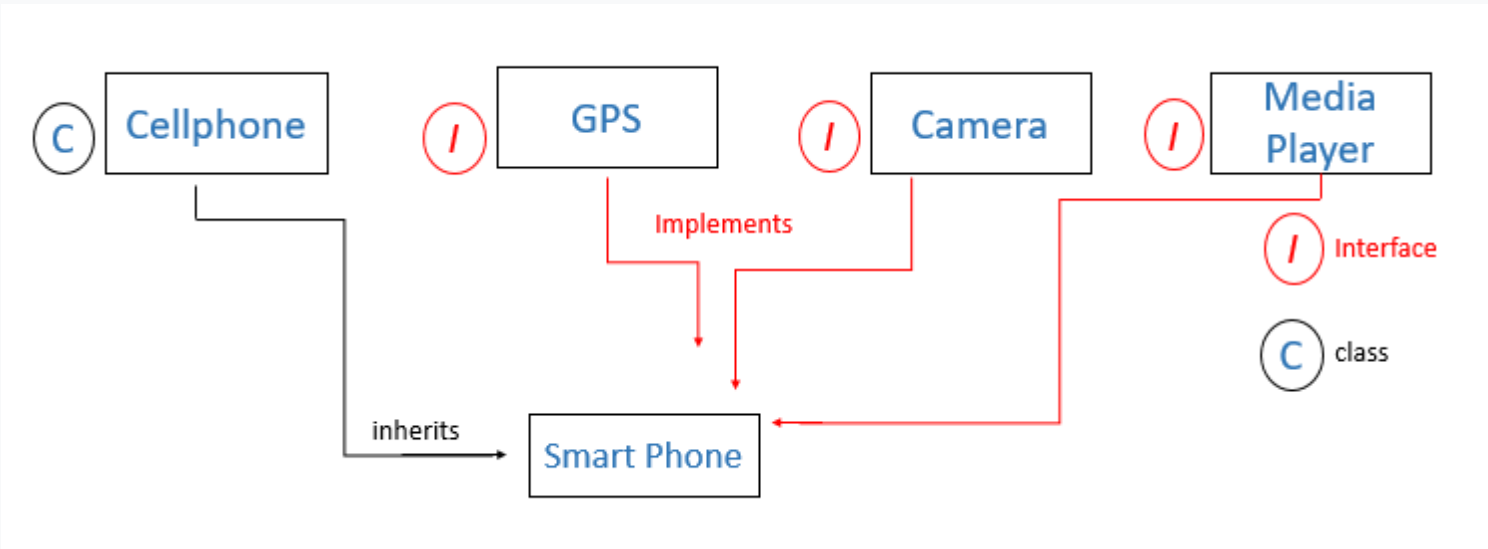
```
package com.company;  
  
interface sampleInterface{  
    void meth1();  
    void meth2();  
}  
interface childSampleInterface extends sampleInterface{  
    void meth3();  
    void meth4();  
}  
class MySampleClass implements childSampleInterface{  
    public void meth1(){  
        System.out.println("meth1");  
    }  
    public void meth2(){  
        System.out.println("meth2");  
    }  
    public void meth3(){  
        System.out.println("meth3");  
    }  
    public void meth4(){  
        System.out.println("meth4");  
    }  
}  
public class cwh_58_inheritance_interfaces {  
    public static void main(String[] args) {  
        MySampleClass obj = new MySampleClass();  
        obj.meth1();  
        obj.meth2();  
        obj.meth3();  
    }  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Polymorphism in Interfaces



GPS g = new Smartphone (); can only use GPS method
Smartphones = new Smartphone (); can only use smartphone methods

Implementing an Interface force method implementation

Code as described/written in the video :

```
package com.company;

interface MyCamera2{
    void takeSnap();
    void recordVideo();
    private void greet(){
        System.out.println("Good Morning");
    }
    default void record4KVideo(){
        greet();
        System.out.println("Recording in 4k...");
    }
}

interface MyWifi2{
    String[] getNetworks();
    void connectToNetwork(String network);
}

class MyCellPhone2{
    void callNumber(int phoneNumber){
        System.out.println("Calling "+ phoneNumber);
    }
    void pickCall(){
        System.out.println("Connecting... ");
    }
}

class MySmartPhone2 extends MyCellPhone2 implements MyWifi2, MyCamera2{
    public void takeSnap(){
        System.out.println("Taking snap");
    }
}
```

```

    public void recordVideo(){
        System.out.println("Taking snap");
    }
    //    public void record4KVideo(){
//        System.out.println("Taking snap and recoding in 4k");
//    }
    public String[] getNetworks(){
        System.out.println("Getting List of Networks");
        String[] networkList = {"Harry", "Prashanth", "Anjali5G"};
        return networkList;
    }
    public void connectToNetwork(String network){
        System.out.println("Connecting to " + network);
    }
    public void sampleMeth(){
        System.out.println("meth");
    }
}

public class cwh_59_polymorphism {
    public static void main(String[] args) {
        MyCamera2 cam1 = new MySmartPhone2(); // This is a smartphone but, use it as a camera
        // cam1.getNetworks(); --> Not allowed
        // cam1.sampleMeth(); --> Not allowed

        cam1.record4KVideo();

        MySmartPhone2 s = new MySmartPhone2();
        s.sampleMeth();
        s.recordVideo();
        s.getNetworks();
        s.callNumber(7979);
    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Practice Questions on Abstract Classes & Interfaces

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 fountain pen with additional method change Nib ()
3. Create a class monkey with jump () and bite () methods Create a class human which inherits this monkey class and implements basic animal interface with eat () and sleep methods
4. Create a class telephone with () , lift () and disconnected () methods as abstract methods create another class smart telephone and demonstrate polymorphism
5. Demonstrate polymorphism using using monkey class from Q3
6. Create an interface TVremote and use it to inherit another interface smart TVremote
7. Create a class TV which implements TVremote interface from Q6

```

package com.company;

abstract class Pen{
    abstract void write();
    abstract void refill();
}

```

```
class FountainPen extends Pen{
    void write(){
        System.out.println("Write");
    }
    void refill(){
        System.out.println("Refill");
    }
    void changeNib(){
        System.out.println("Changing the nib");
    }
}

class Monkey{
    void jump(){
        System.out.println("Jumping...");
    }
    void bite(){
        System.out.println("Biting...");
    }
}

interface BasicAnimal{
    void eat();
    void sleep();
}

class Human extends Monkey implements BasicAnimal{
    void speak(){
        System.out.println("Hello sir!");
    }

    @Override
    public void eat() {
        System.out.println("Eating");
    }

    @Override
    public void sleep() {
        System.out.println("Sleeping");
    }
}

public class cwh_60_ch11ps {
    public static void main(String[] args) {
        // Q1 + Q2
        FountainPen pen = new FountainPen();
        pen.changeNib();

        // Q3
        Human harry = new Human();
        harry.sleep();

        // Q5
        Monkey m1 = new Human();
    }
}
```

```

        m1.jump();
        m1.bite();
        // m1.speak(); --> Cannot use speak method because the reference is monkey which does not
        have speak method

        BasicAnimal lovish = new Human();
        // lovish.speak(); --> error
        lovish.eat();
        lovish.sleep();

    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Exercise 4: Solution & Shoutouts!

```

package com.company;
class Library{
    String[] books;
    int no_of_books;
    Library(){
        this.books = new String[100];
        this.no_of_books = 0;
    }

    void addBook(String book){
        this.books[no_of_books] = book;
        no_of_books++;
        System.out.println(book+ " has been added!");
    }

    void showAvailableBooks(){
        System.out.println("Available Books are:");
        for (String book : this.books) {
            if (book == null){
                continue;
            }
            System.out.println("* " + book);
        }
    }

    void issueBook(String book){
        for (int i=0;i<this.books.length;i++){
            if (this.books[i].equals(book)){
                System.out.println("The book has been issued!");
                this.books[i] = null;
                return;
            }
        }
        System.out.println("This book does not exist");
    }

    void returnBook(String book){

```

```

        addBook(book);
    }

}

public class cwh_61_ex4sol {
    public static void main(String[] args) {
        // You have to implement a library using Java Class "Library"
        // Methods: addBook, issueBook, returnBook, showAvailableBooks
        // Properties: Array to store the available books,
        // Array to store the issued books

        Library centralLibrary = new Library();
        centralLibrary.addBook("Think and grow Rich");
        centralLibrary.addBook("Algorithms");
        centralLibrary.addBook("C++");
        centralLibrary.showAvailableBooks();

        centralLibrary.issueBook("C++");
        centralLibrary.showAvailableBooks();
        centralLibrary.returnBook("C++");
        centralLibrary.showAvailableBooks();
    }
}

```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Interpreted vs Compiled Languages!

Interpreter Vs Compiler :

The interpreter translates one statement at a time into machine code. On the other hand, the compiler scans the entire program and translates the whole of it into machine code

Interpreter :

1. one statement at a time
2. An interpreter is needed every time
3. Partial execution if an error occurs in the program.
4. Easy for programmers.

Compiler :

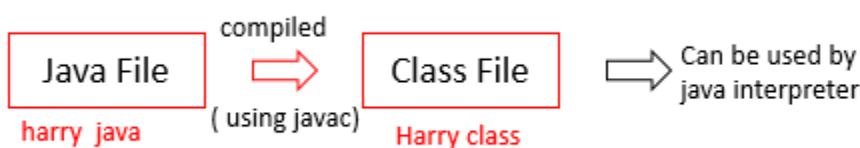
1. Entire program at a time
2. Once compiled, it is not needed
3. No execution if an error occurs
4. Usually not as easy as interpreted once

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Is Java interpreted or compiled?

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 education.
- Hence java is platform-independent (write once run everywhere).

Executing a java program

java Harry java - compiles

java Harry class - Interpreted

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

we can download a source code like VS code to compile & execute our java programs

Handwritten Notes: [Click to Download](#)

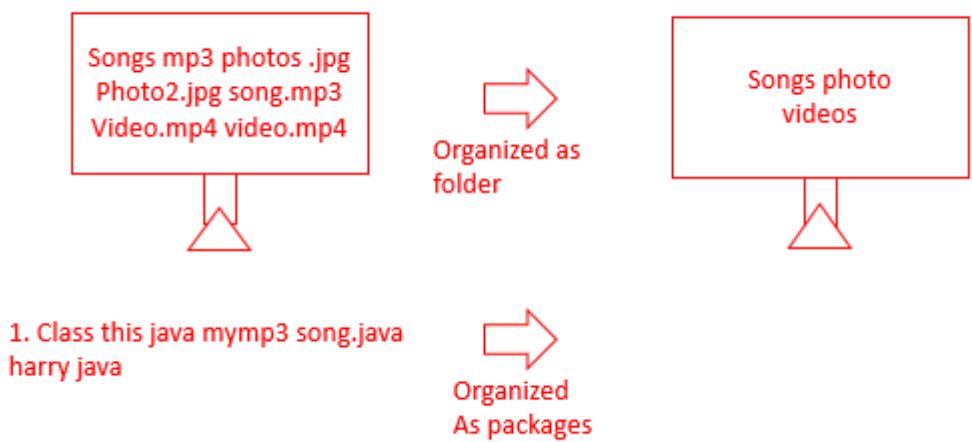
Ultimate Java Cheatsheet: [Click To Download](#)

Packages in Java

- A package is used to group related classes.
- packages help in avoiding name conflicts

There are two types of packages :

- Build-in packages - java API
- User-defined packages - Custom packages



Using a java package :

Import keyword is used to import packages in the java program. Example :

- `import java lang *` - import
- `import java string` - import string from java long
- `s= new java long string (" Harry ")` - use without importing

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Creating Packages in Java

How to create a package in Java :

```
javac -d Harry java
```

Copy

The above code creates a packages folder.

```
java Harry java
```

Copy

The above code creates **Harry** class.

- We can also create inner packages by adding packages inner as the package name.
- These packages once created can be used by other classes.

Code as described/written in the video :

```
package com.company;

import java.util.Scanner;
//import java.util.*;
public class cwh_65_packages {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // java.util.Scanner sc = new java.util.Scanner(System.in);
        int a = sc.nextInt();
        System.out.println("This is my scanner taking input as " + a);
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

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). See the table given below :

Access Modifier	within class	within package	outside package by subclass only	outside package
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Default	Y	Y	N	N
private	Y	N	N	N

Code as described/written in the video :

```
package com.company;

class C1{
    public int x = 5;
    protected int y =45;
    int z = 6;
    private int a = 78;
    public void meth1(){
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        System.out.println(a);
    }
}

public class cwh_66_access_modifiers {
    public static void main(String[] args) {
        C1 c = new C1();
        // c.meth1();
        System.out.println(c.x);
        System.out.println(c.y);
        System.out.println(c.z);
        // System.out.println(c.a);
    }
}
```



```
}  
}
```

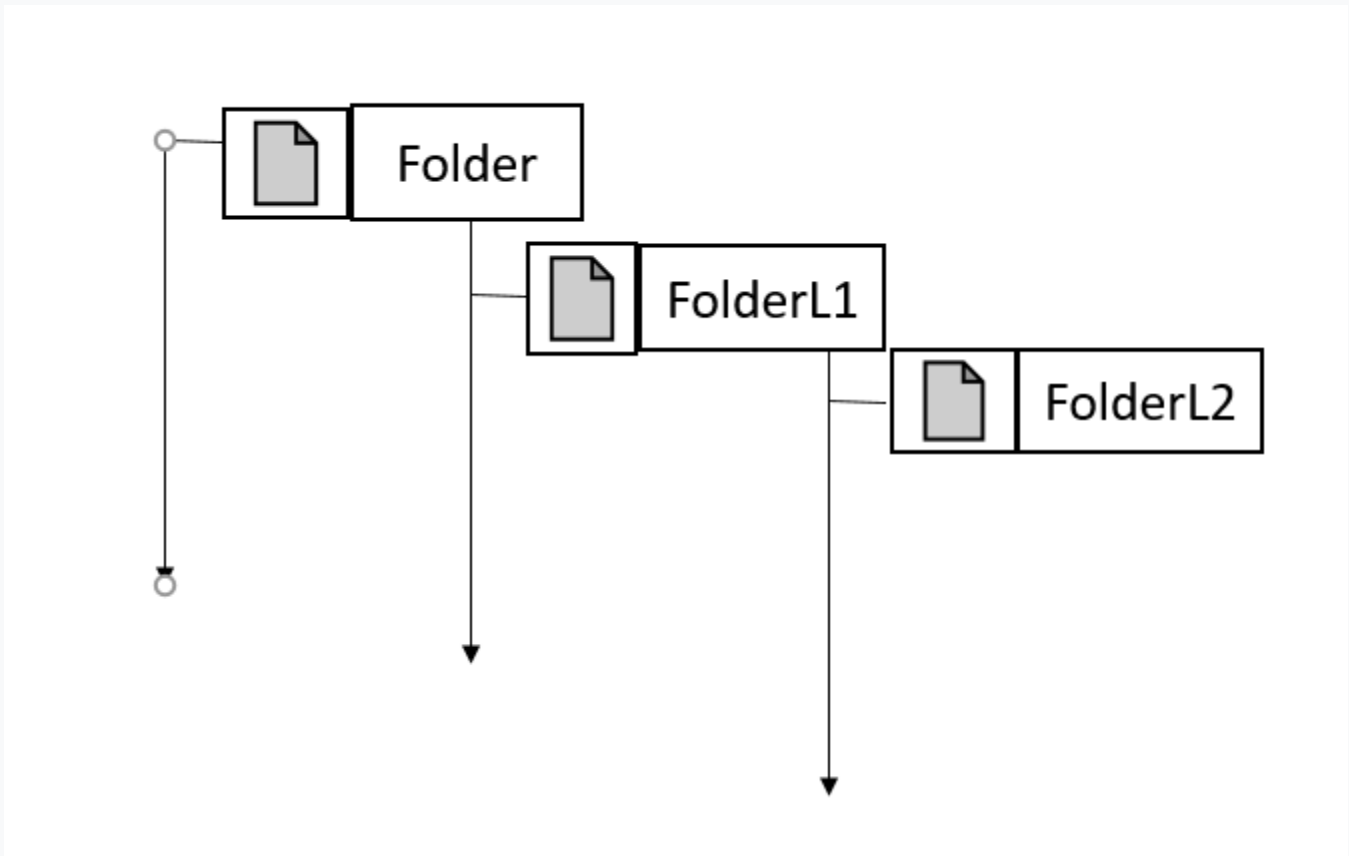
Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Practice Set on Java Package & Access Modifiers

1. Create three classes calculator , Sc calculator and Hybridcalculator and group them into a package
2. Use a build-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 properly from the subclass.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Exercise 5: Creating a Custom Package

You have to create a package named com.codewithharry.shape

This package should have individual classes for Rectangle, Square, Circle, Cylinder, Sphere

These classes should use inheritance to properly manage the code!

Include methods like volume, surface area and getters/setters for dimensions

```
package com.company;
```

```
/*
```

```
*** WRITE THIS CODE IN NOTEPAD ***
```

```
You have to create a package named com.codewithharry.shape
```

```
This package should have individual classes for Rectangle, Square, Circle, Cylinder, Sphere
```

```
These classes should use inheritance to properly manage the code!
```

```
Include methods like volume, surface area and getters/setters for dimensions
```

```
*/
```

```
public class cwh_68_ex5 {
```

```
    public static void main(String[] args) {
```

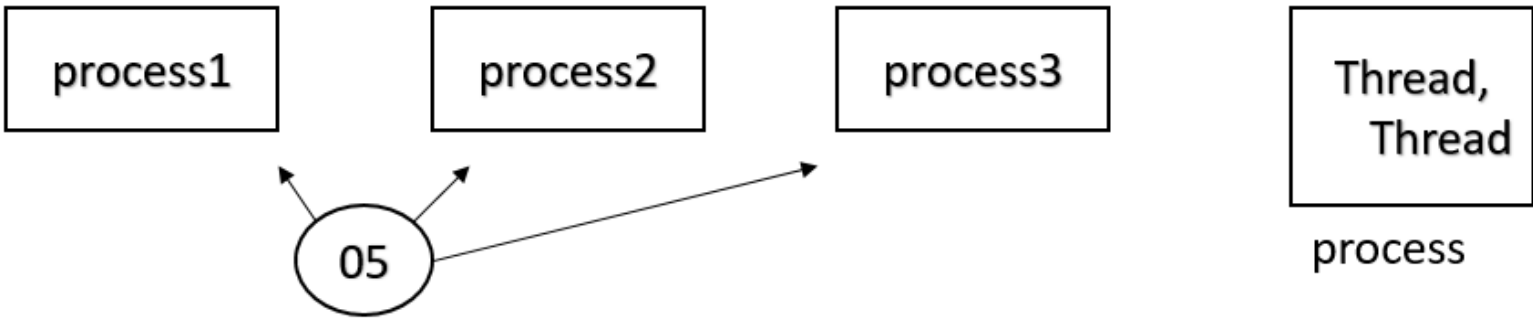
```
}  
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Multithreading in Java

Multiprocessing and multithreading both are used to achive multitasking



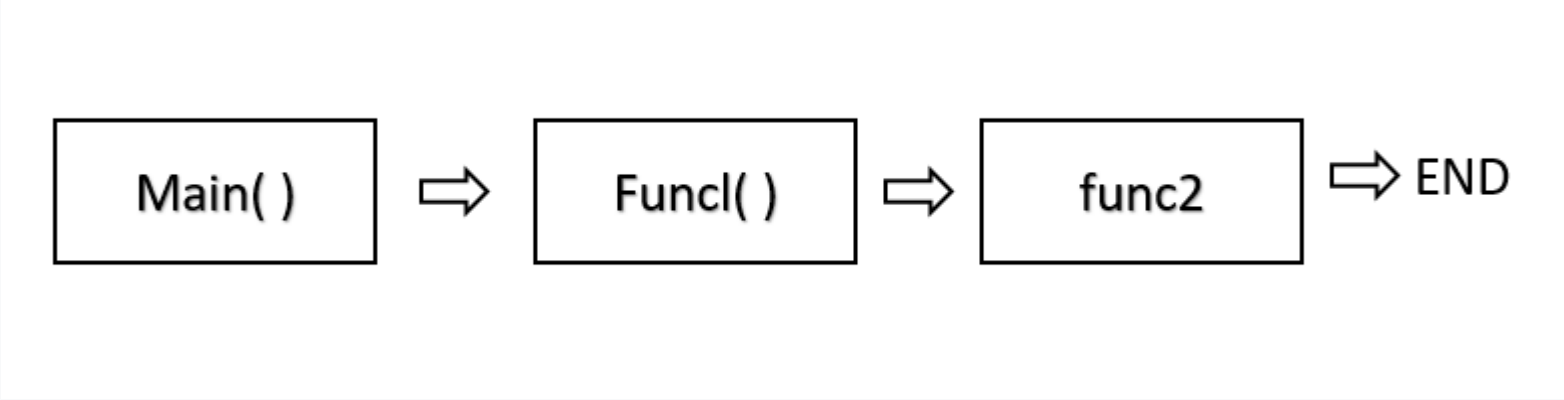
In a nut shell.....

- threds use shared memory area
- threads = faster content switching
- Athread is light-weight where a process is heavyweight

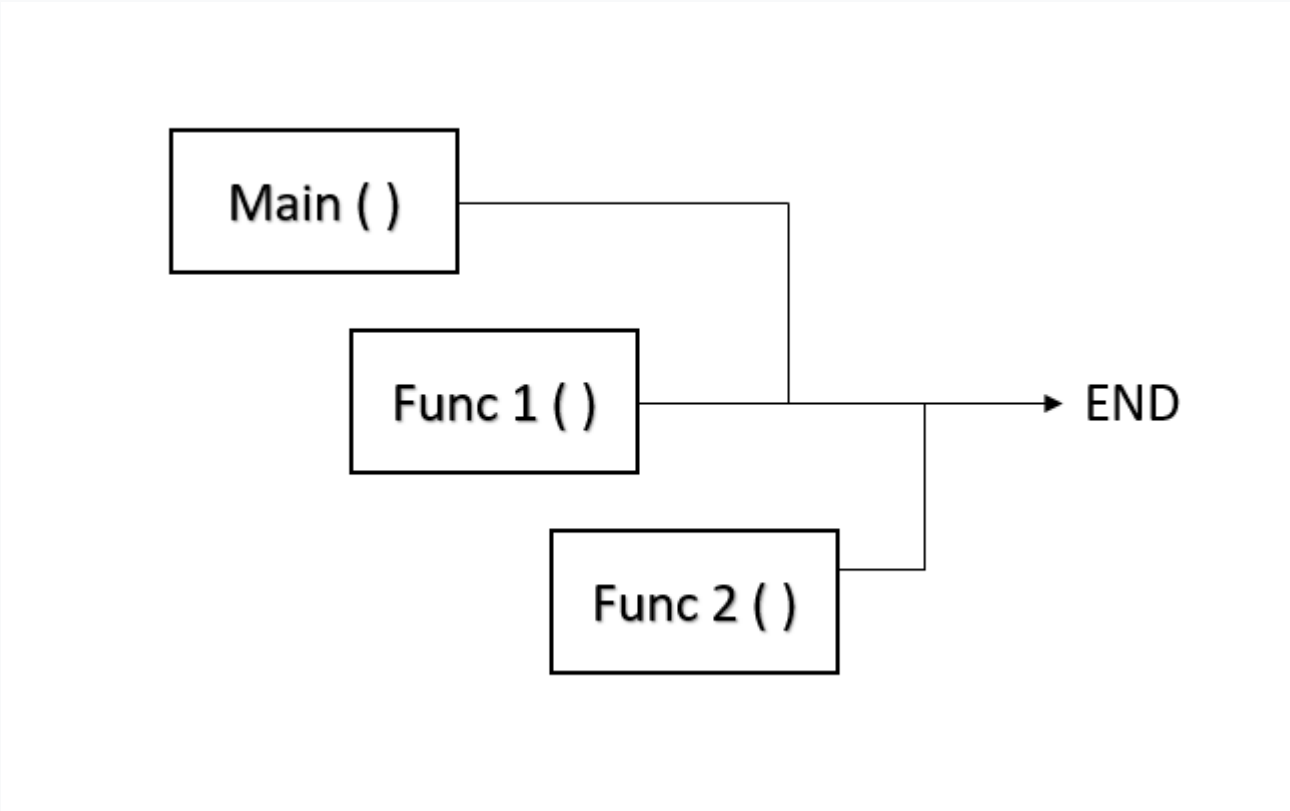
for example = Aword processor can have one thread running in foreground as an editor and another in the background auto saving the document !

Flow of control in java

- Without threading :



- With threading :



Creating a Threading

There are two ways to create a thread in java

1. By extending thread class
2. By implementing Runnable interface

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

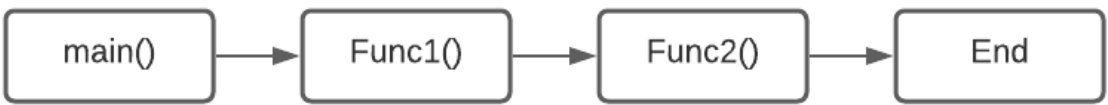
Creating a Thread by Extending Thread class

Multithreading In Java :

- Used to maximize the CPU utilization.
- We don't want our CPU to be in a free state; for example, Func1() comes into the memory and demands any input/output process. The CPU will need to wait for unit Func1() to complete its input/output operation in such a condition. But, while Func1() completes its I/O operation, the CPU is free and not executing any thread. So, the efficiency of the CPU is decreased in the absence of multithreading.
- In the case of multithreading, if a thread demands any I/O operation, then the CPU will let the thread perform its I/O operation, but it will start the execution of a new thread parallelly. So, in this case, two threads are executing at the same time.

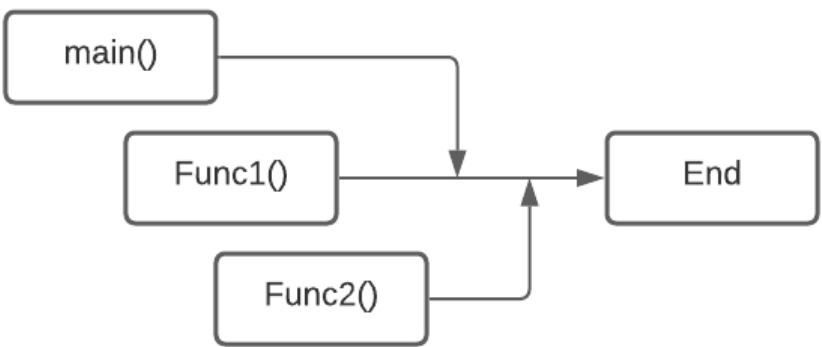
Flow Of Control In Java :

1. Without threading :



In the above image, you can see that Func1() and Func2() are called inside the main() function. But the execution of Func2() will start only after the completion of the Func1().

2. With threading :



Again, Func1() and Func2() are called inside the main function, but none of the two functions is waiting for the execution of the other function. Both the functions are getting executed concurrently.

Ways To Create A Thread In Java

1. By extending the thread class
2. By implementing Runnable interface

Let's see how we can create a thread by extending the thread class.

Extending Thread Class :

To create a thread using the thread class, we need to extend the thread class. Java's multithreading system is based on the thread class.

```
class MyThread extends Thread{
    @Override
    public void run(){
        //code that we want to get executed on running the thread
    }
}
```

```
}  
}
```

Copy

- In the above code, we're first inheriting the Thread class and then overriding the run() method.
- The code you want to execute on the thread's execution goes inside the run() method.

```
class MyThread extends Thread{  
    @Override  
    public void run(){  
        int i =0;  
        while(i<40000){  
            System.out.println("My Cooking Thread is Running");  
            System.out.println("I am happy!");  
            i++;  
        }  
    }  
}  
public class cwh_70 {  
    public static void main(String[] args) {  
        MyThread t1 = new MyThread();  
        t1.start();  
    }  
}
```

Copy

In order to execute the thread, the start() method is used. start() is called on the object of the MyThread class. It automatically calls the run() method, and a new stack is provided to the thread. So, that's how you easily create threads by extending the thread class in Java.

Code as described/written in the video :

```
package com.company;  
  
class MyThread1 extends Thread{  
    @Override  
    public void run(){  
        int i =0;  
        while(i<40000){  
            System.out.println("My Cooking Thread is Running");  
            System.out.println("I am happy!");  
            i++;  
        }  
    }  
}  
  
class MyThread2 extends Thread{  
    @Override  
    public void run(){  
        int i =0;  
        while(i<40000){  
            System.out.println("Thread 2 for Chatting with her");  
            System.out.println("I am sad!");  
            i++;  
        }  
    }  
}
```

```
public class cwh_70 {  
    public static void main(String[] args) {  
        MyThread1 t1 = new MyThread1();  
        MyThread2 t2 = new MyThread2();  
        t1.start();  
        t2.start();  
  
    }  
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Creating a Java Thread Using Runnable Interface

In the previous tutorial, I told you that there are two ways to create a thread in java :

1. By Extending Thread Class
2. By implementing Runnable interface

We've already seen [how to create a thread by extending the thread class](#). In this tutorial, we'll see how to create a Java thread by using a runnable interface.

Steps To Create A Java Thread Using Runnable Interface:

1. Create a class and implement the Runnable interface by using the implements keyword.
2. Override the run() method inside the implementer class.
3. Create an object of the implementer class in the main() method.
4. Instantiate the Thread class and pass the object to the Thread constructor.
5. Call start() on the thread. start() will call the run() method.

Example :

```
class t1 implements Runnable{  
    @Override  
    public void run(){  
        System.out.println("Thread is running");  
    }  
}  
  
public class ClassName{  
    public static void main(String[] args) {  
        t1 obj1 = new t1();  
        Thread t = new Thread(obj1);  
        t.start();  
    }  
}
```

Copy

1. class t1 is implementing the Runnable interface.
2. Overriding of the run() method is done inside the t1 class.
3. In the main() method, obj1, an object of the t1 class, is created.
4. The constructor of the Thread class accepts the Runnable instance as an argument, so obj1 is passed to the constructor of the Thread class.
5. Finally, the start() method is called on the thread that will call the run() method internally, and the thread's execution will begin.

Runnable Interface Vs Extending Thread Class :

Since we've discussed both the ways to create a thread in Java. There might be a question in your mind that should we use the Runnable interface or Thread class to implement a thread in Java. Let me

answer this question for you. The Runnable interface is preferred over extending the Thread class because of the following reasons :

1. As multiple inheritance is not supported in Java, it is impossible to extend the Thread class if your class had already extended some other class.
2. While implementing Runnable, we do not modify or change the thread's behavior.
3. More memory is required while extending the Thread class because each thread creates a unique object.
4. Less memory is required while implementing Runnable because multiple threads share the same object.

Code as described/written in the video :

[illegible]

[illegible]

[illegible]

[illegible]

```
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
System.out.println("I am a thread 2 not a threat 2");
}
}

public class cwh_71_runnable {
    public static void main(String[] args) {
        MyThreadRunnable1 bullet1 = new MyThreadRunnable1();
        Thread gun1 = new Thread(bullet1);

        MyThreadRunnable2 bullet2 = new MyThreadRunnable2();
        Thread gun2 = new Thread(bullet2);

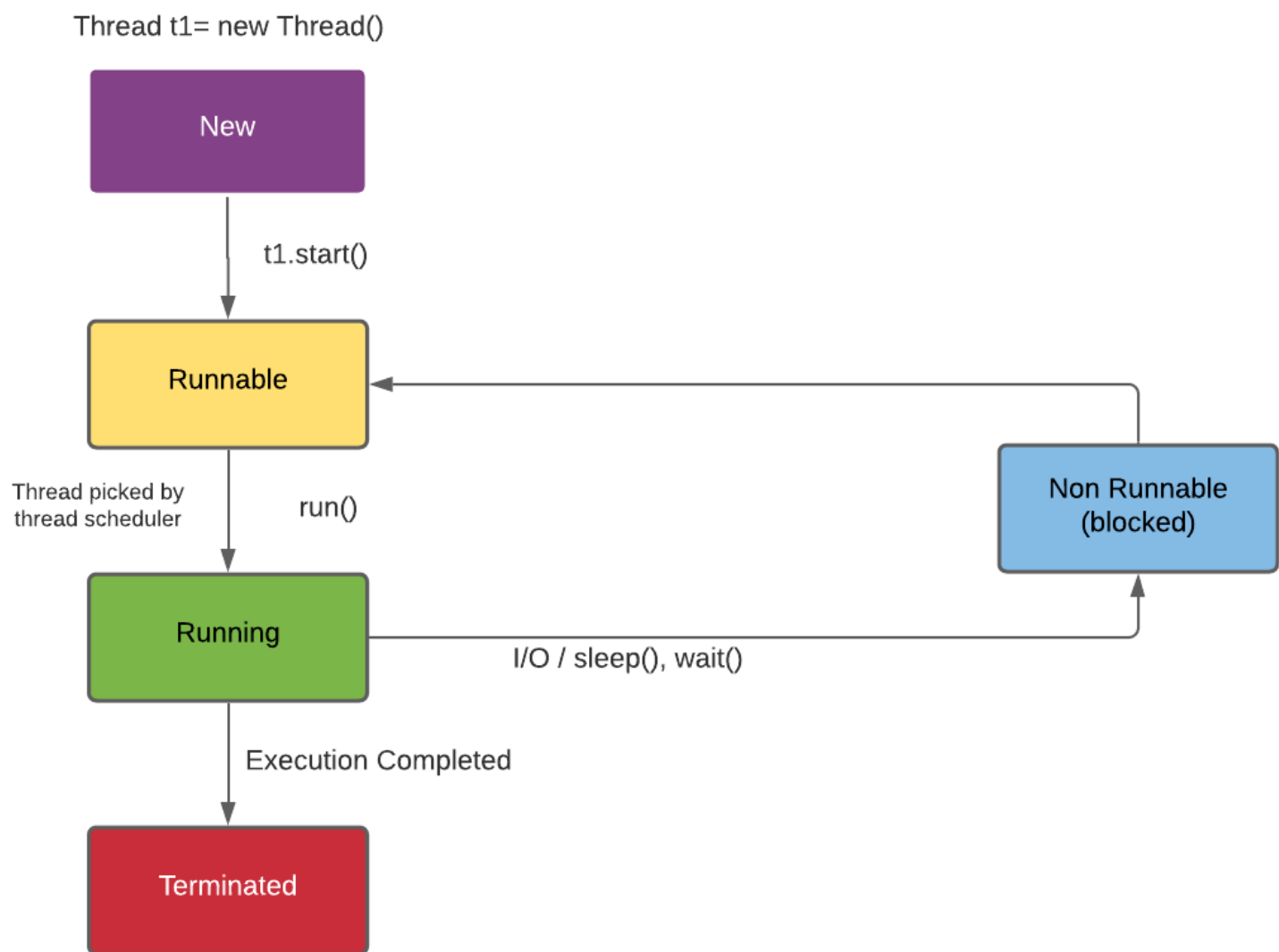
        gun1.start();
        gun2.start();
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Thread Life Cycle



1. New - Instance of thread created which is not yet started by involving start(). In this state, the thread is also known as the born thread.
2. Runnable - After invocation of start() & before it is selected to be run by the scheduler.
3. Running - After the thread scheduler has selected it.
4. Non-runnable - thread alive, not eligible to run.
5. Terminated - run() method has exited.

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Constructors from Thread class in Java

The Thread class

Below are the commonly used constructors of the thread class:

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

```
package com.company;

class MyThr extends Thread{
    public MyThr(String name){
        super(name);
    }
    public void run(){
        int i = 34;
        System.out.println("Thank you");
        // while(true){
        //     System.out.println("I am a thread");
        // }
```

```

    }
}
public class cwh_73_thread_constructor {
    public static void main(String[] args) {
        MyThr t1 = new MyThr("Harry");
        MyThr t2 = new MyThr("Ram Candr");
        t1.start();
        t2.start();
        System.out.println("The id of the thread t is " + t1.getId());
        System.out.println("The name of the thread t is " + t1.getName());
        System.out.println("The id of the thread t is " + t2.getId());
        System.out.println("The name of the thread t is " + t2.getName());

    }
}

```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Thread Priorities

In a Multithreading environment, all the threads which are ready and waiting to be executed are present in the Ready queue. The thread scheduler is responsible for assigning the processor to a thread. But the question is on what basis the thread scheduler decides that a particular thread will run before other threads. Here comes the concept of priority in the picture.

1. Every single thread created in Java has some priority associated with it.
2. The programmer can explicitly assign the priority to the thread. Otherwise, JVM automatically assigns priority while creating the thread.
3. In Java, we can specify the priority of each thread relative to other threads. Those threads having higher priorities get greater access to the available resources than lower priorities threads.
4. Thread scheduler will use priorities while allocating processor.
5. The valid range of thread priorities is 1 to 10 (but not 0 to 10), where 1 is the least priority, and 10 is the higher priority.
6. If there is more than one thread of the same priority in the queue, then the thread scheduler picks any one of them to execute.
7. The following static final integer constants are defined in the Thread class:
 - **MIN_PRIORITY**: Minimum priority that a thread can have. Value is 1.
 - **NORM_PRIORITY**: This is the default priority automatically assigned by JVM to a thread if a programmer does not explicitly set the priority of that thread. Value is 5.
 - **MAX_PRIORITY**: Maximum priority that a thread can have. Value is 10.

Priority Methods In Java :

1. setPriority():

- This method is used to set the priority of a thread. `IllegalArgumentException` is thrown if the priority given by the user is out of the range [1,10].

Syntax :

```
public final void setPriority(int x)    // x is the priority [1,10] that is to be set for the thread.
```

Copy

2. getPriority():

- This method is used to display the priority of a given thread.

Syntax :

```
t1.getPriority() // Will return the priority of the t1 thread.
```

Copy

Example :

```
class cwh_Priority extends Thread{
    public void run(){
        System.out.println("I'm thread : "+Thread.currentThread().getName());
        System.out.println("I'm thread :"+Thread.currentThread().getPriority());

    }
    public static void main(String args[]){
        cwh_Priority t1=new cwh_Priority();
        cwh_Priority t2= new cwh_Priority();
        t1.setPriority(Thread.MIN_PRIORITY); // setting priority of t1 thread to MIN_PRIORITY (1)
        t2.setPriority(Thread.MAX_PRIORITY); // setting priority of t2 thread to MAX_PRIORITY (10)
        t1.start();
        t2.start();

    }
}
```

Copy

Output :

```
I'm thread : Thread-0
I'm thread : Thread-1
I'm thread :10
I'm thread :1
```

Copy

Code as described/written in the video :

```
package com.company;

class MyThr1 extends Thread{
    public MyThr1(String name){
        super(name);
    }
    public void run(){
        int i = 34;

        while(true){
//            System.out.println("I am a thread");
            System.out.println("Thank you: " + this.getName());
        }

    }
}

public class cwh_74_thread_priorities {
    public static void main(String[] args) {
        // Ready Queue: T1 T2 T3 T4 T5
        MyThr1 t1 = new MyThr1("Harry1");
        MyThr1 t2 = new MyThr1("Harry2");
        MyThr1 t3 = new MyThr1("Harry3");
        MyThr1 t4 = new MyThr1("Harry4");
```

```
MyThr1 t5 = new MyThr1("Harry5 (most Important)");
t5.setPriority(Thread.MAX_PRIORITY);
t1.setPriority(Thread.MIN_PRIORITY);
t2.setPriority(Thread.MIN_PRIORITY);
t3.setPriority(Thread.MIN_PRIORITY);
t4.setPriority(Thread.MIN_PRIORITY);
t5.setPriority(Thread.MIN_PRIORITY);

t1.start();
t2.start();
t3.start();
t4.start();
t5.start();

}
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Thread Methods

Join() method In Java :

- The join() method in Java allows one thread to wait until the execution of some other specified thread is completed.
- If t is a Thread object whose thread is currently executing, then t.join() causes the current thread to pause execution until t's thread terminates.
- Join() method puts the current thread on wait until the thread on which it is called is dead.

Syntax :

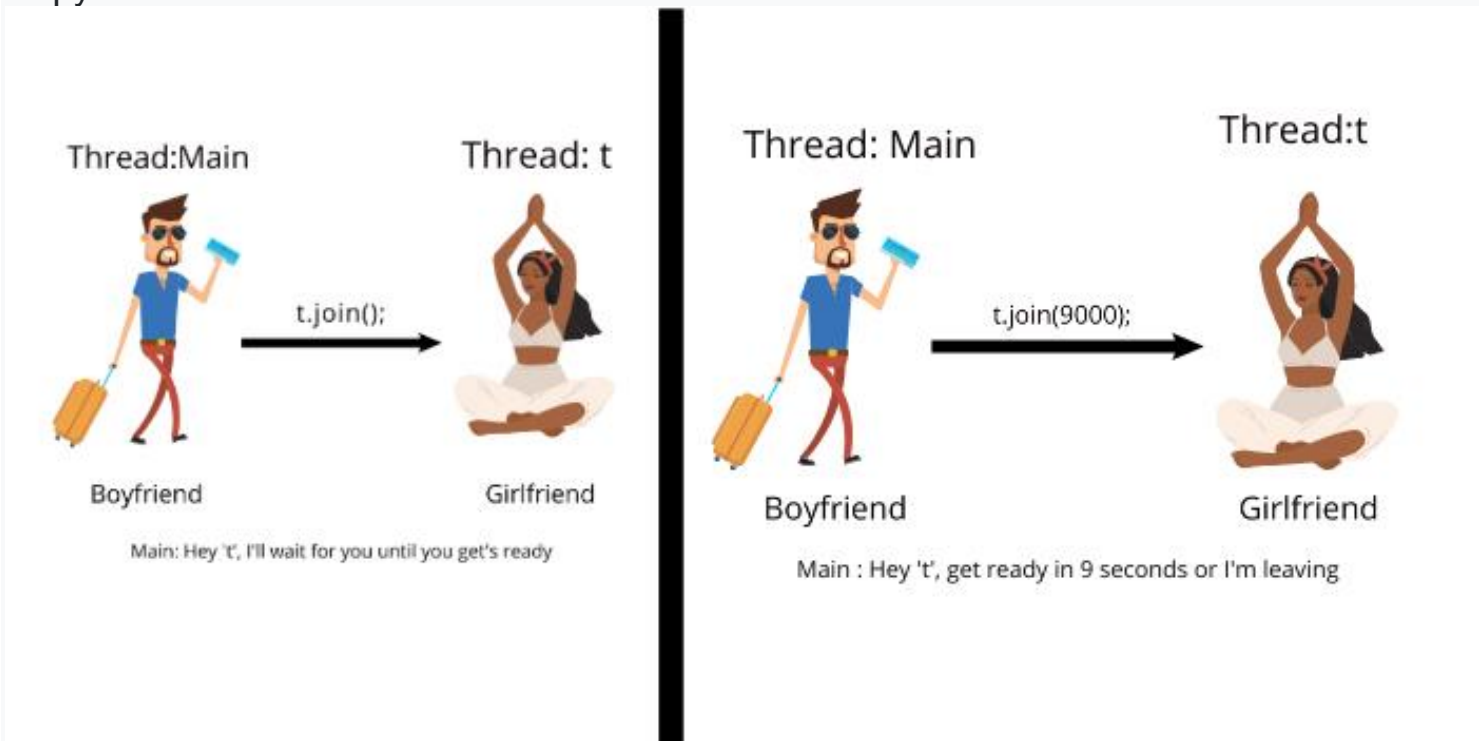
```
public final void join()
```

Copy

You can also specify the time for which you need to wait for the execution of a particular thread by using the Join() method. Syntax :

```
public final void join(long millis)
```

Copy



Sleep() Method :

- The sleep() method in Java is useful to put a thread to sleep for a specified amount of time.
- When we put a thread to sleep, the thread scheduler picks and executes another thread in the queue.

- Sleep() method returns void.
- sleep() method can be used for any thread, including the main() thread also.

Syntax :

- public static void sleep(long milliseconds)throws InterruptedException
- public static void sleep(long milliseconds, int nanos)throws InterruptedException

Parameters Passed To Sleep() Method :

1. long millisecond: Time in milliseconds for which thread will sleep.
2. nanos : Ranges from [0,999999]. Additional time in nanoseconds.

Example :

```
import java.io.*;
import java.lang.Thread;
public class cwh {
    public static void main(String[] args)
    {
        try {
            for (int i = 1; i <=5; i++) {
                Thread.sleep(2000);
                System.out.println(i);
            }
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Copy

In the above example, the main() method will be put to sleep for 2 seconds every time the for loop executes.

Output :

```
1
2
3
4
5
```

Copy

Interrupt() method :

- A thread in a sleeping or waiting state can be interrupted by another thread with the help of the interrupt() method of the Thread class.
- The interrupt() method throws InterruptedException.
- The interrupt() method will not throw the InterruptedException if the thread is not in the sleeping/blocked state, but the interrupt flag will be changed to true.

Syntax :

```
Public void interrupt()
```

Copy

Different scenarios where Interrupt() method can be used:

Case 1: Interrupting a thread that doesn't stop working :

```
class CWH1 extends Thread{
    public void run(){
```



```

        try {
            for (int i=0;i<5;i++){
                System.out.println("Child Thread");
                Thread.sleep(4000); /* Child thread is put to sleep for 4000ms. As soon as child
thread goes to sleep main thread interrupts it. And, InterruptedException is generated which is
handled by the catch block. */

            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child Thread Interrupted");
        }
        System.out.println("Thread is running");

    }
}

public class CWH extends Thread{
    public static void main(String[] args) {
        CWH1 t= new CWH1();
        t.start();
        t.interrupt();
        System.out.println("Main Thread");

    }
}

```

Copy

In the above code, the for loop runs for the first time, but the child thread is put to sleep after that. So, the main() method interrupts the child thread, and InterruptedException is generated. Here, the child thread comes out of the sleeping state, but it does not stop working.

Output :

```

Main Thread
Child Thread
Child Thread Interrupted
Thread is running

```

Copy

Case 2: Interrupting a thread that works normally :

```

class CWH1 extends Thread{
    public void run(){
        for (int i=0;i<10;++i){
            System.out.println(i);
        }
    }
}

public class CWH extends Thread{
    public static void main(String[] args) {
        CWH1 t= new CWH1();
        t.start();
        t.interrupt();
        System.out.println("Main Thread");
    }
}

```



```
}  
}
```

Copy

Here the thread works normally because no exception occurred during the thread's execution, so the `interrupt()` only sets the value of the thread flag to true.

Output :

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Copy

Code as described/written in the video :

```
package com.company;  
  
class MyNewThr1 extends Thread{  
    public void run(){  
        int i = 0;  
        while(true){  
//            System.out.println("I am a thread");  
            System.out.println("Thank you: ");  
            try {  
                Thread.sleep(455);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            i++;  
        }  
    }  
}  
  
class MyNewThr2 extends Thread{  
  
    public void run(){  
        while(true){  
//            System.out.println("I am a thread");  
            System.out.println("My Thank you: ");  
        }  
    }  
}  
  
public class cwh_75_thread_methods {  
    public static void main(String[] args){  
        MyNewThr1 t1 = new MyNewThr1();  
        MyNewThr2 t2 = new MyNewThr2();  
        t1.start();  
//        try{
```

```
//          t1.join();
//      }
//      catch(Exception e){
//          System.out.println(e);
//      }

      t2.start();

  }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Java Tutorial: Practice Questions on Thread

Question 1: Write a program to print "Good morning" and "Welcome" continuously on the screen in Java using threads.

Answer 1:

```
import java.util.ArrayList;

class Thread1 extends Thread{
    public void run(){
        while (true){
            System.out.println("Welcome");
        }
    }
}

class Thread2 extends Thread {
    public void run() {
        while (true) {
            System.out.println("Good morning");
        }
    }
}

public class CWH {
    public static void main(String[] args) {
        Thread1 t1= new Thread1();
        Thread2 t2= new Thread2();
        t1.start();
        t2.start();
    }
}
```

Copy

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

Answer 2:

```
import java.util.ArrayList;

class Thread1 extends Thread{
    public void run(){
        try {
```

```

        Thread.sleep(200);
    }

    catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("Welcome");
}
}

class Thread2 extends Thread {
    public void run() {
        System.out.println("Good morning");
    }
}

public class CWH {
    public static void main(String[] args) {
        Thread1 t1= new Thread1();
        Thread2 t2= new Thread2();
        t1.start();
        t2.start();
    }
}

```

Copy

Question 3: Demonstrate gerPriority() and setPriority() methods in Java threads.

Answer 3:

```

import java.util.ArrayList;

class Thread1 extends Thread{
    public void run(){
        System.out.println("Welcome");
    }
}

class Thread2 extends Thread {
    public void run() {
        System.out.println("Good morning");
    }
}

public class CWH {
    public static void main(String[] args) {
        Thread1 t1= new Thread1();
        Thread2 t2= new Thread2();
        t1.start();
        t2.start();
        t1.setPriority(5);
        t2.setPriority(1);

        System.out.println(t1.getPriority());
        System.out.println(t2.getPriority());
    }
}

```

Copy

Question 4: How do you get the state of a given thread in Java?

Answer 4: **getState()** method is used to get the state of a given thread in Java.

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

Answer 5: `currentThread()` method is used to reference the current thread in Java.

Code as described/written in the video :

```
package com.company;

class Practice13 extends Thread{
    public void run(){
        while(true){
            System.out.println("Good Morning!");
        }
    }
}

class Practice13b extends Thread{
    public void run(){
//        while(false){
//            try {
//                Thread.sleep(200);
//            }
//            catch (Exception e){
//                System.out.println(e);
//            }
//            System.out.println("Welcome");
//        }
    }
}

public class cwh_76_practice13 {
    public static void main(String[] args) {
        Practice13 p1 = new Practice13();
        Practice13b p2 = new Practice13b();
        // p1.setPriority(6);
        // p2.setPriority(9);
        System.out.println(p1.getPriority());
        System.out.println(p2.getPriority());
        System.out.println(p2.getState());
//        p1.start();
        p2.start();
        System.out.println(p2.getState());
        System.out.println(Thread.currentThread().getState());
    }
}
```

Copy

Handwritten Notes: [Click to Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Exercise 5: Solution & Shoutouts!

You have to create a package named `com.codewithharry.shape`
This package should have individual classes for Rectangle, Square, Circle, Cylinder, Sphere
These classes should use inheritance to properly manage the code!
Include methods like volume, surface area and getters/setters for dimensions

Errors & Exception in Java

No matter how smart we are, errors are our constant comparisons.

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 errors!
a =    a + 3;
d = 4; // Variable not declared, syntax errors
```

Copy

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 'S' + 8 to a program that adds 2 numbers.

Syntax errors and logical errors are encountered by the programmers, whereas Run-time errors are encountered by the users.

```
package com.company;

public class cwh_78_errors {
    public static void main(String[] args) {
        int a = 5;
        int b = 9;
        System.out.println(a+b);
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Syntax Errors, Runtime Errors & Logical Errors in Java (Demo)

```
package com.company;

import java.util.Scanner;
```

```

public class cwh_79_errorsdemo {
    public static void main(String[] args) {
        // SYNTAX ERROR DEMO
        // int a = 0 // Error: no semicolon!
        // b = 8; // Error: b not declared!

        // LOGICAL ERROR DEMO
        // Write a program to print all prime numbers between 1 to 10
        System.out.println(2);
        for (int i=1; i<5; i++){
            System.out.println(2*i+1);
        }

        // RUNTIME ERROR
        int k;
        Scanner sc = new Scanner(System.in);
        k = sc.nextInt();
        System.out.println("Integer part of 1000 divided by k is "+ 1000/k);

    }
}

```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Exceptions & Try-Catch Block in Java

Exceptions in Java

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

There are mainly two types of exceptions in java:

- 1) Checked exceptions - compile-time exceptions (Handle by the compiler)
- 2) Unchecked exceptions - Runtime exceptions

Commonly Occurring Exceptions

Following are few commonly occurring exceptions in java:

- 1) Null pointer exception
- 2) Arithmetic Exception
- 3) Array Index out of Bound exception
- 4) Illegal Argument Exception
- 5) Number Format Exception

```

package com.company;

public class cwh_80_try {
    public static void main(String[] args) {
        int a = 6000;
        int b = 0;
        // Without Try:
    }
}

```

```
//      int c = a / b;
//      System.out.println("The result is " + c);
// With Try:
try {
    int c = a / b;
    System.out.println("The result is " + c);
}
catch(Exception e) {
    System.out.println("We failed to divide. Reason: ");
    System.out.println(e);
}
System.out.println("End of the program");
}
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Handling Specific Exceptions in Java

Handling specific Exceptions

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

```
try (

    // code

)

Catch (IOException e) - Handle all Exceptions of type IOException

    // code

)

Catch (Exception e) - Handle all Exceptions of type Arithmetic Exception

    // code

)

Catch (Exception e) - Handle all other Exceptions

    // code

)
```

Copy

```
package com.company;

import java.util.Scanner;

public class cwh_81 {
    public static void main(String[] args) {
        int [] marks = new int[3];
    }
}
```

```

marks[0] = 7;
marks[1] = 56;
marks[2] = 6;
Scanner sc = new Scanner(System.in);
System.out.println("Enter the array index");
int ind = sc.nextInt();

System.out.println("Enter the number you want to divide the value with");
int number = sc.nextInt();
try{
    System.out.println("The value at array index entered is: " + marks[ind]);
    System.out.println("The value of array-value/number is: " + marks[ind]/number);
}
catch (ArithmeticException e){
    System.out.println("ArithmeticException occurred!");
    System.out.println(e);
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("ArrayIndexOutOfBoundsException occurred!");
    System.out.println(e);
}
catch (Exception e){
    System.out.println("Some other exception occurred!");
    System.out.println(e);
}
}
}

```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Nested Try-Catch in Java

Nested try - catch

We can nest multiple try - catch blocks as follows:

```

package com.company;

import java.util.Scanner;

public class cwh_82_nested_try_catch {
    public static void main(String[] args) {
        int [] marks = new int[3];
        marks[0] = 7;
        marks[1] = 56;
        marks[2] = 6;
        Scanner sc = new Scanner(System.in);
        boolean flag = true;
        while(flag) {
            System.out.println("Enter the value of index");
            int ind = sc.nextInt();
            try {
                System.out.println("Welcome to video no 82");
                try {

```



```

        System.out.println(marks[ind]);
        flag = false;
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Sorry this index does not exist");
        System.out.println("Exception in level 2");
    }
} catch (Exception e) {
    System.out.println("Exception in level 1");
}
}
System.out.println("Thanks for using this program");
}
}

```

Copy

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

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

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

The Exception class in Java

We can write our custom Exceptions using the Exception class in java.

```

public class MyException extends Exception {
    // Overridden methods
}

```

Copy

The Exception class has the 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

```

package com.company;

import java.util.Scanner;

class MyException extends Exception{
    @Override
    public String toString() {
        return "I am toString()";
    }

    @Override
    public String getMessage() {
        return "I am getMessage()";
    }
}

class MaxAgeException extends Exception{
    @Override
    public String toString() {
        return "Age cannot be greater than 125";
    }
}

```

```

    }

    @Override
    public String getMessage() {
        return "Make sure that the value of age entered is correct";
    }
}

public class cwh_83_exception_class {
    public static void main(String[] args) {
        int a;
        Scanner sc = new Scanner(System.in);
        a = sc.nextInt();
        if (a<9){
            try{
                // throw new MyException();
                throw new ArithmeticException("This is an exception");
            }
            catch (Exception e){
                System.out.println(e.getMessage());
                System.out.println(e.toString());
                e.printStackTrace();
                System.out.println("Finished");
            }
            System.out.println("Yes Finished");
        }
    }
}

```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Throw vs Throws in Java

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 ;
}

```

Copy

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

```

throw new My Exception ( "Exception throw" );

```

Copy

The throw Keyword

Throws 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
```

```
}
```

Copy

```
package com.company;
```

```
class NegativeRadiusException extends Exception{
```

```
    @Override
```

```
    public String toString() {
```

```
        return "Radius cannot be negative!";
```

```
    }
```

```
    @Override
```

```
    public String getMessage() {
```

```
        return "Radius cannot be negative!";
```

```
    }
```

```
}
```

```
public class cwh_84_throw_throws {
```

```
    public static double area(int r) throws NegativeRadiusException{
```

```
        if (r<0){
```

```
            throw new NegativeRadiusException();
```

```
        }
```

```
        double result = Math.PI * r * r;
```

```
        return result;
```

```
    }
```

```
    public static int divide(int a, int b) throws ArithmeticException{
```

```
        // Made By Harry
```

```
        int result = a/b;
```

```
        return result;
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        // Shivam - uses divide function created by Harry
```

```
        try{
```

```
//            int c = divide(6, 0);
```

```
//            System.out.println(c);
```

```
            double ar = area(6);
```

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

```
        }
```

```
        catch(Exception e){
```

```
            System.out.println("Exception");
```

```
        }
```

```
    }
```

```
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Finally Block in Java & Why is it needed!

Java finally block

Finally block contains the code which is always executed whether the exception is handled or not.

It is used to exception is handled or not.

It is used to execute code containing instructions to release the system resources, close a connection etc.

Sample Code Demonstrating Finally

```
package com.company;

public class cwh_85_finally {
    public static int greet(){
        try{
            int a = 50;
            int b = 10;
            int c = a/b;
            return c;
        }
        catch(Exception e){
            System.out.println(e);
        }
        finally {
            System.out.println("Cleaning up resources...This is the end of this function");
        }
        return -1;
    }

    public static void main(String[] args) {
        int k = greet();
        System.out.println(k);
        int a = 7;
        int b = 9;
        while(true){
            try{
                System.out.println(a/b);
            }
            catch (Exception e){
                System.out.println(e);
                break;
            }
            finally{
                System.out.println("I am finally for value of b = " + b);
            }
            b--;
        }

        try{
            System.out.println(50/3);
        }
        finally {
            System.out.println("Yes this is finally");
        }
    }
}
```

Copy

Handwritten Notes: [Click To Download](#)

Ultimate Java Cheatsheet: [Click To Download](#)

Practice Set on Errors & Exceptions

Chapter- 14 - Practice set

- 1) Write a java program to demonstrate syntax, logical 2 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 given. If max retries exceed 5 print "errors".
- 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.

```
package com.company;

import java.util.Scanner;

public class cwh_86_ps14 {
    public static void main(String[] args) {
        // Problem 1
        // Syntax Error - int a = 7
        int age = 78;
        int year_born = 2000-78; // Logical error
        //      System.out.println(6/0);

        // Problem 2
        try{
            int a = 666/0;
        }
        catch (IllegalArgumentException e){
            System.out.println("HeHe");
        }
        catch (ArithmeticException e){
            System.out.println("Haha");
        }
        }

        // Problem 3
        boolean flag = true;
        int [] marks = new int[3];
        marks[0] = 7;
        marks[1] = 56;
        marks[2] = 6;
        Scanner Sc = new Scanner(System.in);
        int index;
        int i = 0;
        while(flag && i<5){
            try {
                System.out.println("Enter the value of index");
                index = Sc.nextInt();
                System.out.println("The value of marks[index] is " + marks[index]);
                break;
            }
            catch (Exception e) {
                System.out.println("Invalid input. Please enter a valid index between 0 and 4.");
                i++;
            }
        }
    }
}
```

```

        }
        catch (Exception e) {
            System.out.println("Invalid Index");
            i++;
        }
    }
    if(i>=5){
        System.out.println("Error");
    }

}
}

```

Copy

[Download Ch 14 Practice Set pdf here](#)

Java Exercise 6: Custom Calculator | Java Practice Question

Exercise 6: You have to create a custom calculator with following operations:

1. + -> Addition
2. - -> Subtraction
3. * -> Multiplication
4. / -> Division

which throws the following exceptions:

1. Invalid input Exception ex: 8 & 9
2. Cannot divide by 0 Exception
3. Max Input Exception if any of the inputs is greater than 100000
4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater than 7000

```

package com.company;

public class cwh_87_ex6 {
    public static void main(String[] args) {
        /*
        Exercise 6: You have to create a custom calculator with following operations:
        1. + -> Addition
        2. - -> Subtraction
        3. * -> Multiplication
        4. / -> Division
        which throws the following exceptions:
        1. Invalid input Exception ex: 8 & 9
        2. Cannot divide by 0 Exception
        3. Max Input Exception if any of the inputs is greater than 100000
        4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater
        than 7000
        */
    }
}

```

Copy

Java Collections Framework

Collection Framework

A collection represents a group of object Java collections provide classes and Interfaces for us to be able to write code 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 elements in Array?
- Apply certain operations to change this array?

```
package com.company;

import java.util.ArrayList;
import java.util.Set;
import java.util.TreeSet;

public class cwh_89_collections {
    public static void main(String[] args) {
        // ArrayList
        // Set
        // TreeSet
    }
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Collections Hierarchy in Java

How are collections available

Collections in java are available as class and interfaces Following are few commonly used collections in java :

- ArrayList -> For variable size collections
- Set -> For distinct collection
- Stack-> A LIFO data structure
- HashMap -> For strong key - value pairs

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

```
package com.company;

import java.util.ArrayList;
import java.util.Set;
import java.util.TreeSet;

public class cwh_89_collections {
    public static void main(String[] args) {
        // ArrayList
        // Set
        // TreeSet
    }
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

How to View Java Documentation (Correct Way)

Ultimate Java Cheatsheet: [Click To Download](#)

ArrayList in Java: Demo & Methods

- The ArrayList class is the dynamic array found in the java.util package.
- The size of the normal array can not be changed, but ArrayList provides us the ability to increase or decrease the size.
- ArrayList is slower than the standard array, but it helps us to manipulate the data easily.

How to declare an ArrayList :

```
ArrayList<Integer> l1 = new ArrayList<>(); /*Creates an ArrayList object of  
integer type */
```

Copy

Performing various operations on ArrayList :

ArrayList comes with a number of methods that can be used to manipulate the data of the ArrayList. Let's take a look at some of the important methods of ArrayList :

1. Adding an element :

- add() method is used to insert an element in the ArrayList.
- add(Object): Inserts an element at the end of the ArrayList.
- add(Index,Object) : Inserts an element at the given index.

Example :

```
import java.util.*;  
public class CWH extends Thread{  
    public static void main(String[] args) {  
  
        ArrayList<Integer> l1 = new ArrayList<>();  
        l1.add(1);  
        l1.add(2);  
        l1.add(3);  
        l1.add(4);  
        l1.add(6);  
        l1.add(5,5); // inserts 5 at the 5th index in l1  
  
        System.out.println(l1);  
    }  
}
```

Copy

Output :

```
[1, 2, 3, 4, 6, 5]
```

Copy

• Removing an Element :

- remove() method is used to delete or remove an element at a given index from the ArrayList.

Syntax :

```
l1.remove(index number)
```

Copy

Example :

```
import java.util.*;  
public class CWH extends Thread{  
    public static void main(String[] args) {
```



```
        ArrayList<Integer> l1 = new ArrayList<>();
        l1.add(1);
        l1.add(2);
        l1.add(3);
        l1.add(4);
        l1.add(6);
        l1.add(5,5); // inserts 5 at the 5th index in l1

        System.out.println("Array list before : "+ l1);

        l1.remove(0);
        System.out.println("ArrayList after removing the value at index 0 : " + l1);

    }
}
```

Copy

Output :

```
Array list before : [1, 2, 3, 4, 6, 5]
ArrayList after removing the value at index 0 : [2, 3, 4, 6, 5]
```

Copy

- Checking if an ArrayList contains a specific value or not :

- contains() method is used to check if an ArrayList contains a specified element or not. This method returns the boolean value.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayList<Integer> l1 = new ArrayList<>();
        l1.add(1);
        l1.add(2);
        l1.add(3);
        l1.add(4);
        l1.add(5);
        l1.add(6);

        System.out.println("Array list : "+ l1);
        System.out.println("L1 list contains 7 : " + l1.contains(7));
        System.out.println("L1 list contains 4 : " + l1.contains(4));

    }
}
```

Copy

Output :

```
Array list before : [1, 2, 3, 4, 5, 6]
L1 list contains 7 : false
L1 list contains 4 : true
```

Copy

- Merging two ArrayLists :

- The elements of an ArrayList can be merged into another Arraylist with the help of the addAll() method.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayList<Integer> l1 = new ArrayList<>();
        ArrayList<Integer> l2 = new ArrayList<>();

        l1.add(1);
        l1.add(2);
        l1.add(3);
        l1.add(4);
        l1.add(5);
        l1.add(6);

        l2.add(11);
        l2.add(12);
        l2.add(13);
        l2.add(14);

        System.out.println("L1 Array list : "+ l1);
        System.out.println("L2 Array list : " +l2);
        l1.addAll(l2);
        System.out.println("L1 Array list after merging: "+ l1);
        System.out.println("L2 Array list : " +l2);

    }
}
```

Copy

Output :

```
L1 Array list : [1, 2, 3, 4, 5, 6]
L2 Array list : [11, 12, 13, 14]
L1 Array list after merging: [1, 2, 3, 4, 5, 6, 11, 12, 13, 14]
L2 Array list : [11, 12, 13, 14]
```

Copy

- You can add the elements of l2 at the starting of l1 by typing :

```
l1.addAll(0,l2);
```

Copy

- Finding the first occurrence of a specified number in the ArrayList :

- Indexof() method prints the index of the first occurrence of a particular number. Returns -1 if the element is not present in the ArrayList.

Example :

```
import java.util.*;
public class CWH extends Thread{
```

```
public static void main(String[] args) {

    ArrayList<Integer> l1 = new ArrayList<>();

    l1.add(1);
    l1.add(2);
    l1.add(3);
    l1.add(4);
    l1.add(5);
    l1.add(6);
    l1.add(3);

    System.out.println("L1 Array list : "+ l1);
    System.out.println("The first occurrence of 3 in l1 is at index : " +
l1.indexOf(3));

}
```

Copy

Output :

```
L1 Array list : [1, 2, 3, 4, 5, 6, 3]
The first occurrence of 3 in l1 is at index : 2
```

Copy

Similarly, you can also find the index of the last occurrence of an element with the help of the `lastIndexOf()` method.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayList<Integer> l1 = new ArrayList<>();

        l1.add(1);
        l1.add(2);
        l1.add(3);
        l1.add(4);
        l1.add(5);
        l1.add(3);
        l1.add(6);
        l1.add(3);

        System.out.println("L1 Array list : "+ l1);
        System.out.println("The last occurrence of 3 in l1 is at index : " +
l1.lastIndexOf(3));

    }
}
```

Copy

Output :

```
L1 Array list : [1, 2, 3, 4, 5, 3, 6, 3]
```

The last occurrence of 3 in l1 is at index : 7

Copy

Code as described/written in the video :

```
package com.company;

import java.util.*;

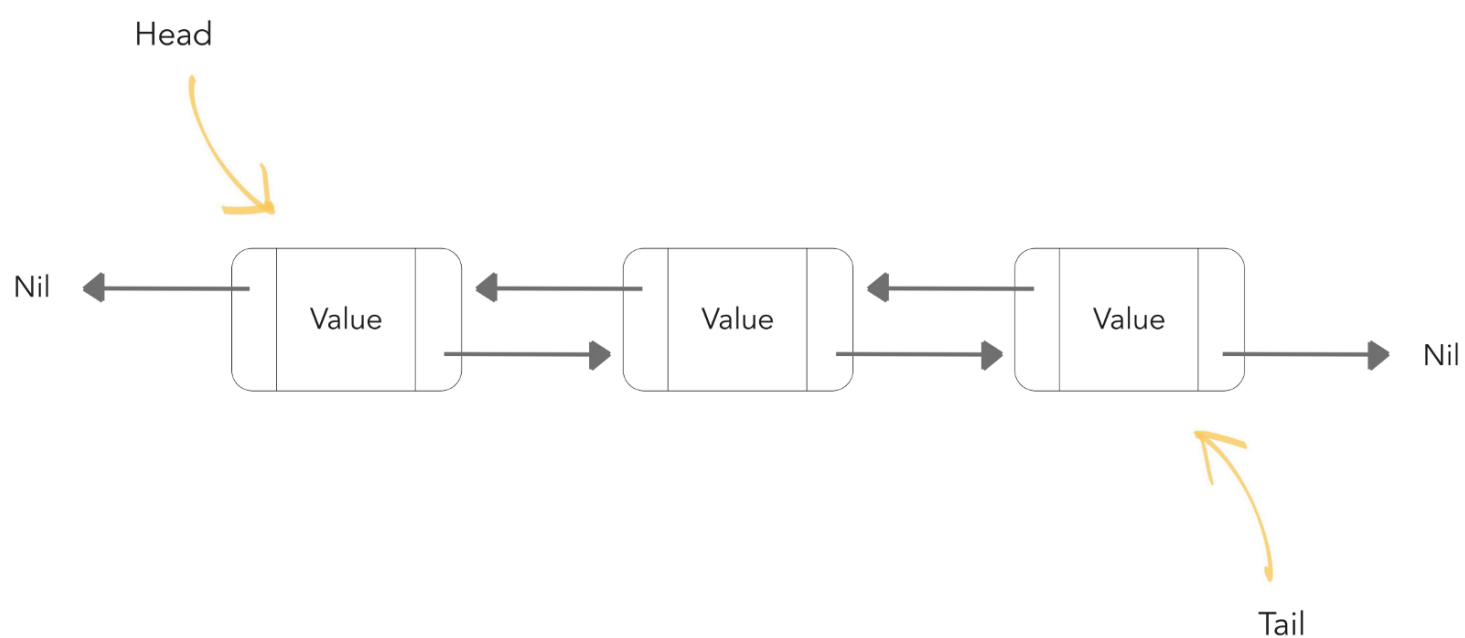
public class cwh_92_linkedlist {
    public static void main(String[] args) {
        LinkedList<Integer> l1 = new LinkedList<>();
        LinkedList<Integer> l2 = new LinkedList<>();
        l2.add(15);
        l2.add(18);
        l2.add(19);

        l1.add(6);
        l1.add(7);
        l1.add(4);
        l1.add(6);
        l1.add(0, 5);
        l1.add(0, 1);
        l1.addAll(0, l2);
        l1.addLast(676);
        l1.addFirst(788);
        System.out.println(l1.contains(27));
        System.out.println(l1.indexOf(6));
        System.out.println(l1.lastIndexOf(6));
        //l1.clear();
        l1.set(1, 566);
        for(int i=0; i<l1.size(); i++){
            System.out.print(l1.get(i));
            System.out.print(", ");
        }
    }
}
```

Copy

LinkedList in Java: Demo & Methods

- The LinkedList class in Java provides us with the doubly linked list data structure.
- Each element of the linked list is known as a node.
- Each node points to the address of the next node & its previous node.



- Linked lists are preferred over the Array list because the insertion & deletion in the linked lists can be done in a constant time. But, in arrays, if we want to add or delete an element in between then, we need to shift all the other elements.
- In a linked list, it is impossible to directly access an element because we need to traverse the whole linked list to get the desired element.

ArrayList Vs. LinkedList :

Although ArrayList & LinkedList both implement the List interface and have the same methods, it is important to understand when to use which one.

- The insertion & deletion can be done in constant time in Linked List, so it is best to use the linked list when you need to add or remove elements frequently.
- Use ArrayList when you want to access the random elements frequently, as it can't be done in a linked list in constant time.

Performing various operations on LinkedList :

1. Adding Element in LinkedList:

- Similar to ArrayList, add() method is used to add elements in a linked list.
- add(Object): Inserts an element at the end of the ArrayList.
- add(Index,Object) : Inserts an element at the given index.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LinkedList<Integer> l1 = new LinkedList<>();

        l1.add(11);
        l1.add(22);
        l1.add(33);
        l1.add(44);
        l1.add(55);
        l1.add(77);
        l1.add(5,77); // Inserts 77 at index 5
        System.out.println("L1 Linked list : "+ l1);
    }
}
```

Copy

Output :

```
L1 Linked list : [11, 22, 33, 44, 55, 77, 77]
```

Copy

2. Removing an element from the LinkedList:

- `remove()` method is used to remove an element from the linked list.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        LinkedList<Integer> l1 = new LinkedList<>();

        l1.add(11);
        l1.add(22);
        l1.add(33);
        l1.add(44);
        l1.add(55);
        l1.add(77);
        l1.add(5,77);
        System.out.println("L1 Linked list before: "+ l1);

        l1.remove(2); //removes element present at 2nd index
        System.out.println("L1 Linked list after: " + l1);

    }
}
```

Copy

Output :

```
L1 Linked list before: [11, 22, 33, 44, 55, 77, 77]
```

```
L1 Linked list after: [11, 22, 44, 55, 77, 77]
```

Copy

3. Changing An Element Of Linked List :

- `set()` method is used to change an already existing element of a linked list.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        LinkedList<Integer> l1 = new LinkedList<>();

        l1.add(11);
        l1.add(22);
        l1.add(33);
        l1.add(44);
        l1.add(55);
        l1.add(66);
```

```
        System.out.println("L1 Linked list before: "+ l1);

        l1.set(2,10); //changes element present at 2nd index (33 changed to 10)
        System.out.println("L1 Linked list after: " + l1);

    }
}
```

Copy

Output :

```
L1 Linked list before: [11, 22, 33, 44, 55, 66]
L1 Linked list after: [11, 22, 10, 44, 55, 66]
```

Copy

4. Inserting an element at the last of the linked list:

- addlast() method is used to insert an element at the start of the linked list.

Example :

```
F. import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        LinkedList<Integer> l1 = new LinkedList<>();

        l1.add(11);
        l1.add(22);
        l1.add(33);
        l1.add(44);
        l1.add(55);
        l1.add(66);
        System.out.println("L1 Linked list before: "+ l1);

        l1.addLast(100); //Inserting 100 at the end of L1
        System.out.println("L1 Linked list after inserting element at last index: " +
l1);

    }
}
```

Copy

Output :

```
L1 Linked list before: [11, 22, 33, 44, 55, 66]
L1 Linked list after inserting element at last index: [11, 22, 33, 44, 55, 66, 100]
```

Copy

5. Inserting an element at the start of the linked list:

- addFirst() method is used to insert an element at the start of the linked list.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        LinkedList<Integer> l1 = new LinkedList<>();
```

```

        l1.add(11);
        l1.add(22);
        l1.add(33);
        l1.add(44);
        l1.add(55);
        l1.add(66);
        System.out.println("L1 Linked list before: "+ l1);

        l1.addFirst(0); //Inserting 0 at the starting of L1
        System.out.println("L1 Linked list after: " + l1);

    }
}

```

Copy

Output :

```

L1 Linked list before: [11, 22, 33, 44, 55, 66]
L1 Linked list after: [0, 11, 22, 33, 44, 55, 66]

```

Copy

Code as described/written in the video :

```

package com.company;

import java.util.*;

public class cwh_92_linkedlist {
    public static void main(String[] args) {
        LinkedList<Integer> l1 = new LinkedList<>();
        LinkedList<Integer> l2 = new LinkedList<>();
        l2.add(15);
        l2.add(18);
        l2.add(19);

        l1.add(6);
        l1.add(7);
        l1.add(4);
        l1.add(6);
        l1.add(0, 5);
        l1.add(0, 1);
        l1.addAll(0, l2);
        l1.addLast(676);
        l1.addFirst(788);
        System.out.println(l1.contains(27));
        System.out.println(l1.indexOf(6));
        System.out.println(l1.lastIndexOf(6));
        //l1.clear();
        l1.set(1, 566);
        for(int i=0; i<l1.size(); i++){
            System.out.print(l1.get(i));
            System.out.print(", ");
        }
    }
}

```



```
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

ArrayDeque in Java

- ArrayDeque = Resizable array + Deque interface.
- ArrayDeque implements the Queue & Deque interface.
- There are no capacity restrictions for ArrayDeque, and it provides us the facility to add or remove any element from both sides of the queue.
- Also known as Array Double Ended Queue.
- It is faster than Linked list and stack.

Constructors of ArrayDeque class :

1. **ArrayDeque():** Used to create an empty array deque that has the capacity to hold 16 elements.
2. **ArrayDeque(int numElements):** Used to create an empty array deque that has the capacity to hold the specified number of elements.
3. **ArrayDeque(Collection<? extends E> c):** Used to create an array deque containing all the elements of the specified collections.

Performing Various Operation On ArrayDeque() :

1. Inserting an element :

- **Insertion at front :** add(), offerFirst() and addFirst() methods are used to insert an element at front of an array deque.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.addFirst(5);
        ad1.offerFirst(10);
        System.out.println(ad1);

    }
}
```

Copy

Output :

```
[10, 5, 6, 56, 9]
```

Copy

- **Insertion At End:** addLast() and offerLast() methods are used to insert an element at the end of the array deque.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
```

```
        ad1.add(9);
        ad1.addLast(5);
        ad1.offerLast(10);

        System.out.println(ad1);

    }
}
```

Copy

Output :

```
[6, 56, 9, 5, 10]
```

Copy

2. Accessing an element :

- **Accessing an element from the head of the deque array:** getFirst() & peekFirst() methods are used to get the first element of the deque array.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.add(10);
        ad1.add(91);
        ad1.add(19);

        System.out.println(ad1.getFirst());
        System.out.println(ad1.peekFirst());

    }
}
```

Copy

Output :

```
6
```

```
6
```

Copy

- **Accessing the last element:** getLast() or peekLast() methods are used to print the last element of the deque array.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
```

```
        ad1.add(9);
        ad1.add(10);
        ad1.add(91);
        ad1.add(19);

        System.out.println(ad1.getLast());
        System.out.println(ad1.peekLast());

    }
}
```

Copy

Output :

```
19
19
```

Copy

3. Removing an element :

- **Removing the first element:** removeFirst() & pollFirst() methods are used to delete an element from the head of the queue.
- removeFirst() throws an exception if the queue is empty.
- pollFirst() returns null if the queue is empty.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.add(10);
        ad1.add(91);
        ad1.add(19);

        ad1.pollFirst(); //deletes 6
        ad1.removeFirst(); //deletes 56

        System.out.println(ad1);

    }
}
```

Copy

Output :

```
[9, 10, 91, 19]
```

Copy

- **Removing the last element:** removeLast() & pollLast() methods are used to delete an element from the tail of the queue.

Example :

```
import java.util.*;
public class CWH extends Thread{
```

```

    public static void main(String[] args) {

        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.add(10);
        ad1.add(91);
        ad1.add(19);

        ad1.pollLast(); //deletes 19
        ad1.removeLast(); //deletes 91

        System.out.println(ad1);

    }
}

```

Copy

Output :

```
[6, 56, 9, 10]
```

Copy

Code as described/written in the video :

```

package com.company;

import java.util.ArrayDeque;

public class cwh_93_arraydeque {
    public static void main(String[] args) {
        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.addFirst(5);
        System.out.println(ad1.getFirst());
        System.out.println(ad1.getLast());
    }
}

```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Code as described/written in the video :

```

package com.company;

import java.util.ArrayDeque;

public class cwh_93_arraydeque {
    public static void main(String[] args) {
        ArrayDeque<Integer> ad1 = new ArrayDeque<>();
        ad1.add(6);
        ad1.add(56);
        ad1.add(9);
        ad1.addFirst(5);
    }
}

```

```
        System.out.println(ad1.getFirst());
        System.out.println(ad1.getLast());
    }
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

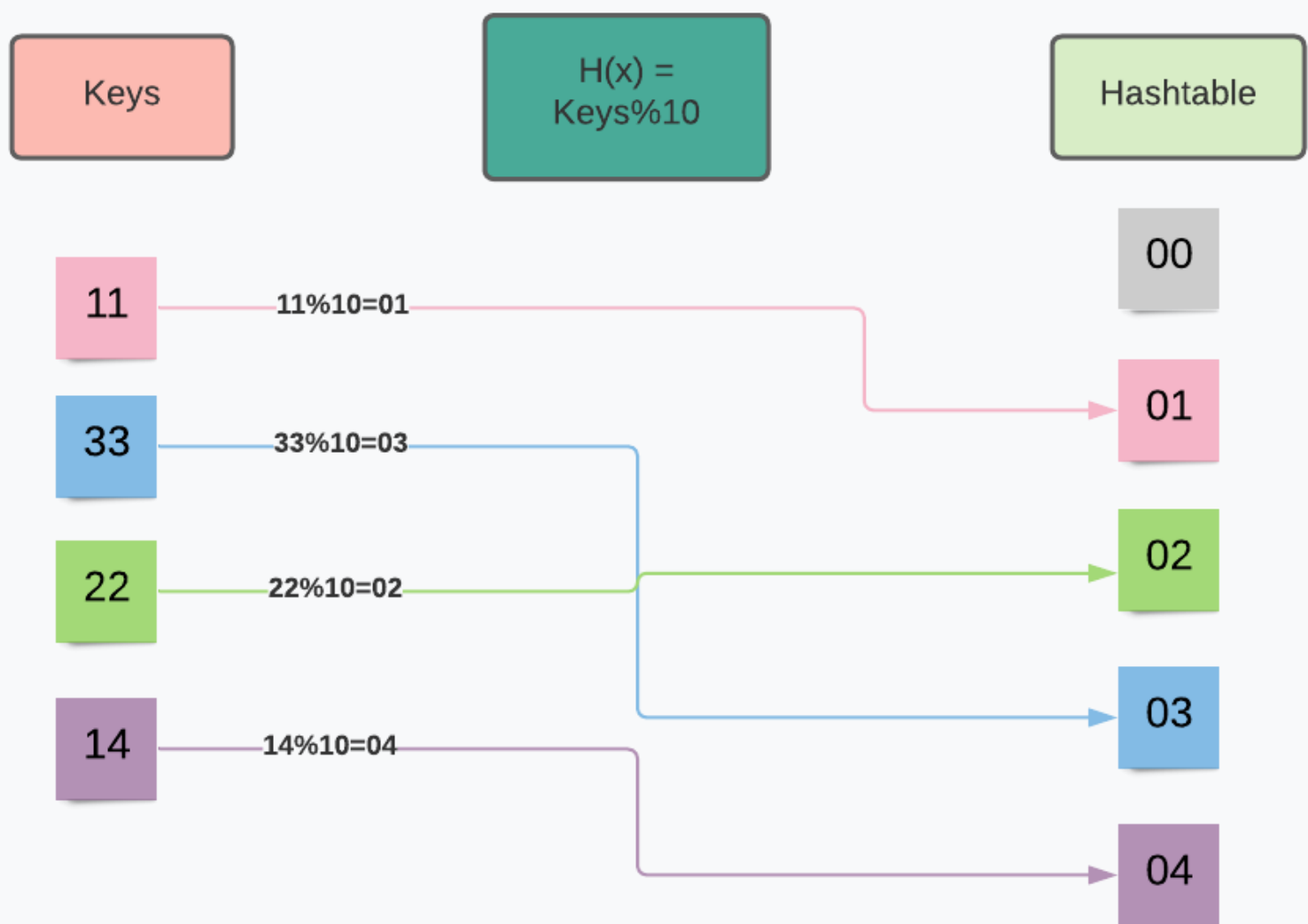
Hashing in Java

Hashing is the technique to convert the range of key-value pairs to a range of indices. In hashing, we use hash functions to map keys to some values.

Example :

Let arr=[11,33,22,14]

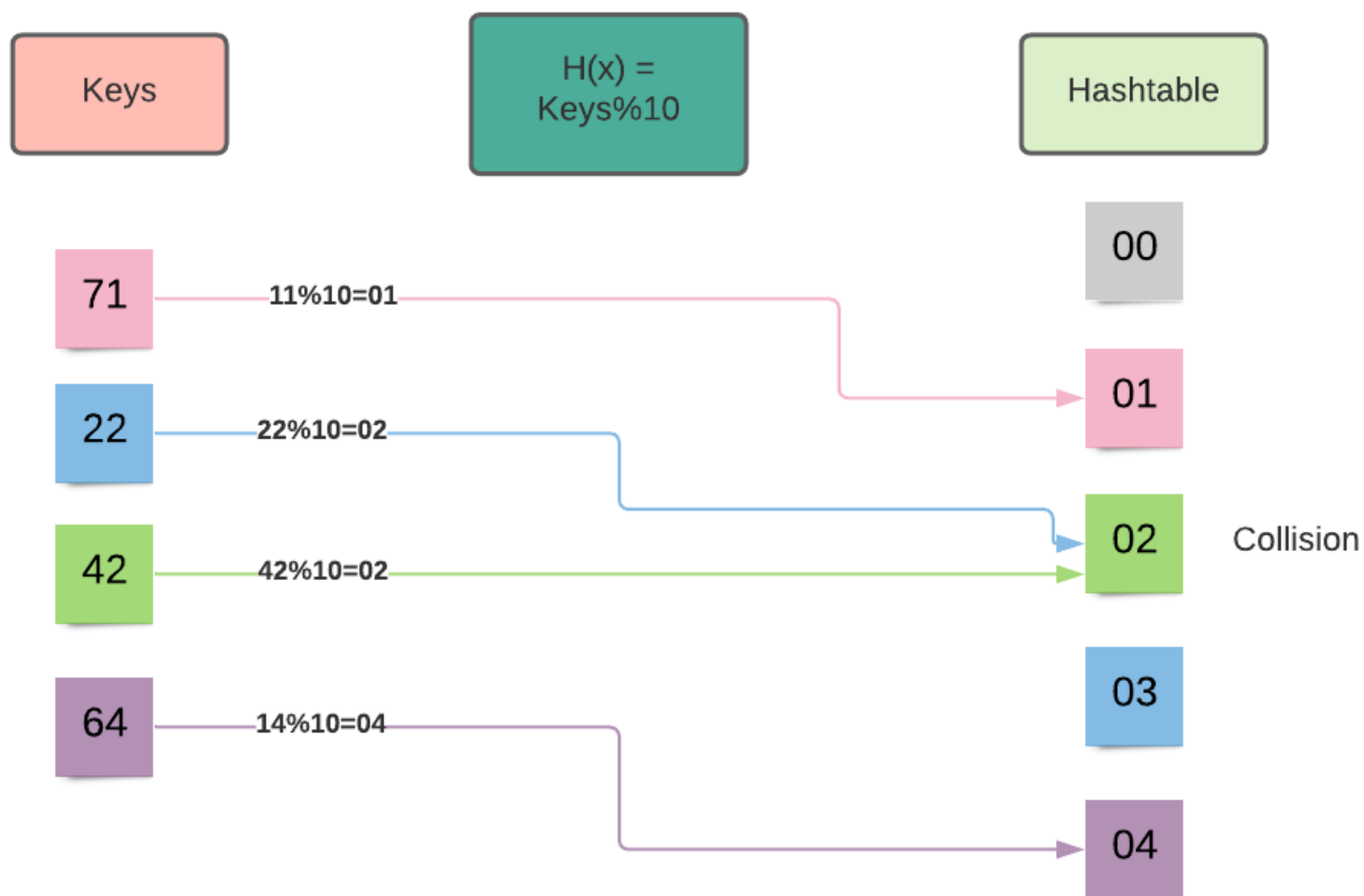
hashIndex = (key %10)



Collision: The hash function may map two key values to a single index. Such a situation is known as a collision.

Example : Let 1l=[22,42,64,71]

$H(x) = \text{keys} \% 10$



In the above image, you can see that the 22 and 44 are mapped to the index number 2. Therefore we need to avoid the collision. Following techniques are used to avoid collision in hashing :

- Open addressing
- Chaining

Ultimate Java Cheatsheet: [Click To Download](#)

HashSet in Java

- HashSet class uses a hash table for storing the elements.
- It implements the set interface.
- Duplicate values are not allowed.
- Before storing any object, the hashset uses the hashCode() and equals() method to check any duplicate entry in the hash table.
- Allows null value.
- Best suited for search operations.

Constructors Of HashSet :

1. **HashSet():** This constructor is used to create a new empty HashSet that can store 16 elements and have a load factor of 0.75.
2. **HashSet(int initialCapacity):** This constructor is used to create a new empty HashSet which has the capacity to store the specified number of elements and having a load factor of 0.75.
3. **HashSet(int initialCapacity, float loadFactor):** This constructor is used to create a new empty HashSet with the capacity & load factor equal to specified integer and float value.
4. **HashSet(Collection<? extends E> c):** This constructor is used to create a HashSet using the elements of collection c.

Performing Various Operations On HashSet :

1. Inserting elements :

- add() method is used to add elements in HashSet.
- The insertion order of the elements does not remains preserved in HashSet.
- All the duplicate elements are ignored because the set contains only unique values.

Example :

```
import java.util.*;  
public class CWH extends Thread{
```

```
public static void main(String[] args) {
    HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
    myHashSet.add(6);
    myHashSet.add(8);
    myHashSet.add(3);
    myHashSet.add(11);
    myHashSet.add(11); // This element will be ignored

    System.out.println(myHashSet);

}
}
```

Copy

Output :

```
[8, 3, 11, 6]
```

Copy

2. Removing elements from the HashSet :

- remove() method is used to delete the specified element from the HashSet.
- This method does not throws any exception if the specified element is not present in the HashSet.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
        myHashSet.add(6);
        myHashSet.add(8);
        myHashSet.add(3);
        myHashSet.add(11);
        myHashSet.add(11); // This element will be ignored

        System.out.println("myHashSet before removing any element : " + myHashSet);
        myHashSet.remove(3); //deletes 3 from the hashset
        System.out.println("myHashSet after removing a element : " + myHashSet);

    }
}
```

Copy

Output :

```
myHashSet before removing any element : [8, 3, 11, 6]
```

```
myHashSet after removing a element : [8, 11, 6]
```

Copy

3. Checking if the HashSet is empty or not :

- - isEmpty() method is used to check if there is any object in the HashSet or not.
 - This method returns a boolean value.

Example :

```
import java.util.*;
```

```
public class CWH extends Thread{
    public static void main(String[] args) {
        HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
        myHashSet.add(6);
        myHashSet.add(8);
        myHashSet.add(3);
        myHashSet.add(11);

        HashSet<Integer> myHashSet1 = new HashSet<>();

        System.out.println(myHashSet.isEmpty());
        System.out.println(myHashSet1.isEmpty());

    }
}
```

Copy

Output :

```
false
true
```

Copy

4. Removing all the elements from the HashSet :

- clear() method is used to remove all the elements from the HashSet at once.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
        myHashSet.add(16);
        myHashSet.add(33);
        myHashSet.add(78);
        myHashSet.add(19);
        myHashSet.add(29);
        myHashSet.add(10);

        System.out.println("myHashSet before : " + myHashSet);
        myHashSet.clear(); //deletes all the elements from the hashset
        System.out.println("myHashSet after  : " + myHashSet);

    }
}
```

Copy

Output :

```
myHashSet before : [16, 33, 19, 10, 29, 78]
myHashSet after  : []
```

Copy

5. Printing the size of the HashSet :

- size() method is used to get the size of the HashSet.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
        myHashSet.add(16);
        myHashSet.add(33);
        myHashSet.add(78);
        myHashSet.add(19);
        myHashSet.add(29);
        myHashSet.add(10);

        System.out.println("The size of myHashSet is : " + myHashSet.size());

    }
}
```

Copy

Output :

```
The size of myHashSet is : 6
```

Copy

Code as described/written in the video :

```
package com.company;

import java.util.HashSet;

public class cwh_95_set {
    public static void main(String[] args) {
        HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);
        myHashSet.add(6);
        myHashSet.add(8);
        myHashSet.add(3);
        myHashSet.add(11);
        myHashSet.add(11);
        System.out.println(myHashSet);
    }
}
```

Copy

- **Ultimate Java Cheatsheet:** [Click To Download](#)

Date and Time in Java

Date & Time in Java

java time -> package for date & time in java from java onwards

Before java 8, java util package used to hold the date time class now these classes are deprecated

How java stores a Date?

Date in java is stored in the form of a long numer. 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 current Time Millis () returns no of second passed Once no. of ms are calculated, we can calculate minutes, seconds & years passed

Quick quiz: Is it save to store the no. of ms in a variable of type long?

```
package com.company;

public class cwh_96_date {
    public static void main(String[] args) {
        System.out.println(System.currentTimeMillis()/1000/3600/24/365);
    }
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

The Date Class in Java

Answer of quiz asked in the previous tutorial:

Question: Is it safe to store the number of milliseconds in a variable of type long?

Answer: Yes, it is absolutely safe to store the number of milliseconds in a variable of type long because the maximum value that can be stored in long is 9223372036854775807. You can see that the maximum value of long is huge. Therefore, we do not need to worry about the value of milliseconds. Notice the output of the below code; the value of current time in millisecond is 10^6 times smaller than the maximum value of long data type.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        System.out.println("The maximum value of long is :" + Long.MAX_VALUE);
        System.out.println("The value of current time in ms : " + System.currentTimeMillis());
    }
}
```

Copy

Output :

```
The maximum value of long is :9223372036854775807
The value of current time in ms : 1621708466975
```

Copy

Date Class In Java :

- Date class in java is available in java.util package.
- This class provides the instant in time with precision of millisecond.

Constructors of the date class :

1. **Date():** This constructor is used when we need an object of current date and time.
2. **Date(long milliseconds):** This constructor creates a date object from the number of milliseconds passed since January 1, 1970.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Date d= new Date();
        System.out.println("The current date is : " + d);
    }
}
```

```
        Date d1= new Date(1621709639111l);

        System.out.println("The date calculated form miliseconds is : " + d1);

    }
}
```

Copy

Output :

```
The current date is : Sun May 23 00:24:17 IST 2021
The date calculated form miliseconds is : Sun May 23 00:23:59 IST 2021
```

Copy

Methods of date class :

1. compareTo() :

- Checks for the equality of the two dates.
- Returns 0 if the dates are equal; else, returns 1.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Date d= new Date();

        Date d1= new Date(2021,12, 24); //both dates are different
        System.out.println(d1.compareTo(d));

    }
}
```

Copy

Output :

```
1
```

Copy

2. getTime() :

- This method returns the number of milliseconds passed since the midnight of January 1, 1970.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Date d= new Date(2021,5,23);

        System.out.println("The number of milliseonds passed since Jan 1, 1970 :"+d.getTime() );

    }
}
```

Copy

Output :

```
The number of milliseonds passed since Jan 1, 1970 :61582530600000
```

Copy

3. `getYear()` : Prints the current year.
`getDate()` : Prints the current date.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Date d= new Date();
        System.out.println("The current date is : "+ d.getDate());
        System.out.println("The current year is : "+ d.getYear()); //19

    }
}
```

Copy

Output :

```
The current date is : 23
The current year is : 121
```

Copy

Code as described/written in the video :

```
package com.company;

import java.util.Date;
public class cwh_97_date_class {
    public static void main(String[] args) {
//        System.out.println(Long.MAX_VALUE);
//        System.out.println(System.currentTimeMillis());
        Date d = new Date();
        System.out.println(d);
        System.out.println(d.getTime());
        System.out.println(d.getDate());
        System.out.println(d.getSeconds());
        System.out.println(d.getYear());
    }
}
```

Copy

Ultimate Java Cheatsheet: [Click To Download](#)

Calendar Class in Java

- The calendar class in java provides the methods that helps in converting date between a specific instant in time.
- It is an abstract class.
- Since it is an abstract class, we can not create an instance of this class with the help of a constructor.
- We use the static method `Calender.getInstance()` in order to implement a sub-class.

Example to demonstrate the `getInstance()` method :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        System.out.println(c.getCalendarType()); //getCalendarType() returns the type of the
calendar
```

```
}  
}
```

Copy

Output :

```
gregory
```

Copy

Constructors of the Calendar class :

1. **Calendar():** This constructor is used to construct a calendar with the default time zone & locale.
2. **Calendar(Time zone, Locale locale):** This constructor is used to construct a calendar with the specified time zone & locale.

Methods of the Calendar class :

1. get(int field) :

- o This method returns the value of the specified calendar field.

Example :

```
import java.util.*;  
public class CWH extends Thread{  
    public static void main(String[] args) {  
        Calendar c = Calendar.getInstance();  
        System.out.println("Current year is :"+ c.get(Calendar.YEAR));  
        System.out.println("Current month is :"+ c.get(Calendar.MONTH)); //The indexing  
for month field ranges from [0,11]  
        System.out.println("Current day is :"+ c.get(Calendar.DAY_OF_WEEK));  
        System.out.println("Current hour is :"+ c.get(Calendar.HOUR_OF_DAY));  
        System.out.println("Current minute is :"+ c.get(Calendar.MINUTE));  
        System.out.println("Current second is :"+ c.get(Calendar.SECOND));  
  
    }  
}
```

Copy

Output :

```
Current year is :2021  
Current month is :4  
Current day is :1  
Current hour is :12  
Current minute is :3  
Current second is :3
```

Copy

2. add(int field, int amount) :

- o This method is useful for calculating the time after or before of a specified calendar field.

Example :

```
import java.util.*;  
public class CWH extends Thread{  
    public static void main(String[] args) {  
        Calendar c = Calendar.getInstance();  
        System.out.println("Current date is : " + c.getTime());  
    }  
}
```

```
        c.add(Calendar.YEAR, 4);
        System.out.println("After 4 years  : "+ c.getTime());
        c.add(Calendar.YEAR, -12);
        System.out.println("Before 12 years  : "+ c.getTime());
        c.add(Calendar.MONTH,2);
        System.out.println("After 2 months  : "+ c.getTime());

    }
}
```

Copy

Output :

```
Current date is : Sun May 23 12:14:24 IST 2021
After 4 years  : Fri May 23 12:14:24 IST 2025
Before 12 years  : Thu May 23 12:14:24 IST 2013
After 2 months  : Tue Jul 23 12:14:24 IST 2013
```

Copy

3. `getWeeksInWeekYear()` :

- Returns the number of weeks.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        System.out.println(c.getWeeksInWeekYear());
    }
}
```

Copy

Output :

```
52
```

Copy

4. `getMaximum(int field)` :

- Returns the maximum value for the specified calendar field.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        System.out.println(" The maximum no. of weeks in a year : " +
c.getMaximum(Calendar.WEEK_OF_YEAR));

    }
}
```

Copy

Output :

```
53
```

GregorianCalendar class & TimeZone in java

- GregorianCalendar class is the concrete sub-class of the Calendar class.
- This class supports both the Julian and Gregorian calendar systems.

Difference between Calendar and GregorianCalendar class :

The calendar class is an abstract class. So, the instance of this class can not be instantiated. Therefore, we need to use the static method Calendar.getInstance() to initialize the object of the Calendar class :

```
Calendar c = Calendar.getInstance();
```

Copy

Since the GregorianCalendar class is a concrete subclass, it can be initialized as :

```
GregorianCalendar gcal = new GregorianCalendar();
```

Copy

Constructors of the GregorianCalendar class :

1. **GregorianCalendar():** This constructor is used to initialize an object with the current time in the default time zone.
2. **GregorianCalendar(int year, int month, int day):** This constructor is used to initialize an object with the date-set specified as parameters in the default time zone and default locale.
3. **GregorianCalendar(int year, int month, int day, int hours, int minutes):** This constructor initializes an object with the given date and time set in the default locale and time zone.
4. **GregorianCalendar(int year, int month, int day, int hours, int minutes, int seconds):** This constructor initializes an object with the more specific time and date-set passed as a parameter in the default locale and time zone.
5. **GregorianCalendar(Locale locale):** Initializes a GregorianCalendar object with the current date and time in the default time zone and the specified locale.
6. **GregorianCalendar(TimeZone timeZone):** Initializes a GregorianCalendar object with the current date and time in the default locale and the specified time zone.
7. **GregorianCalendar(TimeZone timeZone, Locale locale):** Initializes an object with the locale and timezone passed as parameters.

Methods of the Gregorian class :

1. isLeapYear(int year) :

- Checks if the year passed as a parameter is a leap year or not.
- This method returns a boolean value.

Example :

```
import java.util.*;

public class CWH extends Thread{

    public static void main(String[] args) {

        GregorianCalendar cal = new GregorianCalendar();
        System.out.println(cal.isLeapYear(2000));
        System.out.println(cal.isLeapYear(2021));

    }
}
```

Copy

Output :

```
true
false
```

Copy

2. roll(int field, boolean up) :

- This method adds/subtracts a single unit of time from the specified time field.
- true = rolls up the value by 1.
- false = rolls down the value by 1.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        GregorianCalendar c = new GregorianCalendar();
        System.out.println("Date before rolling : " + c.getTime());

        c.roll(Calendar.MONTH, true);
        c.roll(Calendar.DATE, false);
        c.roll(Calendar.YEAR, true);

        System.out.println("Date after rolling : " + c.getTime());
    }
}
```

Copy

Output :

```
Date before rolling : Wed May 26 07:53:24 IST 2021
Date after rolling : Sat Jun 25 07:53:24 IST 2022
```

Copy

3. hashCode():

- This method returns the hashCode of the calendar object.

Example :

```
import java.util.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        GregorianCalendar c = new GregorianCalendar();
        System.out.println("Calendar : " + c.getTime());

        System.out.println("The hashCode for this calendar is : "+ c.hashCode());
    }
}
```

Copy

Output :

```
Calendar : Wed May 26 08:08:33 IST 2021
The hashCode for this calendar is : 1358707903
```

Copy

Code as described/written in the video :

```
package com.company;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;
```



```

public class cwh_99_gregorian {
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        System.out.println(c.getTime());
        System.out.println(c.get(Calendar.DATE));
        System.out.println(c.get(Calendar.SECOND));
        System.out.println(c.get(Calendar.HOUR));
        System.out.println(c.get(Calendar.HOUR_OF_DAY) + ":" + c.get(Calendar.MINUTE) + ":" +
c.get(Calendar.SECOND));
        GregorianCalendar cal = new GregorianCalendar();
        System.out.println(cal.isLeapYear(2018));
        System.out.println(TimeZone.getAvailableIDs()[0]);
        System.out.println(TimeZone.getAvailableIDs()[1]);
        System.out.println(TimeZone.getAvailableIDs()[2]);
    }
}

```

java.time API - Classes & Methods

Date and time features in Java is primarily supported by two packages :

- java.util
- java.time

The package java.time was added with the release of Java 8 with the aim of solving problems faced by Java developers while handling date and time with java.util package such as representing a date without time, etc.

Classes of Java.time :

1. Clock class:

- This class provides access to the current instant, date and time zone using a time-zone.
- Clock class is an abstract class therefore it is not possible to create instance of the clock class.

Some methods of the clock class :

- **abstract ZoneId getZone()** : This methods returns the time zone being used to create date and time objects.

Example :

```

import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Clock cl = Clock.systemDefaultZone();
        System.out.println(cl.getZone());
    }
}

```

Copy

Output :

```
Asia/Calcutta
```

Copy

- **abstract Instant instant()** : This methods returns the current instant of the clocks.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Clock cl = Clock.systemUTC();

        System.out.println(cl.instant());
    }
}
```

Copy

Output :

```
2021-05-26T06:43:05.064640700Z
```

Copy

2.

3. Duration class :

- This class is used to measure time in seconds and nano seconds.
- This class is immutable.

Some Methods of the duration class :

- **boolean isNegative()** : This method is used to check if the duration is negative.
Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Duration d1 = Duration.between(LocalTime.MIN,LocalTime.NOON); //LocalTime.MIN = '00:00' , LocalTime.NOON = '12:00'
        System.out.println(d1.isNegative());

        Duration d2 = Duration.between(LocalTime.MAX,LocalTime.MIN); //LocalTime.MAX = '23:59:59.999999999' , LocalTime.MIN = '00:00'
        System.out.println(d2.isNegative());
    }
}
```

Copy

Output :

```
false
true
```

Copy

- **isZero()** : This method is used to check if the duration is zero. Returns boolean value.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        Duration d1 = Duration.between(LocalTime.MIN,LocalTime.MIDNIGHT);
//LocalTime.MIN = '00:00' , LocalTime.NOON = '00:00'
        System.out.println(d1.isZero());

        Duration d2 = Duration.between(LocalTime.MAX,LocalTime.MIN); //LocalTime.MAX = '23:59:59.999999999' , LocalTime.MIN = '00:00'
```

```
        System.out.println(d2.isZero());
    }
}
```

Copy

Output :

```
true
false
```

Copy

4. LocalDate class :

- This class is useful for representing the dates in the year-month-day format.
- With the help of LocalDate class, dates can be represented without time.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalDate d = LocalDate.now();
        System.out.println(d);
    }
}
```

Copy

Output :

```
2021-05-26
```

Copy

Some methods of the LocalDate class :

- **compareTo()** : This method compares the equality of the two dates. Returns boolean value.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalDate d = LocalDate.parse("2021-05-27");
        LocalDate d1= LocalDate.parse("2021-05-26");
        LocalDate d2= LocalDate.parse("2021-05-26");

        System.out.println(d1.equals(d));
        System.out.println(d2.equals(d1));
    }
}
```

Copy

Output :

```
false
true
```

Copy

- **withYear(int Year)** : This method returns a copy of the LocalDate but alters the year with the value of year passed as argument.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalDate d = LocalDate.parse("2021-05-27");
        System.out.println(d.withYear(2001));
    }
}
```

Copy

Output :

```
2001-05-27
```

Copy

5. LocalTime class :

- This class helps us to represent the time without the dates.
- Instances of LocalTime class are mutable.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalTime t = LocalTime.now();
        System.out.println(t);
    }
}
```

Copy

Output :

```
13:13:36.198479100
```

Copy

Some methods of the LocalTime class :

- **LocalTime plusHours(long hoursToAdd)** : This method returns a copy of the LocalTime but with the specified number of hours added.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalTime t = LocalTime.of(13,18,29);
        System.out.println("Time before : " + t);

        LocalTime t1= t.plusHours(5);
        System.out.println("Time after adding 5 hours : " + t1);
    }
}
```

Copy

Output :

```
Time before : 13:18:29
Time after adding 5 hours : 18:18:29
```

Copy

- **LocalTime minusMinutes(long minutesToSubtract)** : This method returns a copy of the LocalTime but with the specified number of minutes subtracted.

Example :

```
import java.time.*;
public class CWH extends Thread{
    public static void main(String[] args) {
        LocalTime t = LocalTime.of(15,28,19);
        System.out.println("Time before : " + t);

        LocalTime t1= t.minusMinutes(8);
        System.out.println("Time after subtracting 8 minutes : " + t1);

    }
}
```

Copy

Output :

```
Time before : 15:28:19
Time after subtracting 8 minutes : 15:20:19
```

Copy

Code as described/written in the video :

```
package com.company;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class cwh_100_java_time {
    public static void main(String[] args) {
        LocalDate d = LocalDate.now();
        System.out.println(d);

        LocalTime t = LocalTime.now();
        System.out.println(t);

        LocalDateTime dt = LocalDateTime.now();
        System.out.println(dt);
    }
}
```

Copy

DateTimeFormatter in Java

- This class helps us to print and parse date and time in our desired format.
- The format() method of the DateTimeFormatter class is used to format the dates using our desired format.

Syntax :

```
public String format(DateTimeFormatter formatter)
```

Copy

Parameter :

The object of the formatter to be used is passed, and it can not be null.

Exception :

This method throws *DateTimeException*.

Return Value :

Returns the string in the format specified by the user.

Example :

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CWH extends Thread{
    public static void main(String[] args) {

        LocalDateTime dt = LocalDateTime.now();
        System.out.println("The current date is : " + dt);

        DateTimeFormatter df = DateTimeFormatter.ofPattern("dd.MM.yyyy"); //
This is the format

        String myDate = dt.format(df); // Creating date string using date
and format

        System.out.println("Date after formattin : "+ myDate);

    }
}
```

Copy

Output :

```
The current date is : 2021-05-26T18:15:42.554864400
Date after formattin : 26.05.2021
```

Copy

- In addition to the format, formatters can be created with desired Locale, Chronology, ZoneId, and DecimalStyle.

Programs to illustrate some of the predefined formatters of the DateTimeFormatter class :

1. ISO_LOCAL_DATE :

- Formats the date according to the International Standard for the representation of dates.

Example :

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CWH extends Thread{
    public static void main(String[] args) {
```

```
        LocalDateTime dt = LocalDateTime.now();

        DateTimeFormatter df = DateTimeFormatter.ISO_LOCAL_DATE;//
Formatting the date in the ISO format

        String myDate = dt.format(df); // Creating date string using
date and format

        System.out.println("Date in ISO format : "+ myDate);

    }

}
```

Copy

Output :

```
Date in ISO format : 2021-05-26
```

Copy

2. ISO_WEEK_DATE :

- - Returns the number of weeks and year.

Example :

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CWH extends Thread{
    public static void main(String[] args) {

        LocalDateTime dt = LocalDateTime.now();

        DateTimeFormatter df = DateTimeFormatter.ISO_WEEK_DATE;//

        String myDate = dt.format(df);
        System.out.println("Date in ISO_WEEK_DATE Format : "+
myDate);

    }

}
```

Copy

Output :

```
Date in ISO_WEEK_DATE Format : 2021-W21-3
```

Copy

3. ISO_ORDINAL_DATE :

- - Returns the year and day of the year.

Example :

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class CWH extends Thread{
```

```
public static void main(String[] args) {

    LocalDateTime dt = LocalDateTime.now();

    DateTimeFormatter df =
DateTimeFormatter.ISO_ORDINAL_DATE;//

    String myDate = dt.format(df);

    System.out.println("Date in ISO_WEEK_DATE Format  : "+
myDate);

}

}
```

Copy

Output :

```
Date in ISO_ORDINAL_DATE Format  : 2021-146
```

Copy

Patterns for formatting and parsing :

Pattern letters for all the alphabets(capital as well as small) are defined as follows :

Symbol	Meaning	Presentation	Examples
-----	-----	-----	-----
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
Q/q	quarter-of-year	number/text	3; 03; Q3; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	week-of-month	number	3
a	am-pm-of-day	text	PM
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-am-pm (1-24)	number	0
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30
s	second-of-minute	number	55
S	fraction-of-second	fraction	978
A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-0	GMT+8; GMT+08:00; UTC-08:00;
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -
08:30:15;			
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -
08:30:15;			
Z	zone-offset	offset-Z	+0000; -0800; -08:00;
p	pad next	pad modifier	1
'	escape for text	delimiter	'
''	single quote	literal	'
[optional section start		
]	optional section end		
#	reserved for future use		


```
{      reserved for future use
}
```

Code as described/written in the video :

```
package com.company;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class cwh_101_datetimeformatter {
    public static void main(String[] args) {
        LocalDateTime dt = LocalDateTime.now(); // This is the date
        System.out.println(dt);

        DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy -- E H:m a"); // This is
the format
        DateTimeFormatter df2 = DateTimeFormatter.ISO_LOCAL_DATE;

        String myDate = dt.format(df); // Creating date string using date and format
        System.out.println(myDate);

    }
}
```

Copy

[Download Notes Here](#)

Advanced Java Practice Set

Question 1: Create an ArrayList and store the names of ten students inside it. Print it using a for each loop.

Answer 1: As we discussed in the [ArrayList tutorial](#) of this playlist, add() method is used to insert element in an ArrayList. Below is the required program :

```
import java.util.ArrayList;
public class CWH{
    public static void main(String[] args) {
        ArrayList ar = new ArrayList();
        ar.add("Student 1");
        ar.add("Student 2");
        ar.add("Student 3");
        ar.add("Student 4");
        ar.add("Student 5");
        ar.add("Student 6");
        ar.add("Student 7");
        ar.add("Student 8");
        ar.add("Student 9");
        ar.add("Student 10");
        for(Object o: ar){
            System.out.println(o);
        }
    }
}
```

Copy

Output :

Student 1
Student 2
Student 3
Student 4
Student 5
Student 6
Student 7
Student 8
Student 9
Student 10

Copy

Question 2: Use the Date class in Java to print the time in the following format : **21:47:02**.

Answer 2: In the [Date class tutorial](#), we saw that how get() method can be used to print the dates in our desired format. Below is required program :

```
Date d = new Date();  
System.out.println(d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds());
```

Copy

Output :

19:13:17

Copy

Question 3: Repeat question number 2 using the Calendar class.

Answer 3: In the [Calendar class tutorial](#), we saw how we can use the Calendar.getInstance() static method to initialize an object of the Calendar class. After creating object, use the get() method to print the date in the desired formate. Below is the required program :

```
import java.util.Calendar;  
import java.util.Date;  
  
public class CWH{  
    public static void main(String[] args) {  
        Calendar c = Calendar.getInstance();  
        System.out.println(c.get(Calendar.HOUR_OF_DAY) + ":" + c.get(Calendar.MINUTE) +  
":" + c.get(Calendar.SECOND));  
    }  
}
```

Copy

Output :

19:15:26

Copy

Question 4: Repeat question number 2 using java.time API.

Answer 4: In the [DateTimeFormatter](#) class tutorial, I told you to use the format() method in order to change the date and time in the desired format.

```
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
public class CWH{  
    public static void main(String[] args) {  
        LocalDateTime dt = LocalDateTime.now(); // This is the date
```

```
        DateTimeFormatter df = DateTimeFormatter.ofPattern("H:m:s"); // This is the format
        String myDate = dt.format(df); // Creating date string using date and format
        System.out.println(myDate);
    }
}
```

Copy

Output :

```
19:27:59
```

Copy

Question 5: Create a Set in java. Try to store the duplicate values elements inside this set and verify that only one instance is stored.

Answer 5: In the [Hashset tutorial](#), we saw that only unique elements can be stored inside a Hashset. Below is the required code :

```
import java.util.HashSet;

public class CWH{
    public static void main(String[] args) {
        HashSet<Integer> s = new HashSet();
        s.add(5);
        s.add(6);
        s.add(46);
        s.add(60);
        s.add(9);
        s.add(6);
        System.out.println(s);
    }
}
```

Copy

Output :

```
[5, 6, 9, 60, 46]
```

Copy

Code as described/written in the video :

```
package com.company;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class cwh_102_ps {
    public static void main(String[] args) {
        // PS Q1
        ArrayList ar = new ArrayList();
        ar.add("Student 1");
        ar.add("Student 2");
        ar.add("Student 3");
        ar.add("Student 4");
    }
}
```

```

        ar.add("Student 5");
        ar.add("Student 6");
        ar.add("Student 7");
        ar.add("Student 8");
        ar.add("Student 9");
        ar.add("Student 10");
        for(Object o: ar){
            System.out.println(o);
        }

// PS Q2
Date d = new Date();
System.out.println(d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds());

// PS Q3
Calendar c = Calendar.getInstance();
System.out.println(c.get(Calendar.HOUR_OF_DAY) + ":" + c.get(Calendar.MINUTE) + ":" +
c.get(Calendar.SECOND));

// PS Q4
LocalDateTime dt = LocalDateTime.now(); // This is the date
DateTimeFormatter df = DateTimeFormatter.ofPattern("H:m:s"); // This is the format
String myDate = dt.format(df); // Creating date string using date and format
System.out.println(myDate);

// PS Q5
HashSet<Integer> s = new HashSet();
s.add(5);
s.add(6);
s.add(46);
s.add(60);
s.add(9);
s.add(6);
System.out.println(s);
    }
}

```

Copy

[Download Notes Here](#)

Java Exercise 6: Solution | Custom Calculator

Below is the solution of exercise number 6 that I gave you all in [tutorial 87](#) :

```

package com.company;

class InvalidInputException extends Exception{
    @Override
    public String toString() {
        return "Cannot add 8 and 9";
    }

    @Override
    public String getMessage() {
        return "I am getMessage()";
    }
}

```

```
}

class MaxInputException extends Exception{

    @Override
    public String toString() {
        return "Input cant be greater than 100000";
    }

    @Override
    public String getMessage() {
        return "I am getMessage()";
    }
}

class CannotDivideByZeroException extends Exception{

    @Override
    public String toString() {
        return "Cannot divide by 0";
    }

    @Override
    public String getMessage() {
        return "I am getMessage()";
    }
}

class MaxMultiplyInputException extends Exception{

    @Override
    public String toString() {
        return "Input cant be greater than 7000 while multiplying";
    }

    @Override
    public String getMessage() {
        return "I am getMessage()";
    }
}

class CustomCalculator {

    double add(double a, double b) throws InvalidInputException, MaxInputException{
        if(a>100000 || b>100000){
            throw new MaxInputException();
        }
        if(a==8 || b==9) {
            throw new InvalidInputException();
        }
        return a + b;
    }

    double subtract(double a, double b) throws MaxInputException{
        if(a>100000 || b>100000){
            throw new MaxInputException();
        }
        return a - b;
    }

    double multiply(double a, double b)throws MaxInputException, MaxMultiplyInputException{
        if(a>100000 || b>100000){
```

```

        throw new MaxInputException();
    }
    else if(a>7000 || b>7000){
        throw new MaxMultiplyInputException();
    }
    return a * b;
}

double divide(double a, double b) throws CannotDivideByZeroException, MaxInputException{
    if(a>100000 || b>100000){
        throw new MaxInputException();
    }
    if(b==0){
        throw new CannotDivideByZeroException();
    }
    return a / b;
}
}

public class cwh_103_ex6sol {
    public static void main(String[] args) throws InvalidInputException,
        CannotDivideByZeroException, MaxInputException, MaxMultiplyInputException {
        CustomCalculator c = new CustomCalculator();
//        c.add(8, 9);
//        c.divide(6, 0);
//        c.divide(600000000, 40);
        c.multiply(5, 9888);
        /*
        Exercise 6: You have to create a custom calculator with following operations:
        1. + -> Addition
        2. - -> Subtraction
        3. * -> Multiplication
        4. / -> Division
        which throws the following exceptions:
        1. Invalid input Exception ex: 8 & 9
        2. Cannot divide by 0 Exception
        3. Max Input Exception if any of the inputs is greater than 100000
        4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater
        than 7000
        */

    }
}

```

Copy

Java Exercise 7: Library Management System in Java

Create a library management system that is capable of issuing books to the students. Every book should have info like:

1. Book name
2. Book Author
3. Issued to
4. Issued on

Users should be able to add books, return issued books, issue books. Assume that all the users are registered with their names in the central database.

```
package com.company;
```

```

public class cwh_104_ex7 {
    public static void main(String[] args) {
        /*
        Create a library management system which is capable of issuing books to the students.
        Book should have info like:
        1. Book name
        2. Book Author
        3. Issued to
        4. Issued on
        User should be able to add books, return issued books, issue books
        Assume that all the users are registered with their names in the central database
        */
    }
}

```

Copy

Generating our own JavaDocs for our Package

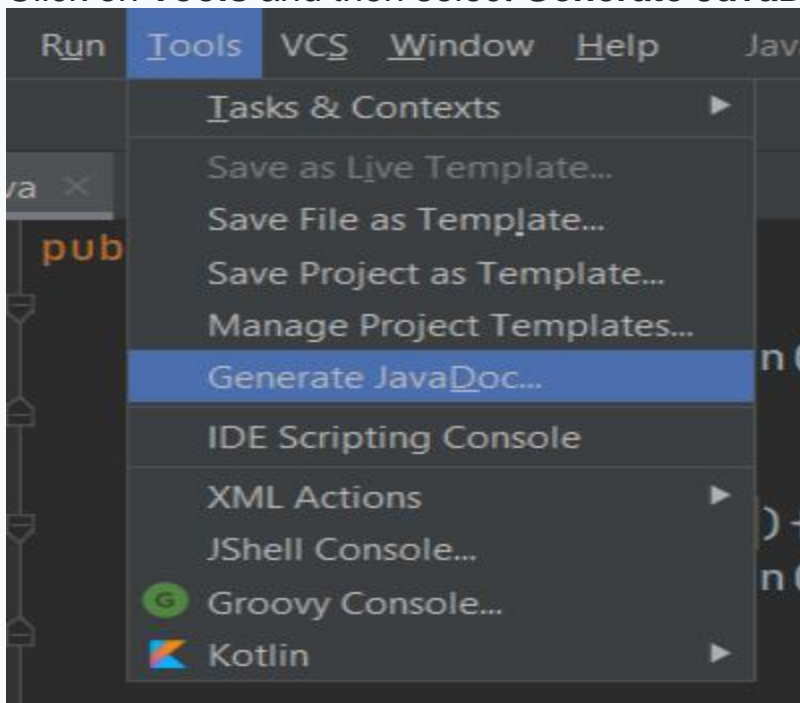
A reliable documentation is a must for a developer. Just imagine how difficult it would be to write code in a language you are not familiar with, without proper documentation. Documentation helps us to develop, maintain and transfer knowledge to other developers. As you are already aware that there is a proper documentation available for Java. But have you ever thought of creating your own documentation? In this tutorial, I will tell you how you can automatically generate documentation for your Java packages with the help of JavaDoc.

What is JavaDoc?

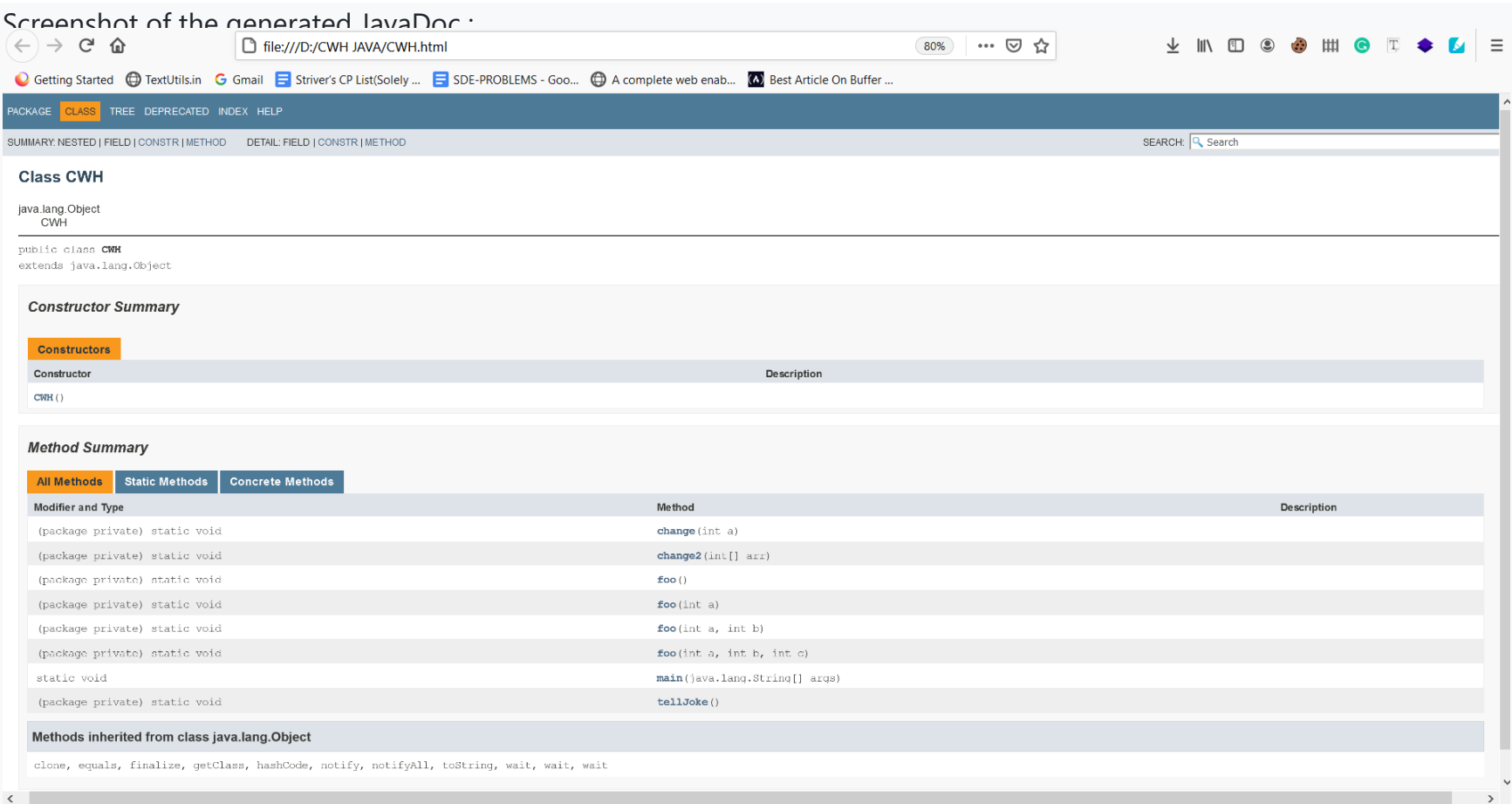
- It is a tool that automatically generates standard documentation in HTML format from the Java source code.
- This tool comes built-in with JDK(Java Development Kit).

Steps to create documentation with JavaDoc in IntelliJIdea :

1. Open your java program in IntelliJIdea.
2. Click on **Tools** and then select **Generate JavaDoc**.



3. A pop window will open. Select the project and packages for which you want to create the JavaDoc.
4. Select the classes for which you want to generate the documentation. By default, documentation will be created for all the classes.
5. Select the location where you want to save your JavaDoc by clicking on the **Output directory**.
6. Click on **Ok**, and your documentation will be saved to your specified location.



This is all for this tutorial, and in the next tutorial, we will deal with the tags for documenting classes in JavaDocs.

Code as described/written in the video :

```
package com.company;

public class cwh_105_javadoc {
    static void foo(){
        System.out.println("Good Morning bro!");
    }

    static void foo(int a){
        System.out.println("Good morning " + a + " bro!");
    }

    static void foo(int a, int b){
        System.out.println("Good morning " + a + " bro!");
        System.out.println("Good morning " + b + " bro!");
    }

    static void foo(int a, int b, int c){
        System.out.println("Good morning " + a + " bro!");
        System.out.println("Good morning " + b + " bro!");
    }

    static void change(int a){
        a = 98;
    }

    static void change2(int [] arr){
        arr[0] = 98;
    }

    static void tellJoke(){
        System.out.println("I invented a new word!\n" +
            "Plagiarism!");
    }
}
```



```
public static void main(String[] args) {
    // tellJoke();

    // Case 1: Changing the Integer
    //int x = 45;
    //change(x);
    //System.out.println("The value of x after running change is: " + x);

    // Case 1: Changing the Array
    // int [] marks = {52, 73, 77, 89, 98, 94};
    // change2(marks);
    // System.out.println("The value of x after running change is: " + marks[0]);

    // Method Overloading
    foo();
    foo(3000);
    foo(3000, 4000);
    // Arguments are actual!

}
}
```

Copy

Javadocs: Tags for Documenting Classes

Below is the list of the JavaDoc tags :

Tag	Syntax	Description
@author	@author name-text	Describes the author of a class.
@version	@version version-number	Adds a "Version" heading which specifies the current version of the release or file.
@since	@since release-date	Adds a "Since" heading that tells about the release date.
@see	@see 	Adds a "See Also" heading that refers to the other element of the documentation.
@return	@return return-description	Adds a "Return" description that tells about the return value of the method.
@param	@param param-description	Provides the information about the method parameters in the "Parameters" section.
@throws	@exception exception-name description	Displays the exception that can be thrown by a method (same as @exception)
{@code}	{@code text}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.
@deprecated	@deprecated deprecatedtext	Adds a "Deprecated" heading indicating that this API should no longer be used.

Comments In JavaDoc :

Like Java programs, we can also include comments in Java documentation for a better understanding. A JavaDoc comment is known as doc comment in general.

Syntax :

```
/**Documentation comment */
```

Copy

Including HTML inside the JavaDoc :

You can include HTML tags in the JavaDoc. Example :

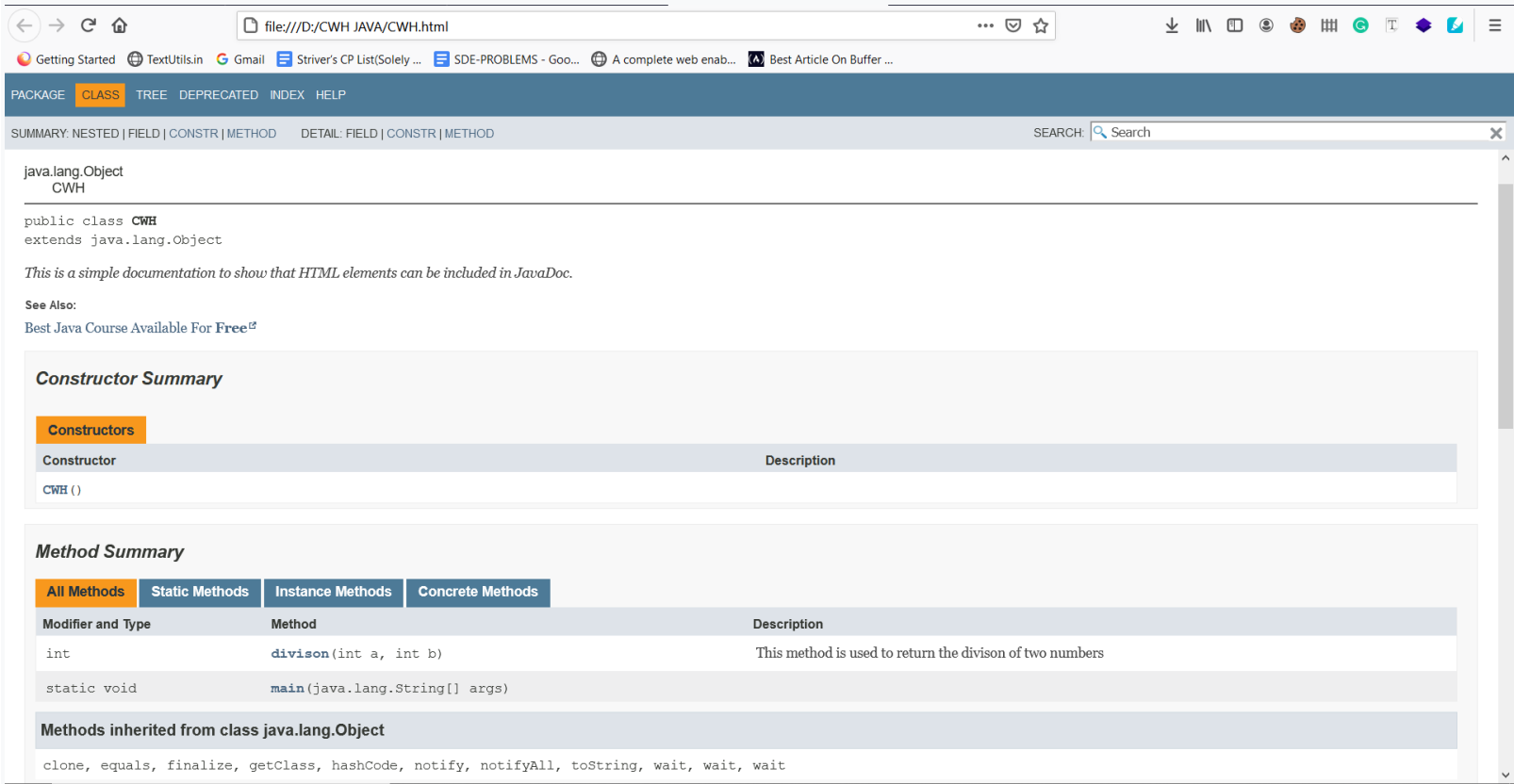
```
/**
 * <i>This is a simple documentation to show that HTML elements can be included in Javadoc.</i>
 * @see <a href="https://www.codewithharry.com/videos/java-tutorials-for-beginners-1">Best Java
Course Available For <b>Free</b></a>
 * */

public class CWH {
    /** This method is used to return the division of two numbers
     * @throws ArithmeticException if divided by 0
     * @return Integer
     * @param a First parameter - Integer
     * @param b Second parameter - Integer
     * */
    public int divison(int a, int b){
        return a/b;
    }

    public static void main(String[] args) {
        System.out.println("This is my main method");
    }
}
```

Copy

Screenshot of the generated Javadoc :



Code as described/written in the video :

```
package com.company;

/**
 * This class is to demonstrate what javadoc is and how it is used in the java industry
 * This is <i>italic</i> word<p>this is a new paragraph</p>
 * @author Harry (CodeWithHarry)
 * @version 0.1
 * @since 2002
 * @see <a href="https://docs.oracle.com/en/java/javase/14/docs/api/index.html"
target="_blank">Java Docs</a>
 */
```

```
public class cwh_106_javadoc {

    public void add(int a, int b){
        System.out.println("The sum is: " + a+b);
    }

    public static void main(String[] args) {
        System.out.println("This is my main method");
    }

}
```

Copy

Javadocs: Method Tags For Generating java Documentation

JavaDoc tags :

Tag	Syntax	Description
@author	@author name-text	Describes the author of a class.
@version	@version version-number	Adds a "Version" heading which specifies the current version of the release or file.
@since	@since release-date	Adds a "Since" heading that tells about the release date.
@see	@see 	Adds a "See Also" heading that refers to the other element of the documentation.
@return	@return return-description	Adds a "Return" description that tells about the return value of the method.
@param	@param param-description	Provides the information about the method parameters in the "Parameters" section.
@throws	@exception exception-name description	Displays the exception that can be thrown by a method (same as @exception)
{@code}	{@code text}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.
@deprecated	@deprecated deprecatedtext	Adds a "Deprecated" heading indicating that this API should no longer be used.

Code as described/written in the video :

```
package com.company;

/**
 * This is a good class
 */
public class cwh_107_method_tags {

    /**
     *
     * @param args These are arguments supplied to the command line
     */
    public static void main(String[] args) {
        System.out.println("I am main method");
    }

    /**
     * Hello this is a method and this is the most beautiful method of this class
     * @param i This is the first number to add
     * @param j This is the second number to add
     * @return Sum of two numbers as an integer
     * @throws Exception if i is 0
     * @deprecated This method is deprecated please use + Operator
     */
}
```

```
public int add(int i, int j) throws Exception{
    if(i==0){
        throw new Exception();
    }
    int c;
    c= i+ j;
    return c;
}
}
```

Copy

Annotations in Java

- Annotations provides metadata to class/methods.
- Annotations start with '@'.
- Annotations are helpful for detecting erros. Example : @override annotations will make sure that there are no typos while overriding a method.

Important Annotations In Java :

1. @Override:

- This annotation makes sure that the sub class method is successfully overriding the parent class method.
- While overriding a class, there is a chance of typing errors or spelling mistakes. In such cases, the method will not get overridden and you will get an error.
- Override exception helps us to encounter such situtations by extracting a warning from the compiler.

Example :

```
class KeyPadPhone{
    void sendMessage(){
        System.out.println("Text message sent!");
    }
}

class AndroidPhone extends KeyPadPhone{
    @Override
    void sendMessage(){
        System.out.println("Message sent via WhatsApp!");
    }
}

public class CWH{
    public static void main(String args[]){
        AndroidPhone Samsung = new AndroidPhone();
        Samsung.sendMessage();
    }
}
```

Copy

Output :

```
Message sent via WhatsApp!
```

Copy

2. @Deprecated :

- This annotation is used to mark a deprecated method.
- If developer uses the deprecated method then the compiler generated a warning.

- There high chance of removal of deprecated methods in future versions therefore it is better to not use them.

Example :

```
class KeyPadPhone{
    @Deprecated
    void sendMessage(){
        System.out.println("Text message sent!");
    }
}

class AndroidPhone extends KeyPadPhone{
    @Override
    void sendMessage(){
        System.out.println("Message sent via WhatsApp!");
    }
}

public class CWH{
    public static void main(String args[]){
        AndroidPhone Samsung = new AndroidPhone();
        Samsung.sendMessage();
    }
}
```

Copy

Build Output :

```
java: sendMessage() in KeyPadPhone has been deprecated
```

Copy

3. @SupressWarnings :

- This annotation helps us to supress some warnings that are being generated by compiler.

Example :

```
class KeyPadPhone{
    @Deprecated
    void sendMessage(){
        System.out.println("Text message sent!");
    }
}

class AndroidPhone extends KeyPadPhone{
    @Override
    void sendMessage(){
        System.out.println("Message sent via WhatsApp!");
    }
}

public class CWH{
    public static void main(String args[]){
        @SuppressWarnings("deprecation")
        AndroidPhone Samsung = new AndroidPhone();
        Samsung.sendMessage();
    }
}
```

```
}
```

Copy

Build Output :

```
This time no warning is generated because we've suppressed the deprecation warning.
```

Copy

4. @FunctionalInterface :

- An interface which contains only one abstract method is known as functional interface.
- @FunctionalInterface annotation helps us to make sure that a functional interface is not having more than one abstract method.

Example :

```
@FunctionalInterface
interface myFunctionalInterface {
    void method1();
    void methodd2();
}

public class CWH{
    public static void main(String args[]){
        System.out.println("Functional interface annotation");
    }
}
```

Copy

Output :

```
java: Unexpected @FunctionalInterface annotation
    myFunctionalInterface is not a functional interface
    multiple non-overriding abstract methods found in interface myFunctionalInterface
```

Copy

The above code generates error because the **myFunctionalInterface** is containing more than one abstract method.

Code as described/written in the video :

```
package com.company;

@FunctionalInterface
interface myFunctionalInteface{
    void thisMethod();
    // void thisMethod2();
}

class NewPhone extends Phone{
    @Override
    public void showTime(){
        System.out.println("Time is 8PM");
    }
    @Deprecated
    public int sum(int a, int b){
        return a+b;
    }
}
```

```
public class cwh_108_java_annotations {
    @SuppressWarnings("deprecation")
    public static void main(String[] args) {
        NewPhone phone = new NewPhone();
        phone.showTime();
        phone.sum(5, 6);
    }
}
```

Copy

Java Anonymous Classes & Lambda Expressions

- Anonymous class is nothing but a class without any name.
- They are used to override a class method or interface.
- Anonymous classes in Java help us to write more concise and readable code.

Syntax :

```
// Demo can be interface or abstract class.
Demo t = new Demol()
{
    // data members and methods
    public void Demo_method()
    {
        .....
        .....
    }
};
```

Copy

How Anonymous class helps us to write concise code?

Take a look at the code given below :

```
@FunctionalInterface
interface Animal{
    void bark();
}

class Dog implements Animal{

    @Override
    public void bark() {
        System.out.println("Dog barks!");
    }
}

class AnonDemo{
    public static void main(String[] args) {
        Dog Bruno = new Dog();
        Bruno.bark();
    }
}
```

Copy

- In the above example, **Animal** is a FunctionalInterface containing a **bark()** method inside it.
- Class **Dog** implements the **Animal** interface and overrides the **bark()** method.
- **Bruno** is an object of Dog class on which we are running the **bark()** method. Now, let's see the output of the above code :

Output :

Dog barks!

Copy

The same output can be generated without creating the **Dog** class. This is the scenario where the Anonymous class comes into the picture. With the help of the Anonymous class, we can declare and instantiate a class at the same time. Let's see how it is done :

```
@FunctionalInterface
interface Animal{
    void bark();
}

class AnonDemo{
    public static void main(String[] args) {
        Animal Bruno = new Animal() {
            @Override
            public void bark() {
                System.out.println("Dog barks!");
            }
        };
        Bruno.bark();
    }
}
```

Copy

In the above code, we've created the **Bruno** object by referencing the **Animal** interface. So, that's how we have overridden the **bark()** method without creating any separate class.

Ways to create an Anonymous Java class :

The Anonymous class in Java can be created by two ways :

1. By extending a class
2. By implementing an interface

Let's take an example for both ways listed above.

- By extending a class :

```
• abstract class Vehicle{
•     abstract void drive();
• }
•
• class AnonDemoByClass{
•     public static void main(String[] args) {
•         Vehicle car = new Vehicle() {
•
•             @Override
•             void drive() {
•                 System.out.println("I'm driving a car.");
•             }
•         };
•         car.drive();
•     }
• }
```

}

Copy

Output :

I'm driving a car.

Copy

- By implementing an interface :

```
@FunctionalInterface
interface Human{
    void walk();
}

class AnonDemo{
    public static void main(String[] args) {
        Human John = new Human() {
            @Override
            public void walk() {
                System.out.println("John walks.");
            }
        };
        John.walk();
    }
}
```

Copy

Output :

```
John walks.
```

Copy

Lambda Expressions :

- Lambda expressions were introduced in Java 8.
- They are similar to methods, but they don't need a name.

Syntax :

```
(parameter1, parameter2) -> { code to be executed }
```

Copy

Take a look at the example given below :

```
@FunctionalInterface
interface LambdaExp{
    void meth1(int a, int b);
}

class LambdaExpDemo{
    public static void main(String[] args) {

        LambdaExp obj =(a,b)->{
            System.out.println("The value of a and b is : "+ a + "," + b);
        };
        obj.meth1(5,10);
    }
}
```

Copy

Output :

```
The value of a and b is : 5,10
```

Copy

Code as described/written in the video :

```
package com.company;

@FunctionalInterface
interface DemoAno{
    void meth1(int a);
    // void meth2();
}

//
//class HarryFunc implements DemoAno{
//    @Override
//    public void meth1() {
//        System.out.println("This is method 1");
//    }
//}

//class AnonyDemo implements DemoAno{
//    public void display(){
//        System.out.println("Hello");
//    }
//
//    @Override
//    public void meth1() {
//        System.out.println("I am meth1");
//    }
//
//    @Override
//    public void meth2() {
//        System.out.println("I am meth2");
//    }
//}

public class cwh_109_lambda {
    public static void main(String[] args) {
        // DemoAno obj = new AnonyDemo();
        // obj.meth1();

        // Anonymous Class
        // DemoAno obj = new DemoAno() {
        //     @Override
        //     public void meth1() {
        //         System.out.println("I am meth1");
        //     }
        //
        //     @Override
        //     public void meth2() {
        //         System.out.println("I am meth2");
        //     }
        // };
        // obj.meth1();

        // Lambda Expressions

        // DemoAno obj = new HarryFunc();
    }
}
```

```
//      obj.meth1();  
  
      DemoAno obj = (a)->{System.out.println("I am method 1 from this lambda " + a);};  
      obj.meth1(6);  
  }  
}
```

Copy

Java Generics

- Introduced from JDK 5.0 onwards.
- The Java Generics helps us to deal with the compiler time type-safety.
- With the help of the Generics, we can write a single method and call it with different argument types(integer, strings, etc.).

Advantages of Generics :

1. Bugs can be detected at compile-time:

- While developing any application or program, it is always better to catch the bug/problem at the compile-time instead of runtime so that we can provide a smooth experience to the user.
- Let's take an example to see how Java Generics helps us to detect problems at compile-time:

Example :

```
import java.util.ArrayList;  
  
public class CWH {  
    public static void main(String[] args) {  
        //      Without Java Generics :  
  
        ArrayList myArrayList = new ArrayList();  
        myArrayList.add(10); //Integer value  
        myArrayList.add("Harry Bhai!"); //String value  
        myArrayList.add(20.4); //Double value  
        System.out.println(myArrayList);  
  
    }  
}
```

Copy

Output :

```
[10, Harry Bhai!, 20.4]
```

Copy

In the above code, notice that we can store any type of object in a collection without Generics. But, this is not the case with the Generics. It allows us to store only one type of object. Take a look at the example given below :

```
import java.util.ArrayList;  
  
public class CWH {  
    public static void main(String[] args) {  
        //      With Java Generics :  
  
        ArrayList<Integer> myArrayList = new ArrayList();  
        myArrayList.add(10); //Integer value  
        myArrayList.add("Harry Bhai!"); //String value  
        myArrayList.add(20.4); //Double value
```

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

    }
}
```

Copy

Output :

```
java: incompatible types: java.lang.String cannot be converted to java.lang.Integer
java: incompatible types: double cannot be converted to java.lang.Integer
```

Copy

The same code produces the **Incompatible type** error because we can only store the integer object type.

2. Type-casting not required :

- Let's suppose you created an ArrayList(without using Generics), and you want to store the value at index 0 into an integer variable named "x." Are you allowed to do this in Java? The answer is a big NO! This is because the ArrayList returns an object, but we're storing the value in an integer variable. In such cases, we need to type-cast the object into our desired data type. But, if we use Generics, then there is no need to typecast. Take a look at the below example to get a better understanding :

```
import java.util.ArrayList;

public class CWH {
    public static void main(String[] args) {
        // Without Java Generics :

        ArrayList myArrayList = new ArrayList();
        myArrayList.add(10); //Integer value
        myArrayList.add("Harry Bhai!"); //String value
        myArrayList.add(20.4); //Double value

        int x = myArrayList.get(0);
        System.out.println(x);
    }
}
```

```
}
```

Copy

Output :

```
java: incompatible types: java.lang.Object cannot be converted to int
```

Copy

The above code produces an error because we've not typecasted the object into the integer type. Now, let's typecast and see the results :

```
import java.util.ArrayList;

public class CWH {
    public static void main(String[] args) {
        // Without Java Generics :

        ArrayList myArrayList = new ArrayList();
        myArrayList.add(10); //Integer value
        myArrayList.add("Harry Bhai!"); //String value
```

```
        myArrayList.add(20.4); //Double value

        int x = (int) myArrayList.get(0); //b=object typecasted into integer
        System.out.println(x);

    }

}
```

Copy

Output :

```
10
```

Copy

Now, let's see how we can get the desired results with the help of the Generics :

```
import java.util.ArrayList;

public class CWH {
    public static void main(String[] args) {
        //      With Java Generics :

        ArrayList<Integer> myArrayList = new ArrayList();
        myArrayList.add(10);
        myArrayList.add(20);
        myArrayList.add(30);
        myArrayList.add(40);

        int x = myArrayList.get(0);
        System.out.println(x);

    }
}
```

Copy

Output :

```
10
```

Copy

Code as described/written in the video :

```
package com.company;

import java.util.ArrayList;
import java.util.Scanner;

class MyGeneric<T1, T2>{
    int val;
    private T1 t1;
    private T2 t2;

    public MyGeneric(int val, T1 t1, T2 t2) {
```

```

        this.val = val;
        this.t1 = t1;
        this.t2= t2;
    }

    public T2 getT2() {
        return t2;
    }

    public void setT2(T2 t2) {
        this.t2 = t2;
    }

    public int getVal() {
        return val;
    }

    public void setVal(int val) {
        this.val = val;
    }

    public T1 getT1() {
        return t1;
    }

    public void setT1(T1 t1) {
        this.t1 = t1;
    }
}

public class cwh_110_generics {
    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList();
//        ArrayList<int> arrayList = new ArrayList(); -- this will produce an error
//        arrayList.add("str1");
        arrayList.add(54);
        arrayList.add(643);
//        arrayList.add(new Scanner(System.in));

        int a = (int) arrayList.get(0);
//        System.out.println(a);
        MyGeneric<String, Integer> g1 = new MyGeneric(23, "MyString is my string ", 45);
        String str = g1.getT1();
        Integer int1 = g1.getT2();
        System.out.println(str + int1);
    }
}

```

Copy

File Handling in Java

- File handling is a crucial part of any programming language.
- In Java, file handling is done with the help of the File class of the java.io package.

Common file handling operations :

1. Creating a new file.
2. Writing in a file.
3. Reading an existing file.
4. Deleting a file.

To perform any of the above operations on a file in Java, we need to create an object of the **File** class as shown in the below code:

```
import java.io.File; // Importing the File class

File obj = new File("filename.txt"); // Specify the name of the file
```

Copy

Now, let's see how we can perform the above operation on a file in Java.

1. Creating a new file :

- o **createNewFile()** method is used to create a new file. Take a look at the below example :

```
import java.io.File;
import java.io.FileWriter;

public class CWH {
    public static void main(String[] args) {

        File myFile = new File("CWH_file1.txt");
        try {
            myFile.createNewFile();
            System.out.println("File created successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Copy

Output :

```
File created successfully.
```

Copy

2. Writing to a file :

- o **FileWriter** class is used with its **write()** method to write some content in a file.
- o Do not forget to use the **close()** method when you're finished writing to a file. Take a look at the below example :

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class CWH {
    public static void main(String[] args) {

        File myFile = new File("CWH_file1.txt");
        try {
            FileWriter fileWriter = new FileWriter("CWH_file1.txt");
            fileWriter.write("CodeWithHarry is one step solution for your all
programming problems.\nKeep learning, Keep coding!");
            fileWriter.close();
        }
    }
}
```

```
○         } catch (IOException e) {  
○             e.printStackTrace();  
○         }  
○  
○  
○     }  
○ }
```

```
}
```

Copy

Output of the CWH_file1.txt :

```
CodeWithHarry is one step solution for your all programming problems.  
Keep learning, Keep coding!
```

Copy

3. Reading a file :

- The **Scanner** class is used to read a file.
- It is important to enclose the method in a try-catch block to handle the IOException.

Example :

```
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
public class CWH {  
    public static void main(String[] args) {  
  
        File myFile = new File("CWH_file1.txt");  
        try {  
            Scanner sc = new Scanner(myFile);  
            while(sc.hasNextLine()){  
                String line = sc.nextLine();  
                System.out.println(line);  
            }  
            sc.close();  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Copy

Output :

```
CodeWithHarry is one step solution for your all programming problems.  
Keep learning, Keep coding!
```

Copy

4. Deleting a file :

- The **delete()** method is used to delete a file in Java.

Example :

```
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;
```



```
public class CWH {  
    public static void main(String[] args) {  
  
        File myFile = new File("CWH_file1.txt");  
        if(myFile.delete()){  
            System.out.println("I have deleted: " + myFile.getName());  
        }  
        else{  
            System.out.println("Some problem occurred while deleting the file");  
        }  
  
    }  
}
```

Copy

Output :

```
I have deleted: CWH_file1.txt
```

Copy

Code as described/written in the video :

```
package com.company;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Scanner;  
  
public class cwh_111_file {  
    public static void main(String[] args) {  
  
        // Code to create a new file  
        /*  
        File myFile = new File("cwh111file.txt");  
        try {  
            myFile.createNewFile();  
        } catch (IOException e) {  
            System.out.println("Unable to create this file");  
            e.printStackTrace();  
        }  
  
        // Code to write to a file  
        try {  
            FileWriter fileWriter = new FileWriter("cwh111file.txt");  
            fileWriter.write("This is our first file from this java course\nOkay now bye");  
            fileWriter.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
        // Reading a file
```

```

        File myFile = new File("cwh111file.txt");
        try {
            Scanner sc = new Scanner(myFile);
            while(sc.hasNextLine()){
                String line = sc.nextLine();
                System.out.println(line);
            }
            sc.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        */
        // Deleting a file
        File myFile = new File("cwh111file.txt");
        if(myFile.delete()){
            System.out.println("I have deleted: " + myFile.getName());
        }
        else{
            System.out.println("Some problem occurred while deleting the file");
        }

    }
}

```

Copy

[Download Chapter 16 Notes here](#)

Advanced Java 2 - Practice Set

Question 1: Create a class and a method with deprecated annotation. What is its effect on program execution?

Answer 1: We discussed the deprecated annotation in [Annotations in Java tutorial](#). Below is the required program :

```

class MyDeprecated{
    @Deprecated
    void meth1(){
        System.out.println("I am method 1");
    }
}

public class CWH {
    public static void main(String[] args) {
        MyDeprecated d = new MyDeprecated();
        d.meth1();
    }
}

```

Copy

Output :

```
I am method 1
```

Copy

There is no as such special effect on the program of deprecated annotation. The only thing is that compiler generated a warning if we use deprecated method or class in our program.

Question 2: Suppress the warning generated in question number 2.

Answer 2: We saw how compiler-generated warnings could be suppressed by using [@SuppressWarnings annotation](#). The required program is given below:

```
class MyDeprecated{
    @Deprecated
    void meth1(){
        System.out.println("I am method 1");
    }
}

public class CWH {
    public static void main(String[] args) {
        @SuppressWarnings("deprecation")
        MyDeprecated d = new MyDeprecated();
        d.meth1();
    }
}
```

Copy

Question 3: Create an interface and generate an instance from it.

Answer 3:

```
interface MyInt{
    void display();
}

public class CWH {
    public static void main(String[] args) {
        MyInt i = () -> System.out.println("I am display");
        i.display();
    }
}
```

Copy

Output :

```
I am display
```

Copy

Question 4: Write a Java program to generate a multiplication table of a given number and write it to a file.

Answer 4: The following program saves the multiplication table of 19 into a file named "MultiplicationTable.txt".

```
import java.io.FileWriter;
import java.io.IOException;

public class CWH {
    public static void main(String[] args) {
        int i = 19;
        String table = "";
        for (int j = 0; j < 10; j++) {
            table += i + "X" + (j+1) + "=" + i*(j+1);
            table += "\n";
        }
        try {
            FileWriter fileWriter = new FileWriter("MultiplicationTable.txt");
```

```

        fileWriter.write(table);
        fileWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Copy

Code as described/written in the video :

```

package com.company;

import java.io.FileWriter;
import java.io.IOException;

class MyDeprecated{
    @Deprecated
    void meth1(){
        System.out.println("I am method 1");
    }
}

interface MyInt{
    void display();
}

public class cwh_112 {

    public static void main(String[] args) {
//        MyDeprecated d = new MyDeprecated();
//        d.meth1();
//        MyInt i = () -> System.out.println("I am display");

        int i = 19;
        String table = "";
        for (int j = 0; j < 10; j++) {
            table += i + "X" + (j+1) + "=" + i*(j+1);
            table += "\n";
        }
        try {
            FileWriter fileWriter = new FileWriter("MultiplicationTable.txt");
            fileWriter.write(table);
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Copy

Exercise 7: Solutions + Shoutouts

Below is the solution of exercise number 7 that I gave you all in [tutorial 104](#) :

```

package com.company;

```

```

import java.util.ArrayList;

/*
Create a library management system which is capable of issuing books to the students.
Book should have info like:
1. Book name
2. Book Author
3. Issued to
4. Issued on
User should be able to add books, return issued books, issue books
Assume that all the users are registered with their names in the central database
*/
class Book{
    public String name, author;

    public Book(String name, String author) {
        this.name = name;
        this.author = author;
    }

    @Override
    public String toString() {
        return "Book{" +
            "name='" + name + '\'' +
            ", author='" + author + '\'' +
            '}';
    }
}

class MyLibrary{
    public ArrayList<Book> books;
    public MyLibrary(ArrayList<Book> books) {
        this.books = books;
    }
    public void addBook(Book book){
        System.out.println("The book has been added to the library");
        this.books.add(book);
    }
    public void issueBook(Book book, String issued_to){
        System.out.println("The book has been issued from the library to " + issued_to);
        this.books.remove(book);
    }
    public void returnBook(Book b){
        System.out.println("The book has been returned");
        this.books.add(b);
    }
}

public class cwh_113_ex7sol {
    public static void main(String[] args) {
        // Exercise 7 Solution
        ArrayList<Book> bk = new ArrayList<>();
        Book b1 = new Book("Algorithms", "CLRS");
    }
}

```

```
        bk.add(b1);

        Book b2 = new Book("Algorithms2", "CLRS2");
        bk.add(b2);

        Book b3 = new Book("Algorithms3", "CLRS3");
        bk.add(b3);

        Book b4 = new Book("Algorithms4", "CLRS4");
        bk.add(b4);
        MyLibrary l = new MyLibrary(bk);
        l.addBook(new Book("algo4", "myAuthor"));
        System.out.println(l.books);
        l.issueBook(b3, "Harry");
        System.out.println(l.books);

    }
}
```

Copy