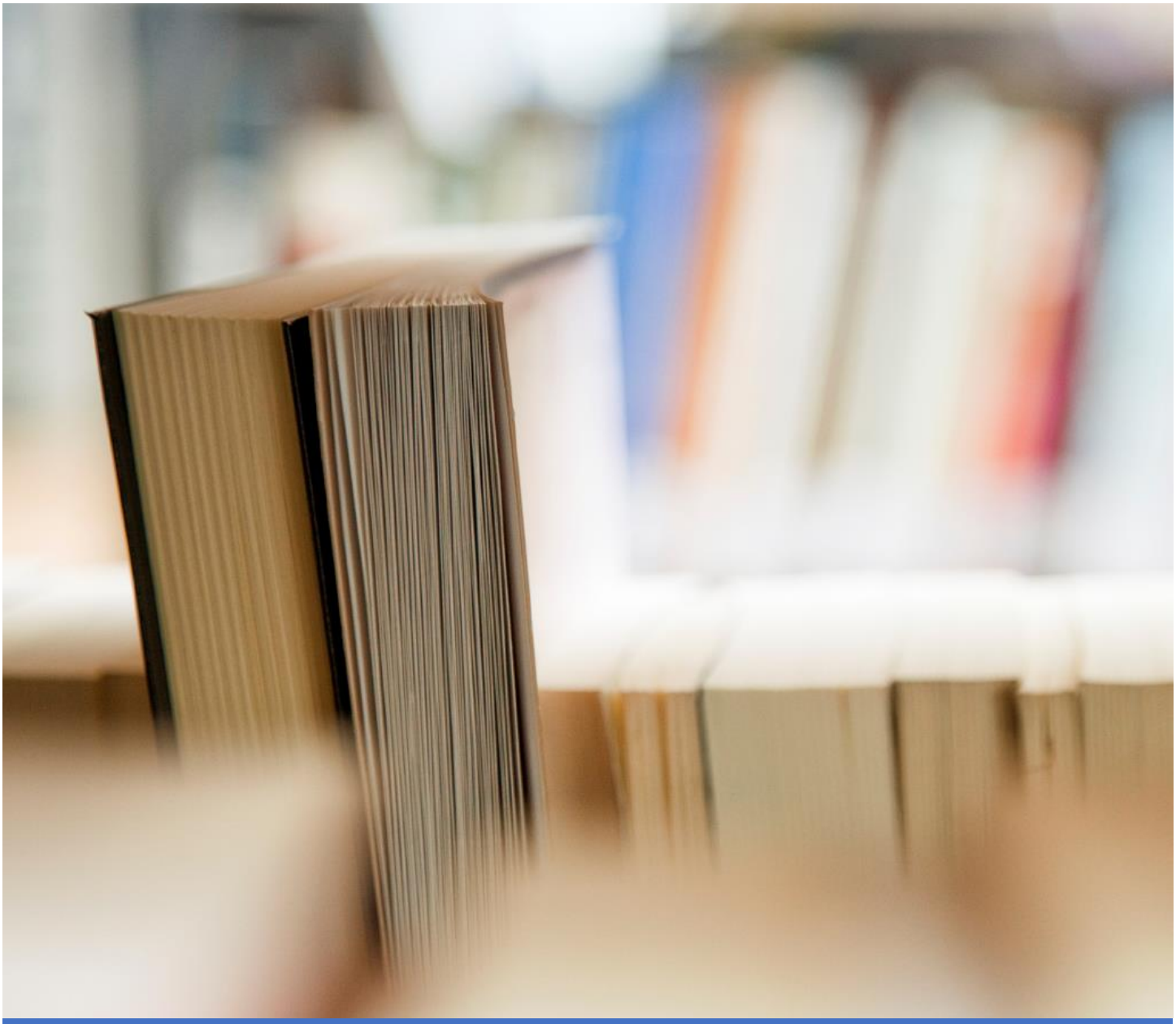


Core Java

JEEVAN KUMAR SAHU

Syllabus

Part -1

1. What is Java?
2. Programming language and type
3. Why Java features
4. Why java is the platform-independent
5. JDK, JRE, JVM
6. File structure in the Windows operating system and command prompt
7. Structure of Java program
8. Printing statement
9. Tokens and Types
10. Variable & Data types
11. Operators
12. Control Statement
    1. Conditional statement
        a. Branching control statement
        b. Looping control statement
    2. Unconditional Statement
        a. Break
        b. Return etc
13. Method and Types
14. Method call
15. Hard code and Dynamic Reading

Part – 2

1. Static and static members
2. Non-static and non-static members
3. Concept and principal of oops
    a. Class
    b. Object
    c. Encapsulation
    d. Inheritance
    e. Polymorphism
    f. Abstraction

Part - 3

1. Object class
2. String class
3. Array
4. Structure of Java Source file
5. Package
6. Access modifier
7. Exception handling
8. Wrapper class
9. Collection Framework
10. Multi-Threading
11. File handling
    A. Threads
    B. Regular Expression
    C. Lambda Expression and Function Interface

Part - I

What is Software?

- It is a set of programs or a collection of programs. Which is used to solve the real-world problem.

What is a Program?

- It is a set of instructions or statements.

What is instruction or statement?

- The sentence we use in the programming language is an instruction or statement.

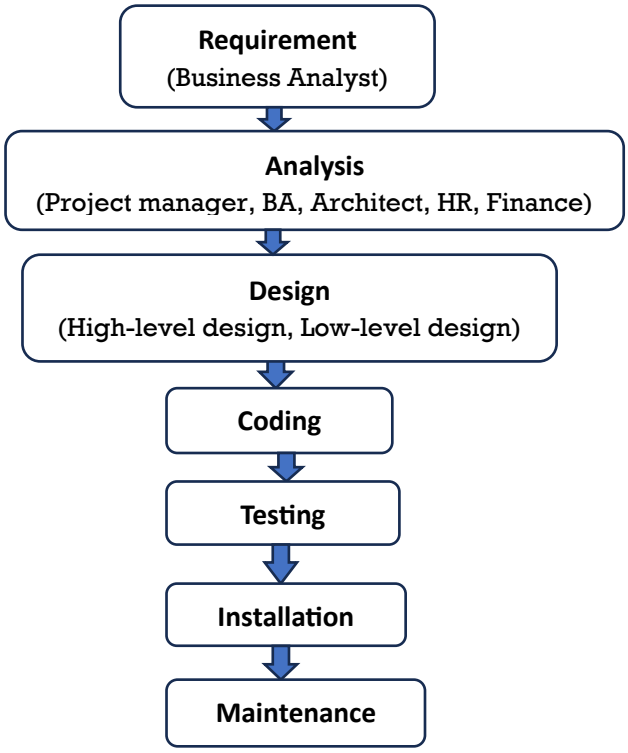
What is programming?

- The skill of writing a program in programming.

What is a programming language?

- It is a medium or language used to give instructions to the machine to perform a specific task.

SDLC (Software Development life cycle)



What is a programming language?

- It is a medium or language used to instruct the machine or computer to perform specific tasks, known as programming language.

- Based on the approach to solving the real-world problem it is as follows
  a) Object-oriented language
  b) Procedure-oriented language
  c) Functional-oriented language
  d) Logical-oriented language
  e) Scripting-oriented language

- Based on the level of the programming language it is 3 types
  a) Machine-level programming language / Low-level language
  b) Mid-level programming language / Assembly level programming language
  c) High-level programming language

a) Machine-level programming language / Low-level language

- The language which is easily readable understandable executable directly by the machine is known as low-level language. It is also known as machine-level language.
  For, ex – 0,1 (binary language)

  The disadvantage of low-level language
  - It is challenging to understand the programmer directly.
    For ex- 5C is not understand by machine

b) Mid-level programming language / Assembly level programming language

- The language which having some pre-defined words such as add, sub, mul, div, etc for some specific tasks is known as mid-level.
- There are pre-defined words known as Mnemonics.

  Here,
  Add word use for addition.
  Sub word used for subtraction.
  Mul word used for multiplication.

  These are pre define words that are not directly understandable by machines. therefore to make machines understandable. where use the translation called to assembly to convert mid-level into machine-level language.
  For ex- Instruction, the set was used in 8085,8086 architectures.

  The disadvantage of mid-level language
  - It has limited pre-defined words to do limited tasks.
  - It does not have a control statement to control the flow of execution.
  - Such as a looping statement.

c) High-level language

- The language which is easily readable understandable and easy to instruct the machine by the programmer is known as high-level language.
  Ex – Java, C, C++, Python, etc.

- The high-level programming language is not directly understandable by machines to make machines understandable using the translate called a compiler, interpreters to convert high-level to low-level language.
- Every high-level language has its own compiler interpreter.
- A high-level language is used to create some. Applications or software.

The file structure in Windows

In Windows hard disk can be divided into multiple portions and each portion is known as a drive.
Each drive is represented by one name and colon (:)
E.g.- C:, D:, E:
The drive consists of some folders. Folders are also known as directories.
There are 2 types of folders as follows.

I. User home directory
II. Current working directory / Working directory

I. User home directory

- The folder/directory created with a "user name" inside the user's directory of **C** drive is called user home directory.
- As soon as we open the command prompt, we are always inside the user home directory.

II. Current working directory / Working directory

The folder/directory which is currently in use is known as the current working Directory / Working Directory.

Command Prompt

- This is one of the user interfaces to give instructions to the machine by giving the command.
- This is also known as a command user interface (CUI).

Some important commands of the command prompt: -

Dir: -

- It is used to display the contents of the current directory.

Cls: -

- It is used to clear the screen of the command prompt.

Mkdir or Md (Make directory): -

- It creates a new folder or directory. We should pass the folder name as input to this command.
- **Single word folder name**: - for single word directory name you can directly pass the name.
  E.X. – md F1
- **Multi-word folder name**: - for multi-world folder creating you have to enclose the complete folder name by double quotes ("…. ") and pass it to the command.
  Ex- md "java folder"
- You can create multiple folders at a time to give the multiple folders' names to the **md** or **mkdir** command.
  Ex – mkdir f1, f2, "sql note", F6

Rmdir/Rd -

- It is used to remove the folder or directory. We should pass the folder name as input to the command.
  Ex 1 – rd f1 Or Rmdir f2
  Ex 2 – rd f1, f2, f3

Rename/Ren –

- It is used to rename the folder or directory; we should pass the old folder name by the new folder name as input to this command.
  Syntax – rename/ren old_folder_name new_folder_name
  Ex – rename f6 d6

Cd: -

- It is used to change the directory.
  - If you want to move in the forward directory, we should pass directory name as the input to this command. Such as directory name D:\java\jjf-cjd-a3>
    Ex – cd Jeewan, result is D:\java\jjf-cjd-a3\Jeewan>
  - If you want to move the backward directory, we should pass the double dot(..) as the input to this command. Such as directory name D:\java\jjf-cjd-a3\Jeewan\Part-1\session_1>
    Ex – cd .. , result is D:\java\jjf-cjd-a3\Jeewan\Part-1>

Javac: -

- This command is used to compile the Java source file.
- We should pass the source file name as an input to this command including the extension as java.
  Ex - D:\java\jjf-cjd-a3\Jeewan\Part-1\session_1>javac prog1.java

Java: -

- This command is used to execute the Java class file.
- We should pass the only class name as an input to this command but not any extension.

Ex - D:\java\jjf-cjd-a3\Jeewan\Part-1\session_1>java prog1

Note –

* Command prompt is not case-sensitive. There are two types of case

* Upper case (Capital Letter) or Lower case (Small letter) means you can use any command in upper case as well as lower case.

Java

JAVA is a high-level and object-oriented programming language. It is not easily understandable by the machine. So, its needs to change high level programming language to machine understandable programming language by the help of compiler & interpreter.

- It is developed by James Gosling and his team sun microsystem in the year 1995
- The father of Java is James Gosling.
- Now it is tack-care by Oracle 2010.
- The first name of Java was OAK.
- The current version of Java 21's concerning September 2023.
- Java is the most popular, object-oriented, widely used programming language and platform that is utilized for Android development, web development, artificial intelligence, cloud applications, and much more. So, mastering this gives you great opportunities in bigger organizations.

Edition of Java

1. J2SE/JSE (Java standard edition)
2. J2EE/JEE (Java enterprise edition)
3. J2ME/JME (Java micro edition)

1. J2SE/JSE

- It stands for java standard edition. It is used to develop stand-alone applications.

Stand-alone application

- An application that does not depend on a server, as well as network, is known as a stand-alone application.
  E.g. – Notepad, Calculator, etc.

2. J2EE/JEE

- It stands for Java enterprise edition. It is used to develop client-server applications and web applications.

Client-server application

- The application depends on a server as well as a network. It is a known client-server application.
  E.g. – WhatsApp, and Facebook.

Web-application

- A web application is an application that requires a web browser, URL, and network to work or run is known as web-application.
  E.g. – www.gmail.com, www.google.com etc.

3. J2ME/JME

- It stands for java micro edition. It is used to develop micro applications like, android application, Ios applications.
  E.g. – Phone pay, G pay, etc

Why Java / Feature of Java

- Java provides some unique features. Such as ($P_2RIAD_2HOMS_2$).
  1. Platform-independent language
  2. Portable
  3. Robust
  4. Interpreted

5. Architectural Neutral
6. Dynamic
7. Distributed
8. High Performance
9. Object-Oriented Programming
10. Multi-threading
11. Simple
12. Security

What is a platform?

➢ It is the combination of an operating system and a processor.
E.g.- Windows 32-bit, Windows 64-bit, Linux.

What is a bit?

➢ It is the smallest unit is which measures the data or instructions.
- 8 bits make up a byte.
- 1024 bytes make up a kilobyte (KB).
- 1024 kilobytes make up a megabyte (MB).
- 1024 megabytes make up a gigabyte (GB).
- 1024 gigabytes make up a terabyte (TB).

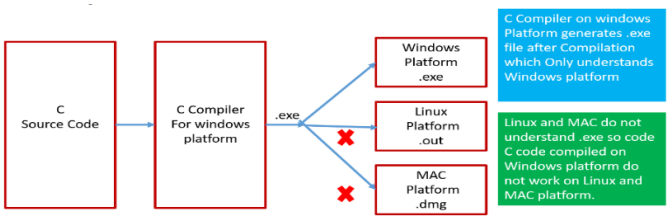
What is platform dependent?

- If one application develops one platform after that it can execute on only the same kind of platform dependent.
- The application is usually developed by C known as platform dependent.

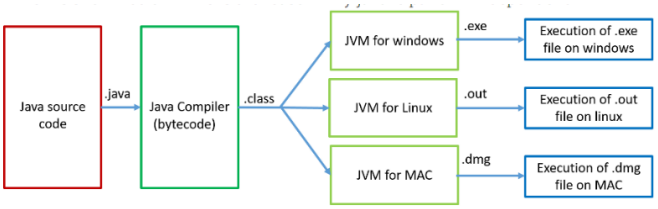
What is the platform In Dependent?

- It is one application developed on one platform and after that, it can be executed on any kind of platform known as platform independent.
- The application is usually developed in Java language and is platform-independent.

C is platform Dependent –



Java is a platform, Independent



Why Java is platform-independent architecture neutral?

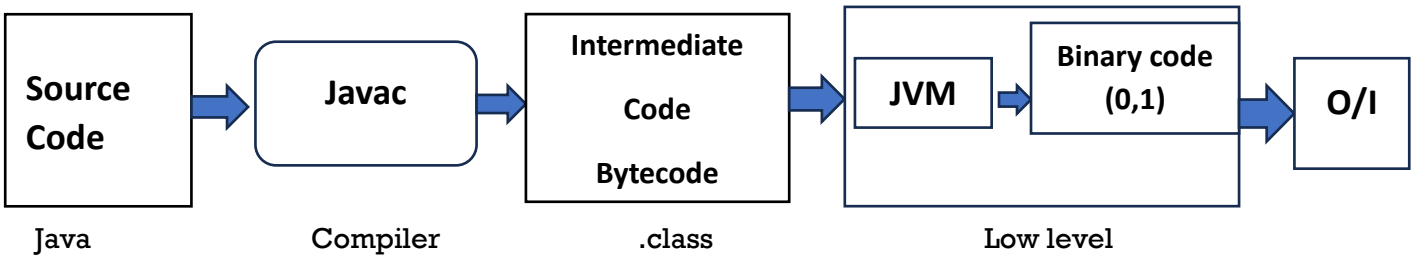
- Java compiler does not convert directly high-level language into low-level language.
- It converts high-level language into one intermediate file or class file. Which is known as a class file or byte code file.
- Class files are only understandable by the Java interpreter called JVM (Java virtual machine).
- Once the class file is generated it can execute on any kind of platform which has JVM on it.
- This architecture makes Java platform independent architectural but JVM Dependent.

- By this architecture Java follows WORA (Write one run anywhere) architecture.

What is WORA architecture?

- WORA stands for write once run anywhere. This means we can write a program in one platform after compilation of the program we can execute it on any of the platforms which have JVM on it.

Conversion of high-level to low-level in Java.



Compiler

- The process of converting Java source file into intermedia file or class file is known as compilation or compiling the program.

Execution or Run

- The process of converting an intermediate file or class file into machine-understandable language is known as execution or running the program.

Java source file

- The file which is written by the programmer in Java language and saved with Java extension is known as a Java source file.

What is a class file

- The file which is created by the Java compiler after successful compilation is known as a class file or byte code file or intermediate file. The extension of the class file is .class.

What is the name of the Java compiler?

Ans – Javac

What is the name of the Java interpreter?

Ans – JVM (Java virtual machine)

Types of Error -

- It is three types
  1. Compile time error
  2. Execution error / Run time error / Exception error
  3. Logical error

1. **Compile time error**
   The time of compiling the program. If we are getting some error because of syntax error are saying that compile time error.
   E.g. – Semicolon (;) expected possible lossy conversion from double to int. SOPER (a), can't find symbols ' a '.

Syntax –

The Grammer and the rule we are using in the programming is known as syntax.

2. **Execution error / Run time error / Exception error**

The time of execution of the program. If we are getting the same error is known as a run time/execution time error.

Ex –

1. FileNotFounException
2. ArithmaticException
3. StackOverFlowError
4. OutOfMemoryError

3. **Logical error**

If in the program there is no compile time error as well as No run time error but after getting the output it not matching with the expected output. This type of error is known as logical error.

Example – If the program requirement is to find out the sum of 5 and 7. And after writing the program there is no compile time error. As well as run output as a 35, when our expected output is 12. This is nothing but a logical error.

JDK, JVM, JRE

JDK

What is JDK?

- JDK (JAVA Development Kit) is a package that consists of JAVA development tools like Java compiler (JAVAC) and JRE for execution.

Developing tools (Java Compiler (JAVAC) + JRE

What is JRE?

- JRE (JAVA Runtime Environment) is an environment that consists of JVM and built-in-class files. which are required for the execution of the JAVA Program.

JVM + Built in class files

What is JVM?

- JVM (Java Virtual Machine) helps to convert Bytecode into current system understandable language (low-level language).

Jit Compiler

JIT Compiler

- JIT (Just In Time) in JAVA is an integral part of the JVM. It helps to improve the performance of Java programs by compiling byte code into native machine code at run time. It accelerates execution performance many times over the previous level.

Steps to create, compile and execute Java file

Step 1: Create a source file

Step 2: Generate class file (compile the program)

Step 3: Execute the class file (execute the program)

To compile the program javac space file name.java

E.g. – javac jeevan.java

To execute the program command is Java class name

Structure of Java programming

Structure of Java program:

Java instructions are always written inside the class.

```
class ClassName
{


}
```

*File name = class Name.Java

Note: -

Every class in Java must have a name, it is known as the class name.

Every class has a block, it is known as a class block.

Note: -

A class in Java can be executed only if the main method is created as follows:

Syntax to create the main method:

```
Public static void main (String[] args)
{
        //statements;
}
```

Note: -

We can create a class without the main method. It is compiled successfully and the class file is generated but we can't execute that class.

Q1, can we execute a Java program without the main method? Justify.

Ans: No

Justification

- We can create a class without the main method. It is compiled successfully and the class file is generated but we can't execute that class.
- Because in Java the program execution starts from the main method, as the main method is not there, we can't execute the program.

Member of the class

In class block, we can create

- Variables
- Methods
- Initializers

These are said to be members of a class.

Variables

- A variable is a container that is used to store data.

Methods

- It is a block of instructions that is used to perform a task.

Initializers

- Initializers are used to execute the start-up instructions.

Printing statement

In Java mainly two types of printing statements are there.

1. Print statement
2. Println statement

1. **Print statement**

- System.out.print(data)
- The print statement is used only to print the data.
- We cannot use the print statement without passing any data, if we use then we will get a compile-time error.

Example:

```
System.out.print("hi");
System.out.print("Developers");
System.out.print("5");
System.out.print("c");
System.out.print("1.14.15");
System.out.print(); // CTE (compile time error)
```

2. Println Statement

- System.out.println (data)
- Println statement is used to print data as well as create a new line.
- We can use the println statement without passing any data, it is just used for printing new lines.

Example:

```
System.out.println("hi");
System.out.println("Developers");
System.out.println(5);
System.out.println('c');
System.out.println(1.1415);
System.out.println();
```

Escape character

These are as a follow-up

1. Backward slash \
2. Backward slash \n
3. Backward slash \t

1. Backward slash (\) -

It is used to hide the original property of a character from the compiler.

Ex –

```
class Prog2
{
        public static void main (String [] args)
        {
                System.out.println("\"");
                System.out.println("\"Hello\"");
        /*      System.out.println(""+"Hello"+""); */
        }
}
```

Output – "
"Hello"

2. Backward slash (\n)

It is used to insert a new line.

Ex –

```
class Prog2
{
        public static void main (String [] args)
        {
                System.out.println("Hello \n world");

        }
} /* Output – Hello
                World */
```

3. **Backward slash (\t)**

It is used to insert a tab space.

Ex –

```
class Prog3
{
        public static void main (String [] args)
        {

                System.out.println("Hello \t world!");

        }
}
```

Output – Hello World

Token

- It is the smallest unit of programming language that is used to compose the instruction or statement and is known as a token.

Types of tokens

There are mainly 6 types of tokens in Java.

1. Keywords
2. Identifiers
3. Separators
4. Comments
5. Literals
6. Operators

Separators

The separators in Java are also known as punctuators. There are mainly 6 types.

1. **Curly Braces {}** - The curly braces denote the starting and ending of a block.

2. **Parentheses ()** - It is used to call the method and declare a method.

3. **Square braces []** - It is used to define array elements.

4. **Dot (.)** - It separates the package name from the sub-packages and class. It also separates a variable or method from a reference variable.

5. **Semicolon (;)** - It is the symbol that can be found at the end of the statements. it separates the two statements.

6. **Comma (,)** - It is used to separate two values statements and parameters.

Comments:

- Comments allow us to specify information about the program inside our Java code. Jaca compiler recognizes these comments as tokens but excludes them. From further processing. The Java compiler treats comments as whitespaces. Java provides the following two types of comments.

1. **Line oriented** - It begins with a pair of forwarding slashes (//).

2. **Block oriented** - It begins with a combination of one forwarding slash and asterisk (/*) and continues until it counts the combination of asterisks and forwarding slash (*/).
   Line Oriented Examples

//Write a Java program to print Hello World.

```
class Prog1
{
	public static void main(String[] args)
	{
		//printing hello world
		System.out.println("Hello world");

	} //end of class
} //end of class
```

//WAJP to print 5 and 9.

```
class Prog2
{
	public static void main(String[] args)
	{
		//printing sum of 5 and 9
		System.out.println("5+9");
		// System.out.println(10);

	} //end of class
} //end of class
```

Block oriented comment

/* Write a Java program to print Hello World. */

```
class Prog1
{
	public static void main (String[] args)
	{
		//printing hello world
		System.out.println("hello world");
		/*
			System.out.println("Block");
```

```
                System.out.println("Oriented");
                System.out.println("Comment");
        */


    } // end of main
} //end of class
```

/*WAJP to print sum of 5 and 9 */

```
class Prog2
{
    public static void main (String[] args)
    {
        //printing sum of 5 and 9
        System.out.println("5+9");
        /*
            System.out.println(10);
            System.out.println(20);
            System.out.println("30+40");
        */


    } // end of main
} //end of class
```

Identifiers

- The name given to components in Java by the programmer is known as identifiers.

List of components: -

- Class
- Method
- Variables
- Interface etc

Rules for identifiers

1. For identifiers, the allowed characters are a to z, A to Z, 0 to 9 underscores (_), and dollar ($).
2. The first letter must be a letter or underscore or dollar.
3. Identifiers should never start with a number but can have a number in it.
4. Identifiers should not have special characters except underscore (_) and dollar $
5. Character space is not allowed in identifiers.
6. We can't use keywords as an identifier.

Note: -

A programmer should follow the rules for an identifier. If the programmer won't follow the rules, then the compiler will give the compile time error.

Conventions

- The coding or industrial standards to be followed by the programmer are known as conventions.

Note: -

- Compiler doesn't validate the convention therefore if conventions are not followed then we won't get a compile-time error.
- It is highly recommended to follow the convention.

Conventions for class

Single word -

- The first letter should be in upper case remaining in lower case.

  Examples – Addition, Calculator, Sum etc…

Multi words -

- The first letter character of every word should be in upper case remaining in lower case.

  Example – SquareRoot, PowerOfDigit, FactoralDigits. Etc.

Convention for variables

Single word -

- All letters should be in lowercase.

  Ex – addition, calculator, sum, etc

Multi words -

- All characters of every word should be in lower case and to separate the words use underscore (_).

  Ex – square_root, power_of_digit, factor_digits, etc.

Convention for methods

Single word -

- The all letters should be in lowercase.

  Ex - print(), println(), addition(), run(), sum(), etc…

Multi words -

- The first character of 1st word should be in lower case and other words 1st letter should be upper case.

  Ex – squareRoot(), poewerOfDigit(), factorialOfDigits(), etc….

Keywords

- A predefined word that the java compiler can understand is known as a keyword.
- Every keyword in Java is associated with a specific task.
- A programmer can't change the meaning of a keyword (can't modify the associated task).

Example –

We have 53 keywords in Java with respect to JDK 1.8.

Class, public, static void etc.,

Rule – Keywords are always written in lower case.

Class related keywords

Class	Interface	Extends

Implements	Package	Import

Object Related keywords

New	Super
Instance of	This

Keywords for datatypes

Byte	Char	Private	Strictfp
Short	Boolean	Static	Transient
Int	Keywords for modifier	Final	Volatile
Long		Abstract	
Float	Public	Synchronized	
Double	Protected	Native	

Keywords for exception handling

Try	finally	Throws
catch	throw	Assert

Keywords for conditional statements

If	Default	Break
Else	While	Continue
Switch	Do	Return
Case	For	

Value related keywords

True	False	Null

Other keywords

Return type keyword	Goto	enum
Void	Const	
Unused keyword	Extra keyword	

Keywords

class	package	super	null	int
interface	import	this	void	long
extends	new	true	byte	float
implements	instanceof	false	short	double

boolean	while	try	private	strictfp
char	do	catch	static	native
else	for	throw	final	volatile
switch	break	assert	abstract	goto
case	continue	public	synchronized	const
default	return	protected	transient	enum

Data/Literals

- The information is using in the programming language is known for being data/literal.

Literals

- The data or value used in Java is known as literals.

It is two types

1. primitive data
2. non-primitive data

1. Primitive data

The Single-value data such as numbers, characters, and Boolean is known as primitive data.

It is three types

a) Number
b) Character
c) Boolean

a) Number data

- It consists of both positive and negative numbers which can be whole number decimal numbers.

It is two types

i. Integer number data
ii. floating point data

i. Integer number data – It consists of positive or negative whole numbers.
E.x- 0, -3, 99, -55.
ii. Floating pointing number data – It consists of both positive and negative decimal numbers.
E.x. – 1.55, -1.111, 0.0, 2.33

b) Character data

- Anything enclosed with a single quote (‘…’) and length is ‘1’ is known as character data.
- Every character has an integer equivalent value which is known as the ASCII value.
- ASCII Stand for ‘‘American stander code for information interchange’’

ASCII TABLE

Character	ASCII Code	Character	ASCII Code	Character	ASCII Code
A	65	a	97	0	48
B	66	b	98	1	49
C	67	c	99	2	50
..	..	..	..	..	..
..	..	..	..	..	..
Z	90	z	122	9	57
[	91			@	64
\	92			(	40
]	93			)	41

Java support 65 thousand pulse character.

d) Boolean Literals

- Boolean literals are used to write logical values. We have 2 Boolean literals.
  1. True
  2. False

  - Both the literals are keywords.
  - True represents logical high 1.
  - False represents logical high 0.

Non-primitive data

- The multiple-value data are known as non-primitive data. Such as object reference string and null.

Object reference

- Java never disclose the original address of the object to access the object member instead of that it is giving one reference address which is known as object reference.

  It is represented by classname@hexadecimal value.

  Ex

ClassName	Reference
Student	student@123db765
Employee	employee@.....
Product	product@.....
O/a	o/a@.....

String Literals

- Anything enclosed within a double quote (" ") is known as string data/literals.
  The length of the string literals can be anything.
  They are case-sensitive.
  Ex – "hello", "true", "h", "8",

Null

- It is a keyword or reserve word.
- It is default value for non-primitive variable.

Variable

- It is a container that is used to store the value or data.

```
class A
{
	Public static void main (String[] agrs)
	{
		/* char section; //variable creation / declaration
		section ‘a’; // fill the container / initialization */
		char section = ‘a’;
		System.out.println(section); //access
	}
}
```

```
class A
{
	Public static void main (String [] agrs)
	{
		/* boolean c; //variable creation / declaration
		c = true;  // fill the container / initialization */
		Boolean c = true;
		System.out.println(c); //access
	}
}
```

```
class A
{
	Public static void main (String [] agrs)
	{
		/* int s; //variable creation / declaration
		s = 8;  // fill the container / initialization */
		int s = 8;
		System.out.println(s); //access
	}
}
```

Variables

- The variable is a container that is used to store the data or value or literal.

Based on which type of data is stored in the variable we have 2 types of variables as follows:

I. Primitive variable
II. Non-primitive variable

Steps to work with the variable

I. Create a variable or declare a variable or define a variable.
II. Fill the variable and assign the value to the variable or initialization.
III. Use the variable or access the variable.

Note - Apart from this we can refill the variable reinitialize, as well as reuse and re-access.

Primitive variable

- The variable which is used to store the primitive type value such as number, character, and Boolean is known as a primitive variable.

  We can create primitive type variable with the help of primitive datatype.

  Syntax to create/declare Primitive variable

  Primitive_datatype identifier1; or Primitive_Datatype identifier1, iderntifier2, identifier3;

  Example

  ```
  int number; or int a, b, c;
  boolean x;
  char ch;
  float price;
  ```

Syntax to Declare and Initialize Primitive variable

primitive_Datatype identifier1=value; //single line declaration & initialization.

Or

Primitive_Datatype identifier1, identifier2, identifier3; // declaration
Identifier1 = value1; //initialization
Identifier1 = value2; //initialization
Identifier1 = value3; //initialization
Example:

```
int a = 5;
Int p, q,r;
P=7;
q=9;
r=67;
```

Non primitive variable

- The variable which is used to store the reference is known as non-primitive variable.
  - It is also known as reference variable.
  - We can create non primitive type variable with the help of non-primitive datatype.

  Syntax to create or declare non primitive variable:

  NonPrimitve_Datatype identifier1;

  Or

  Non Primtive_Datatype identifier1, identifier2, identifier3,

  Example –

  String name; or String name, email;

  Student s;

  A a;

Datatypes

- Datatypes are used to create variables or specific types.

- In Java datatypes are classified into 2 types.
- These are as follows:
  I. Primitive datatypes
  II. Non primitive datatypes

I. Primitive datatypes

- The datatypes which are used to create a variable of primitive types to store primitive values such as the number character Boolean are known as primitive datatypes.
  - There are 8 primitive datatypes in java to create primitive types variables.
  - Those are
    I. byte
    II. short
    III. int
    IV. long
    V. float
    VI. double
    VII. char
    VIII. boolean

    Note all the primitive datatypes are keywords in java.

Primitive datatypes details

Primitive Data types	Default value	size	Range
byte	0	1 byte	-128 to +127
short	0	2 byte	-32768 to +32767
int	0	4 byte	-2147483648 to +2147483647
long	0 l/L	8 byte	
float	0 f/F	4 byte	
char	\u0000	2 byte	
double	0.0 d/D	8 byte	
boolean	false	1 bit	

Ranges of number datatypes

Byte: -128 to 0 to +127
Short: -32768 to 0 to +32767
Int: -2147483648 to 0 to +2147483647
Long: -9223372036854775808 to 0 to +9223372036854775807 etc.

Note – The number of datatypes in increasing order of the capacity is as follows:

Byte< short< int<long<float<double

Note – Float is greater than long not based on the capacity but based on memory mechanism.

II. Non-primitive datatypes

- The datatypes which are used to create the non-primitive variable to store the reference is known as Non-primitive Datatypes.

  Every class name in Java is a non-primitive datatype.

  Example: string, Demo, Student, etc.

  Note:

The default value for non-primitive datatype is null.
Null is a reserved word for value.

Ex –

```
class Student {
        public static void main (Staring [] args){
                Student a;
                a = new Student();

                System.out.println("Data reference: - "+a);
        }
}
```

Type casting

The process of converting one type of data into another type of data is known as Type casting.

There are two types of typecasting:

1. Primitive type casting
2. Non primitive type casting

1. Primitive type casting

i. Widening (Implicit primitive type casting)
ii. Narrowing (Explicit primitive type casting)

2. Non primitive type casting

i. Upcasting (Implicit non primitive type casting)
ii. Down casting (Explicit non primitive type casting)

1. Primitive type casting

The process of converting one primitive data into another primitive value is known as primitive type casting.

There are two types of primitive type casting:

i. Widening (Implicit primitive type casting)
ii. Narrowing (Explicit primitive type casting)

Widening

- The process of converting smaller range data into larger range data of primitive type is known as widening.
- In the widening process there is no data loss.
- Since, there is no data loss compiler can do it implicitly.

Note – it is also possible to do explicitly but not required.

Ex –

//WAJP creates a byte variable store the integer inside it and prints it.

```
class WideningExample1
    {
            public static void main (String [] args)
            {
```

```
                byte a=5;
                int d=a; //widening
                System.out.println("Byte_value – "+a);
                System.out.println("Int_value – "+d);
            }
        }
    /* Output – Byte_value - 5
                Int_value – 5    */
```

//**WAJP creates an integer variable store the double inside it and prints it.**

```
    class WideningExample2
        {
            public static void main (String [] args)
            {
                int a=5;
                double d=a; //widening
                System.out.println("Int_value – "+a);
                System.out.println("Double_value – "+d);
            }
        }
    /* Output – Int_value - 5
                 Double_value - 5.0 */
```

//**WAJP creates a character variable store the integer and double inside it and prints it.**

```
    class WideningExample3
        {
            public static void main (String [] args)
            {
                char ch='A';
                int i=ch; //widening
                double d=ch;
                System.out.println("Char_value – "+ch);
                System.out.println("Int_value – "+i);
                System.out.println("Double_value – "+d);
            }
        }
    /* Output – Char_value – A
                Int_value - 65
                Double_value – 65.0 */
```

Narrowing

- The process of converting larger range of primitive data into smaller range of primitive data is known as Narrowing.
- In narrowing process there is a possibility of data loss.
- Since there is a possibility of data loss, compiler does not do narrowing implicitly.
- It can be done explicitly by the programmer with the help of type cast operator.

Ex –

//**WAJP creates an integer variable stores the byte inside it and prints it.**

```
class NarrowingExample1
    {
        public static void main (String [] args)
        {
            int a=5;
            byte b=(byte)a; //widening
            System.out.println("Int_value – "+a);
            System.out.println("Byte_value – "+b);
        }
    }
/* Output – Int_value - 5
    Byte_value - 5 */
```

//WAJP creates a double variable store the integer inside it and prints it.

```
class NarrowingExample2
    {
        public static void main (String [] args)
        {
            double d=5;
            int i=(int)d; //widening
            System.out.println("Double_value – "+d);
            System.out.println("Integer_value – "+i);
        }
    }
/* Output – Double_value – 5.0
        Integer_value – 5    */
```

//WAJP creates an integer variable store the character and double inside it and prints it.

```
class NarrowingExample3
    {
        public static void main (String [] args)
        {
            int i=65;
            char ch=(char)i;
            double d=(double)ch;
            System.out.println("Char_value – "+ch);
            System.out.println("Int_value – "+i);
            System.out.println("Double_value – "+d);
        }
    }
/* Output – Char_value – A
        Int_value - 65
        Double_value – 65.0 */
```

Typecast operator

- It is a unary operator (only one operand)
- Type cast operator is used to explicitly convert one datatype into another data type.

Scope of the variable

- The visibility of a variable of a variable is known as the scope of a variable.

Based on the scope of a variable we can categorize variables into three types.

I. Local variable
II. Static variable
III. Non-static variable/ instance variable

I. Local Variable

- The variable declared inside a method block or any other block except the class block is known as a local variable.

Characteristics of local variable

- We can't use local variables without initialization, if we try to use a local variable without initialization then we will get a compile-time error.
- Local variables will not be initialization with default values.
- The scope of the local variable is nested inside the block wherever it is declared, hence it can't be used outside the block.

Ex – 1

```
class A
        {
                Public static void main(String[] args)
                {
                        int b; // Local variable
                        int c=7; //Local variable
                }
                public static void add()
                {
                        int x;// local variable
                        int y=90;//local variable
                }
        }
```

Ex – 2

```
        class B
                {
                static int p=17;//Static variable
                Public static void main (String [] args)
                {
                        int b; // Local variable
                        int c=7; //Local variable
                }
                System.out.println(p);

                }
```

Ex – 3

```
        class C
                {
                        static int p=17;//Static variable
                        Public static void main (String[] args)
```

```
        {
            int b; // Local variable
            int c=7; //Local variable

            System.out.println(p);
            System.out.println(c);
            System.out.println(b); //compile time error


        }

    }
```

II. Static variable –

- The variable which is declared inside the class block directory except any other block or outside of any other block with static keyword is known as static keyword.

Characteristics of static variable

- It is assigned with the default value.
- It has the global accessibility
- We can access it from anywhere by using suitable way/method.

Ex –

```
public class A {

    static int p;      // Static variable
    static double d;   // Static variable
    static Boolean b;  // Static variable (Note: "Boolean" should be lowercase)

    public static void main(String[] args) {
      System.out.println(p); // 0
      System.out.println(d); // 0.0 (Note: it's initialized to 0.0 by default)
      System.out.println(b); // null (Note: Boolean is an object type, so it's
initialized to null by default)
    }
}
```

III. Non-Static variable –

- The variable which is declare in the class directory outside of any other block without static keyword is known as non-static variable.

Characteristics of Non-static variable

- It is assigned with the default value.
- It has the global accessibility means we can access if form anywhere with suitable way/method.

Ex –

```
public class A {
  int p;      // non-static variable
  double d;   // non-static variable
  boolean b;  // non-static variable
  String s;   // non-static variable
  char c;     // non-static variable
```

```
    public static void main(String[] args) {
        A a = new A();  // Create an instance of class A

        a.p = 10;      // Assign values to the instance variables
        a.d = 3.14;
        a.b = true;
        a.s = "Hello";
        a.c = 'A';

        System.out.println(a.p); // 10
        System.out.println(a.d); // 3.14
        System.out.println(a.b); // true
        System.out.println(a.s); // Hello
        System.out.println(a.c); // A
    }
}
```

Ex – Local, Static, Non-static variable

```
public class LocalVar {
    char a;              // Instance variable (non-static)
    static int v;        // Static variable (class variable)
    static String name;  // Static variable (class variable)
    String address;      // Instance variable (non-static)

    public static void main(String[] args) {
        int p = 0;       // Primitive local variable, initialized
        String t = null; // Non-primitive local variable, initialized to null
        int y;           // Primitive local variable (uninitialized)

        System.out.print("hy");  // Straight double quotes
        System.out.print(v);
        System.out.print("hy");
    }
}
```

Operator

- Operator are predefined symbol which is used to perform some specific tasks on the given data.

Operand

- The data given as an input to the operator is known as operand.

Based on the number of operands given to the operator. Operators are classified into 3 types.

I. Unary Operator
II. Binary Operator
III. Ternary Operator

I. Unary Operator

- The operator which can accept only one operand is known as unary operator.

  Ex – 'Type cast', 'New operator', logical not, etc.

II. Binary Operator

- The operator which can accept two operands is known as Binary Operator.

  Ex – Arithmetic operator, Relational, assignment operator etc.

III. Ternary Operator

- The operator which can accept 3 operands is known as Ternary Operator.

  Ex – conditional operator

Classification of operator based on task

Based on the task or operation of the operator, it is classified into as follows:

I. Arithmetic operator
II. Assignment operator
III. Relational operator
IV. Logical operator
V. Conditional operator
VI. Increment/decrement operator

etc

Logical operator

1. It is used to combine the condition.
2. It also returns Boolean value.

a. Logical And (&&) –

- If all the expression returns true, then this operator will return true.

Op1	Operator	Op2	O/P
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

Ex –

```
System.out.println(true && true); // true
System.out.println(true && true && true && true && true && false); // false
System.out.println(true && true && false && false && false && false); // false
System.out.println(false && false && false && false); // false
```

b. Logical OR (| |) –

- If any one of the expressions return true, then this operator will return true.

Op1	Operator	Op2	O/P
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false

Ex –

```
System.out.println(true||true); // true
System.out.println(true||true||true||true||true||false); // true
System.out.println(true||true||false||false||false||false); // true
System.out.println(false||false||false||false); // false
```

c. Logical NOT(!) –

- If the expression is true, then it will return false and vice versa.

Operator	Op1	O/P
!	false	true
!	true	false

Ex –

```
System.out.println(! true) ;//false
System.out.println(! false) ;//true
```

Relational operator

1. It returns Booleans value.
2. It used to make condition.

Operator	Operation
==	is equal to equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Arithmetic operator

Operator	Operation
+	Addition
-	Subtract
*****	Multiplication
/	Divide

'+' operator is used to perform addition along with the string concatenation operation.

'/' operator is used to give the quotient of a division operation.

'%' operator is used to give the remainder of a division operation.

Assignment operator

It generally 2 types. Simple Assignments operator and compound assignment operator.

	Operator	**Example**
Simple compound	=	a=5
	+=	a+=5
Equivalent to	-=	a-=5
	*=	a*=5
	/=	a/=5
	%=	a%=5

Ex – 1

```
Class A
{
```

```
        public static void main (String[]args)
        {
                int amt=500;
                System.out.println(amt);
                //amt=amt+6000;
                amt+=6000;
                System.out.println(amt);
        }
}
```

Ex – 2

```
Class B
{
        public static void main (String[]args)
        {
                int amt=1500;
                System.out.println(amt);
                //amt=amt-600;
                amt-=600;
                System.out.println(amt);
        }
}
```

Ex – 3

```
Class C
{
        public static void main (String[]args)
        {
                int amt=1500;
                System.out.println(amt);
                amt=amt/600
                amt/=600;
                System.out.println(amt);
        }
}
```

Conditional operator

Conditional operator:

- It is a ternary operator.

Syntax to create conditional operator:

Operand1? operand2: operand3
Condition? statement1: Statement2

Operation

- The return type of operand 1 must be Boolean.
- If the condition is true statement1 will get executed else statement 2 will get executed.

The return type of the conditional operator depends on the operand2 and operand3.

Increment /Decrement operator

++(increment)	++i (pre increment)	1. Increase the value by 1 of I and update 2. Substitute the updated value 3. Use the updated value
	i++ (past increment)	1. Substitute 2. Increase the value by 1 of I and update 3. Use the substituted value
--(decrement)	--i (pre decrement)	1. Decrease the value by 1 of I and update 2. Substitute the updated value 3. Use the updated value
	i-- (post decrement)	1. Substitute 2. Decrease the value by 1 or I and update **3.** Use the substituted value

Control Statement

- It is the use to control the flow of the execution of the program

It is of two types

i. Decision making statement / Branching statement/ Conditional statement
ii. Looping statement

i. Decision making statement

- The decision statement helps the programmer to skip the block of instructions from the execution if the condition is not satisfied.

Type of decision statements

1. If statement
2. If else statement
3. If else if ladder
4. Switch

ii. What is looping statement

- Loop statement helps the programmer to execute the set of instructions repeatedly.

In java we have different type of loop statements, they are:

1. While loop
2. Do while loop
3. For loop
4. For each/advance for / enhanced for loop

Syntax to create if statement:

```
If(condition)
{

}
```

Workflow

- if the condition is satisfied then the instruction written inside the block gets executed or normal flow of the execution continues (instructions written inside the if block is skipped if condition is not satisfied).

If else if statement:

```
if(condition)
{
        //statement
}
else if(condition)
{
        //statement
}
else if(condition)
{
        //statement
}
else
{
        //statement
}
```

Workflow

- If the condition is satisfied then the instruction written inside the if block gets executed if not satisfied, the condition is checked in the else if block from top to bottom order and if the condition is satisfied in any of the else if block then, only that else if block is gets executed if not satisfied else block gets executed remaining blocks are skipped.

Switch

```
Switch(value/variable/expression)
{
        Case (value/expression)
        {
                //statement;
        }[break;]
        Case (value/expression)
        {
                //statement;
        }[break;]
        Default:
        {
                //statement;
        }
```

Workflow:

- The value/variable/expression passed in the switch gets compared with the value passed in the case from top to bottom order.
- If any of a case is satisfied, the case block is executed and all the blocks present below get executed.
- It no case is satisfied then the default block gets executed.
- For a case we can use a break statement which is optional.

  Note –
  - For a switch we can't pass long, float, double, Boolean.

- For a case we can't pass a variable.

Break

- Break is a keyword it is a control transfer statement.
- Break is used inside the switch and loop block.
- When the break statement is executed, control is transferred outside the block.

While Loop

Syntax to create while loop:

```
while(condition)
        {
                //Statement to be repeated;
        }
```

Workflow:

Case 1: When the condition is true.

The loop continues.
Control executes the statement which belongs to the loop.
After execution once the loop block ends, control goes back to the
Condition and the entire process will be repeated till the condition becomes false.

Case 2: when the condition is false

The loop is stopped i.e., repetition is stopped.
The loop block will not get executed.
The control comes outside the loop to the next statement.

Do while loop

Syntax to create do-while loop:

```
Do
        {
                //statement
        }
        While(condition);
```

Workflow of do while

Workflow

Case 1: When the condition is true

Control goes to the loop block directly, execute the instructions.
Then control goes to the condition, if the condition is true the control goes back to the do block.

Case 2: when the condition is false

Control goes to the loop block directly, execute the instructions.
Then control goes to the condition, if the condition is false the loop is stops and control goes to the next statement.

Difference between while and do while loop:

While	**do-while**
First, the condition is checked, if the condition is true then the loop block gets executed. The minimum iteration can be zero.	In do while, first the loop block gets executed and then the condition is checked. The minimum iteration is one.
Example	Example

Int a=5, b=10, count=0;
While(a>b)
{
Count++;
s.o.pln("value of b is "+b);
}
s.o.pln("lieration "+count);
 	
output:iteration 0
	Int a=5, b=10, count=0;
	Do
	{
	Count++;
	s.o.pln("value of b is "+b);
	}
	While(a>b);
	s.o.pln("iteration"+count);
	
	output:value of b is 10
	iteration 1

For loop

Syntax to create for loop:

```
For (initialization; condition; update)
        {
        // statement to be repeated
        }
```

Workflow:

Step1: control go to the initialization part.
Step2: then it will go to the condition part.
Step3: if the condition is true then it will enter inside the loop block
Step4: once the execution of instruction indside the loop block is completed control will bo to the update segment.
Step 5: then it will go back to the condition. Step2,3,4 will continue until the condition become false.

Note:

All the three segments are optional (initialization, condition, update).
If the condition is not provided by default, it is considered as true.

Looping

- If in the switch statement, we are having similar type of block of instruction for individual cases case in state of writing all the instruction individual we can prove all the similar case and we can write the block of instruction at last. Which is known as looping.

Nested decision-making statement

- If inside one decision making statements another decision-making statement will be there, it is known as nested decision-making statement. It can be nested if or if else ladder and nested swich, etc

Method

- It is a block of instruction which is going to perform a specific task when we will call it.
- We are creating a method inside the class.
- We can create n number of method inside class.
- We cannot create method inside another method but we can call a method from another method n number of times.

Syntax to create the method is as follows

```
[AS] [M] RT MN ([FA])
{
        //implementation / body/block
}
```

Here AM, M & FA is optional to create a method

Am – access method
M – modifier
Rt – return type
Mn – method name
Fa – formal argument

Terminology

Method definition

- It is a combination of method declaring and method implementation.
  Ex – [access modifier] [modifier] returntype methodname ([formal argument]) {

```
        //implementation / body/block/instructions
    }
```

Method declaration

- It is the combination of access modifier,method, return type and method signature.

Method signature

- It is combination of method name and formal argument.
  Ex – methodname ([formal argument])

Method implementation

- This is also known as method block or method body inside this block we are writing the instruction which is going to execute when the method is being called.

We are calling the method from the method called statement.

Method name ([actual argument]);

Note –

Not of formal arguments = Actual argument
Type of formal arguments = Actual argument
Order of formal arguments = Actual argument

Ex

```
method
public static void main ()
    {
    System.out.println();
    }
```

Types of method

- Based on the number of arguments, methods are classified into 2 types.
  i. No argument method
  ii. Parameterized method

No argument method

- A method which does not have formal argument is known as no-argument method.
  Ex –

```
public static void demo ()
{
        System.out.println("demo () no argument method");
```

```
}
```

Parameterize method

- The method which has formal argument is known as parameterized method.
- Parameterized methods are used to accept the data.

Ex: -

```
public static void demo (int a)
{
        System.out.println("demo (int a) parameterised method")
}
```

Method calls statement

- The statement which is used to call a method is known as a method as a method call statement.

Syntax to create a method call statement:

```
methodname ();
```

We can call a no-argument method without passing an actual argument in the method call statement.

How to call the parameterized method.

Syntax to create a parameterize method

```
methodName (Actual argument1, actual argument 2, .....)
```

ex – method:

```
p. s. v. main (String [] args) {
        // statements;
}
```

Rule to call the parameterized method

- The number of actual arguments should be same as the number of formal arguments.
- The type of corresponding actual argument should be same as the same type of formal argument, if not compiler tries to convert implicit conversion if it is not possible then we will get compile time error.
- The sequence of actual argument should be same as the sequence of formal argument.

When your make method is parameterized

- When the execution of block of instruction of the method depends on calling method or caller method value you have make our parametrize.

Return type

- It is use to specify that when type of data is return by the method after execution of the method is completed, to the caller
- It is done by the help of return statement.

Return statement

- It is used to return the value to return the value to the caller.

Syntax of return statement

```
return [data/variable/expression]
```

Return keyword -

- Return is a keyword.
- It is a control transfer statement.
- When the return statement is executed, the execution of the method is terminated and control is transfer to the calling method.

Rule for return statement

- The type specified as return type should be same as the type of value passed in a return statement. Return statement should be last statement and it can many but only one should be executable.

Ex –

```
public string m1()
{
    int mno=1;
    if(mno==1) {
        return "jan";
    }
    else if (mno==2) {
        return "feb";
    }
    else {
        return "invalid";
    }
}
```

Types of return type: -

- A method can have the following return type

1. void
2. primitive return type
3. non primitive return type

void return type with return statement -

void

- it is a keyword. when method is not pomaching to return any data to the caller. We have declared a method return type as void.
- We can have the return statement inside the block when method block when method having a return type as void but it is not compulsory.
- The time of having the return statement inside the method block we should not provide any to return statement. Nightery 0 or 0.0, null because 0 or 0.0 are primitive data whereas null is not primitive data.

Ex - 1

```
public void m1()
{
    System.out.println("hy");
}
```

Ex – 2a

```
public void m1()
{
    return // we can take return statement
}
```

Ex - 2b

```
public void m1()
{
        // return statement is not compulsory
}
```

Ex – 3

Case -1

```
public void m1()
{
        return;//cts
}
```

Case – 2

```
public void m1()
{
        return 0;//cte
}
```

Case – 3

```
public void m1()
{
        return 0.0;//cte
}
```

Case – 4

```
public void m1()
{
        return null; //cte
}
```

Ex – 4

```
p. s. v. m1() {
        return null; //cte (we can't pass null also)
}
```

Primitive return type with return statement -

- When method is primitive data, we are using primitive return type bye help primitive return data type. Such as byte, short, int, long, double, float, char, Boolean.
- When method having primitive return type it is compulsory to have return statement inside the method and return statement should return corresponding primitive data.
- If we are not passing corresponding data to the return statement with respect to return type, compiler trying to convert it implicitly. if it possible if not we will get compile time error.

Ex 1
Case – 1

```
public int m1()
{
        return 5;
```

```
}
```

Case – 2

```
public int m1()
{
    //cte
}
```

Case – 3

```
public int m1()
{
    return;//cte
}
```

Ex – 2
Case – 1

```
public int m1()
{
    return 50;
}
```

Case – 2

```
public doutble m1()
{
    return 5.3;
}
```

Case – 3

```
public char m1()
{
    return ‘a’;
}
```

Case – 4

```
public boolean m1()
{
    return true;
}
```

Ex – 2
Case – 1

```
public int m1()
{
    return 50;
}
```

Case – 2

```
public doutble m1()
{
        return 5.3;
}
```

Case – 3

```
public char m1()
{
        return 'a';
}
```

Case – 4

```
public boolean m1()
{
        return true;
}
```

Ex – 3
Case – 1

```
public double m1()
{
        int a=5;
        int b=10;
        return a*b; //implicitly
}
```

Case – 2

```
public doutble m1()
{
        return 'a';
}
```

Case – 3

```
public int m1()
{
        return 'a';
}
```

Ex – 4
Case – 1

```
public double m1()
{
        return 5.3;//cte
}
```

Case – 2

```
public char m1()
{
```

```
        return 3.5; //cte
    }
```

Case – 3

```
    public boolean m1()
    {
        return 5; //cte
    }
```

Non primitive return type with return statement -

- If the method promises to return non-primitive data, we have to use non primitive return type by the help of non-primitive return type by the help of non-primitive data type such as String or any of class name.
- When method is having non-primitive return type its half return statement inside the method block.
- Return statemen return corresponding non primitive data.
- The data is not matching with return type, then compiler try convert implicitly or else will get compile time error.

Ex – 1
Case – 1

```
    public string m1()
    {
        return "hy';
    }
```

Case – 2

```
    public string m1()
    {
        return null;
    }
```

Case – 3

```
    public string m1()
    {
        //cte
    }
```

Case – 4

```
    public string m1()
    {
        return;//cte
    }
```

Case – 5

```
    public a m1()
    {
        A a=new A();
```

```
        return a; //a type reference.
    }
```

Case -6

```
    p. s. Student m2 () {
        Student s=new Student ();
        return s;
    }
```

Argument

Formal Argument:

- A variable which is declared in a method Declaration is known as Formal argument.

Actual Argument:

- The values passed in the method call statement is known as Actual Argument.

Calling method:

- The method which is trying to call another method is known as the calling method (caller).

Called method:

- The method which is being called by the caller is known as a called method.

Ex –

```
class a
{
    public static void main () {
        System.out.println("start");
        m1();
        System.out.println("end");
    }
    public static void m1(){
        System.out.println("hy from m1");
        m2();
    }
    public static void m2(){
        System.out.println("hy from m2");
    }
}
```

Method calls statement flow:

- Execution of calling method is paused
- Control is transferred to the called method.
- Execution of called method begins.
- One the execution of the called method is completed the control is transferred back to the calling method.
- Execution of calling method resumes.

Static

- It is a modifier or keyword.
- If the member is static, we can access it directly without creating the object

Ex – class Demo {

```
    //when the components are non-static
```

```
int a=8; // non static
public static void mani (String [] args)
        m1(); // we can not access directly
        System.out.println(a); // we cannot access directly
    }
    public void m1() {
        System.out.println("Hy") ;// we cannot access directly
    }
}
```

Ex – 2

```
public class Q7 {
    //when the components are static
    static int a=8; //static
    public static void main (String [] args) {
        m1();
        System.out.println(a);
    }
    //static
    public static void m1() {
        System.out.println("Hy");
    }
}
```

Final

- It is a keyword as well as modifier.
- If the variable is final, we cannot reinitialize the variable but we can re accessing

Ex – not final

```
class x
    {
        Public static void main (String [] args);
        {
        // variable not final:
        Int a=10;
        System.out.println(a);
        A=80;
        System.out.println(a);
        }
    }
```

Ex – final

```
class x
    {
        Public static void main (String [] args);
        {
        // variable final:
        Final Int a=10;
        System.out.println(a);
        A=80;
        System.out.println(a);
        // compile time error.
```

```
        }
    }
```

Modifiers

- Modifier is used to change the behaviour of java components
  Static, Final, Abstract, Synchronized, Transient, Volatile, Public, Private, protected

Access modifiers

- Access modifiers are responsible to change/modify the accessibility of the member.
- We have 4 types of access modifiers as follows:
  1. Private
  2. Default
  3. Protected
  4. Public

1. **Private**
   - If is a class level modifier, it is responsible for variables, methods and constructors.
   - If the member of a class is prefixed with a private modifier, then it is accessible only within the class accessing outside the class is not possible.

2. **Default**
   - The accessibility of default modifiers is only within the package. It can't be accessed from outside the package.
   - If you don't declare any access modifiers then it is considered as a default access modifier.

3. **Protected**
   - The access level of a protected modifier is within the package and outside the package through child class.
   - If you don't make the child class, it cannot be accessed from outside the package.

4. **Public**
   - The access level of the public modifier is everywhere.
   - It can be accessed from within the class, outside the class, within the package as well as outside the package.

Scope of an access modifier:

Access modifiers	Within the class	Within the package	Outside the package	Outside the package by child class
private	Yes	No	No	no
default	Yes	Yes	No	No
protected	Yes	Yes	No	Yes
public	Yes	Yes	Yes	yes

Scope of access modifiers

The scope of the access modifiers based on the accessibility is:

Private<default<protected<public

Main Method –

- It is a method from where java program execution going to start.
- The execution of a java program always starts from the main method defined as follows.

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

Purpose of the Main method

- Start the execution
- Control the flow of the execution
- End of execution

Note –

- A method can be executed only when it is called, we can a method any number of times, therefore it is said to be code reusability.
- The main method is always called by JVM.

Why main method is public, static, void

public

- main method is public because jvm can call the main method from outside the class.

static

- It is static because jvm can call the main method without creating the object.

Void

- It is void because Main method is not returning any value to the jvm once after execution done.

main

- Main is the name or identifier which is given to the jvm to identify the main method.

string [] args

- It has string [] as an argument because all the data can be converted into string type and vice verca.

Can be passed data to the main method it as formal argument
Yes we can

By the help command line argument
By calling the main method explicitly.

What is command line argument.

- The argument or value to the passing command promt to the java program command line argume. Ex – java A 1 2 hy 5.3 //these all the elements are going to argument to store the args variable of main method.

What is method

- If one class having more than one method with same method name but changing in formal argument over loading.
- Change formal argument means changing number of formal arguments
- Change in type of argument
- Change in sequence of formal argument

Note – the time of method over loading on return time and name of the formal argument.

Ex –

```
Class {
```

```
        Public static void main (string [] args) {
                System.out.println ("Hy");
                Add ();
                Add (5);
                Add (5,7);
        }
        Public static void add () {
                System.out.println("Hy1");
        }
        Public static void add (int a) {
                System.out.println(a+9);
        }
        Public static void add (int a, int b) {
                System.out.println(a+b);
        }
}
```

Can be over load the main method

- Yes, we can over load the main method changing the formal argument.

Ex –

```
        Class K {
                Public static void main (string[] args){
                        System.out.println("Hy");
                        Main ();
                        main (5);
                        main (5,7);
                }
                Public static void main () {
                        System.out.println ("Hy1");
                }
                Public static void main (int a) {
                        System.out.println (a+9);
                }
                Public static void main (int a, int b) {
                        System.out.println (a + b);
                }
        }
```

Ex Method overloading

```
class B {
        //method overloading which is differ in datatype of FA
        public static void m1() {
                System.out.println("hy");
        }
        Public static void m1(int a) {
                System.out.println("hello");
        }
        public static void main (String [] args) {
                System.out.println("hiii");
                m1 ();
                m1 (6);
        }
```

```
    }
```

Ex 2 –

```
    class C {
        public static void m1(String a, int b) {
            System.out.println("hy");
        }
        Public static void m1(int a, String b) {
            System.out.println("hello");
        }
        public static void main (String [] args) {
            System.out.println("hiii");
            m1 (6, "hy");
            m1 ("56",6);
        }
    }
```

Part – II

What is Dynamic reading?

- The process of reading data from the user through keyboard at the execution time of the program is known as dynamic read.

Steps to achieve dynamic read

1. Import scanner class
   For example, import java.util.scanner;
2. Create object for the scanner class
   Scanner sc=new Scanner (System.in);
3. By using reference variable sc call the required method of scanner class to read the values/data from the user.

```
Byte – nextByte ();
Short – nextShort ();
Int – nextInt ();
Long – nextLong ();
Float – nextFloat ();
Double – nextDouble ();
Char – next.charAt (0);
Boolean -nextBoolean ();
String – next (); //single word
String – nextLine (); // multi word
```

Static and Static members

What is static

- Static is keyword and modifier
- Any member of a class is prefixed with a static modifier then it is known as a static member of a class.
- Static members are also known as class members.
  Note –
  Static members can be prefixed only for a class member (members declared in a class).

Ex - Static variable and modifiers

```
public class MyClass {
   // Static variable (class-level variable)
   static int staticVariable = 0;


   // Non-static variable
   int instanceVariable = 0;


   public static void main(String[] args) {
      // Accessing static variable
      staticVariable = 10;


      // Creating objects of the class
      MyClass obj1 = new MyClass();
      MyClass obj2 = new MyClass();
```

```
        // Accessing instance variable through objects
        obj1.instanceVariable = 5;
        obj2.instanceVariable = 8;


        // Accessing static variable through class or object
        System.out.println("Static Variable: " + MyClass.staticVariable);
        System.out.println("Instance Variable (obj1): " + obj1.instanceVariable);
        System.out.println("Instance Variable (obj2): " + obj2.instanceVariable);
    }

}
```

Ex – Static method and modifiers:

```
public class MathUtils {

    // Static method to add two numbers
    public static int add(int a, int b) {
        return a + b;
    }

    // Non-static method (instance method) to subtract two numbers
    public int subtract(int a, int b) {
        return a - b;
    }

    public static void main(String[] args) {
        int result1 = MathUtils.add(5, 3); // Calling the static method directly
        System.out.println("Result of addition: " + result1);


        MathUtils mathObj = new MathUtils();
        int result2 = mathObj.subtract(10, 4); // Calling the instance method through an object
        System.out.println("Result of subtraction: " + result2);
    }

}
```

Static members

- Static method
- Static variable
- Static initializers

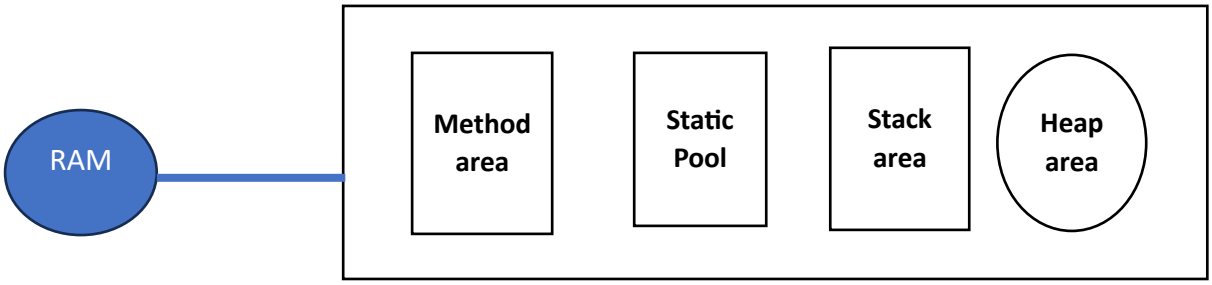
Ex –

```
public class StaticMembersExample {
    // Static variable
    static int staticVar;

    // Static initializer (executed when the class is loaded)
    static {
        System.out.println("Static initializer called.");
        staticVar = 42;
    }

    // Static method
    public static void staticMethod() {
        System.out.println("Static method called.");
    }
```

```
    public static void main (String [] args) {
        // Accessing the static variable
        System.out.println("Static variable: " + staticVar);

        // Calling the static method
        staticMethod();
    }
}
```

Java runtime memory

- To execute the java program a portion of memory in RAM is allocated for JRE.
- In that portion of memory allocated, we have different range of memory, hence they are classified as follows.
  1. Method area
  2. Static pool area
  3. Static area
  4. Heap area



Method Area:

- All the methods and multi-line initializers blocks will be loaded in a method area with reference (instruction of the methods).

Static pool area – it is having many more class static area for each class.

Class static area

- For every class there is a dedicated block of memory is created in the static pool area (Static pool) which is known as class static area.
- The static members of the class will be allocated inside the memory created for the class.

Stack area

- The stack area is used for the execution of instructions.
- For every method that is under execution a block of memory is created in this static area which is known as a frame.
- Once the execution of a method is completed the frame is removed.

Heap area

- In a heap area, a block of memory is created for the instance of a class (object).
- Every block of memory is created with the help of reference.
- All the non-static members or instance member will be stored inside the object block which is present inside the heap area.

Static variables

- Variable declared in a class block and prefixed with static modifier is known as static variable.

Characteristics

- It is a member of the class.
- It will be assigned a default value.
- Memory will be allocated inside the class static area.
- It is global in nature; it can be used within the class as well as in different classes.
- We can use a static variable with the help of the class name as well as with the help of object reference.
- We can access the static variable from different classes directly with the help of the class name.

Static method

- A method prefixed with a static modifier is known as the static method.

Characteristics:

- Static method block is loaded in the method area and reference of the static method is stored inside the class static area (static pool).
- We can use the static method with or without creating an object of the class.
- We can use the static method with the help of the class name.
- A Static method of the class can be used in any class with the help of a class name.

  Note –

  If static variables and local variable are in the same name then we can differentiate static variable with the help of class name.

Class Loading Process

The process of loading the class members into the java time environment is known as class loading process. This is done by class loader. The steps class loading process as follows:

1. One block of memory will be created on the Static Pool Area (SPA) with the name of the respected class which one u gave to the jvm to execute the program, this block is known as Class Static Area (CSA).
2. All the method (static and non-static) and multiline initializers (static and non-static) will be loaded into the method are with a reference.
3. All the static members will be store in the CSA. Static members mean (static variable, static method, static multi line initializer).
   a. If is there any static variable present in the class then it will store inside the CSA with the default value.
   b. If is there any static method then it will be store in the CSA in a form on Table which consists of 2 columns (1 column represents method signature and another represents reference of the method)
   c. If is there any static multi line initializer then it will be store in the CSA in the form of table which is previously created.
4. If is there any static initializer present in the class then all will be executed one by one from top to bottom manner.
5. Now we can say that the class loading process done. As the class loading process done, then now JVM will call the main method to start the execution of the program.

   Note –

   JVM will call only the main method of the initial loaded class.

STATIC INITIALIZERS:

We have two types of static initializers. Those are,

1. Single line static initializer
2. Multi line static initializer

1. Single line static initializer:

Syntax to create single line static initializers:

Static data type variable = value/expression;
Ex: static int a=10;

2. **Multi line static initializer:**
   Syntax to create multi line static initializers:

```
Static {
        // statements;
        }
```

Example:

```
Static {
        System.out.println("welcome to ATM");
        }
```

Characteristics:

- Static initializers get executed implicitly during the loading process of the class.
- A class can have more than one static initializer they execute top to bottom order.

Purpose of static initializer:

- Static initializers are used to executed the startup instructions during the class loading process.
- As the static blocks get executed before the actual execution of the main method.

Static context:

- The block which belongs to the static method and multi-line static initializer is known as static context.
- Inside a static context, we can use the static method members of the same class directly by using its name.
- Inside a static context, we can't use the non-static members of the same or different class directly by using its name or by using its class name.
- This keyword is not allowed inside the static context.

What is Object?

Object:

- Any substance which has existed in the real world is known as an object.
- Every object will have attributes and behaviours.

Object in Java:

- According to object-oriented programming, the object is a block of memory created in the heap area during the runtime, it represents a real-world object.
- A real-world object consists of attributes and behaviour.
- Attributes are represented with the help of non-static variables.
- Behaviours are represented with the help of non-static methods.

What is class?

Class:

- According to real-world situations before constructing an object blueprint of the object must be designed, it provides the specification of the real-world object.
- Similarly in object-oriented programming before creating an object the blueprint of the object must be designed which provides the specification the object, this is done with the help of class.

DEFINITION OF CLASS:

- It is a user-defined non-primitive data type. it represents the blueprint of the real-world object.
- The class provides the specification of real-world objects.

NOTE:

- We can create N number of objects for a class. Multiple Objects created using same class is called as Similar Object or Identical Object.
- Object is also known as an instance of a class.

Steps to create an object:

STEP 1: Create a class or use an existing class if already created.
STEP 2: Instantiation

INSTANTIATION:

The process of creating an object is known as instantiation.
Syntax to create an object:
new ClassName ([actual argument]);
Here, ClassName ([Actual Argument]) is constructor

new:

- new is a keyword.
- It is a unary operator.
- It is used to create a block of memory inside a heap area during runtime.
- Once the object is created it returns the reference of an object.

EXAMPLE:

Step 1: Designing a class

```
class Employee {
        String ename;
        int eld:
}
```

Step 2: Instantiation

```
new Employee ();
```

NON-PRIMITIVE DATA TYPE:

- Every class name in java is a non-primitive data type.
- Non primitive data types are used to create a non-primitive variable to store the reference of an object.

EXAMPLE:

```
class Employee {
        String ename;
        int eid;
}
Class Driver {
        public static void main (String[] args){
        Employee e = new Employee ();
        Sopin(e); // Employee@4dvec1
        }
}
```

NON-STATIC:

- Any member declared in a class and not prefixed with a static modifier is known as a non-static member of a class.
- Non-static members belong to an instance of a class. Hence it is also known as an instance member or object member.
- The memory for the non-static variable is allocated inside the heap area (instance of a class).
- We can create any number of instances for a class.
- Non-static members will be allocated in every instance of class.

Non static members

- Non-static variable
- Non-static method
- Non-static initializers
- Constructor

CONSTRUCTOR:

- Constructor is a special type of non-static method whose name is same as the class name but it does not have a return type.

Syntax to create the constructor:

A programmer can define a constructor by using the following syntax:

```
[access_modifier] className ([Formal_Arguments]) {
// initialization;
}
```

PURPOSE OF THE CONSTRUCTOR:

- During the execution of the constructor,
- Non-static members of the class will be loaded into the object.
- If there is a non-static initializer in the class, they start executing from top to bottom order.
- Programmer written instruction of the constructor gets executed.

NOTE:

If the programmer fails to create a constructor, then the compiler will add a default constructor**.**

CLASSIFICATION OF CONSTRUCTOR:

Constructors can be classified into two types based on the formal argument,

1. No argument constructor
2. Parameterized constructor

NO ARGUMENT CONSTRUCTOR:

A constructor which doesn't have a formal argument is known as a no-argument constructor.

Syntax to create no argument constructor:

```
[access modifier] className () {
code;
}
```

NOTE:

If the programmer fails to create a constructor, then the compiler Implicitly adds a no-argument constructor only which is known as default constructor.

PARAMETERIZED CONSTRUCTOR:

The constructor which has some formal argument is known as parameterized constructor.

Syntax to create Parameterised constructor

```
[access modifier] className ([Formal Argument]) {
    // statement;
}
```

PURPOSE OF THE PARAMETERIZED CONSTRUCTOR:

Parameterized constructors are used to initialize the non-static variables by accepting the data from the constructor in the object creation statement.

CONSTRUCTOR OVERLOADING:

If a class is having more than one constructor but differ in signature, is known as constructor overloading

- Differ in signature means
- Differ in number of arguments
- Differ in Datatype of arguments
- Differ in sequence of arguments

RULE

The signature of the constructor must be different.

CONSTRUCTOR CHAINING:

- A constructor calling another constructor is known as constructor chaining.
- In java, we can achieve constructor chaining by using two ways
  1. this () (this call statement)
  2. super () (super call statement)

This ():

It is used to call the constructor of the same class from another constructor.

RULE FOR THIS CALL STATEMENT:

- this () can be used only inside the constructor.
- It should always be the first statement in the constructor.
- The recursive call to the constructor is not allowed (Calling by itself).
- If a class has n constructors we can use this statement in n-1 constructors only(at least a constructor should be without this ().

  NOTE

  the constructor has this () statement, then that constructor doesn't load nonstatic members du execute the nor static initializer & which constructor doesn't have that one only doing these task

NON-STATIC METHOD:

- A method declared in a class block and not prefixed with a static modifier is known as a non-static method.

CHARACTERISTICS:

- A method block will be loaded inside the method area with the reference and a reference of the method will be stored inside the instance of a class [object].
- We can't call the non-static method of a class without creating an instance of a class[object] inside the static context directly but we can use inside non-static context.
- We can't access the non-static method with the help of class names.
- The non-static method can be accessed directly inside the non-static context by their name but can't be accessed directly inside the static context by their names. It is also applicable to the variable.

NON-STATIC VARIABLE:

- A variable declared inside a class block and outside any of method or multiline initializer block and not prefixed with a static modifier is known as a non-static variable.

CHARACTERISTICS:

- We can't use the non-static variable without creating an object inside the static context directly but we can use inside non-static context.
- We can only use the non-static variable with the help of object reference.
- Non-static variables are assigned with default values during the object loading process.
- Multiple copies of non-static variables will be created (once for every object).

NON-STATIC CONTEXT:

- The block which belongs to the non-static method and multi-line non-static initializer is known as non-static context.
- Inside a non-static context, we can use static and non-static members of the same class directly by using their name.

NON-STATIC INITIALIZERS:

- Non-static initializers will execute during the loading process of an object.
- Non-static initializers will execute once for every instance of a class created. [object created].

PURPOSE OF NON-STATIC INITIALIZERS:

Non-static initializers are used to execute the startup instructions for an object.

TYPES OF NON-STATIC INITIALIZERS:

1. Single line non-static initializer
2. multi-line non-static initializer

1. SINGLE LINE NON-STATIC INITIALIZER:

Syntax to create single line non static initializers:
datatype variable = value / reference;

2. MULTI LINE NON-STATIC INITIALIZER:

Syntax to create multi line non static initializers:

```
{
// statements;
}
```

NOTE:

All the Non-static initializers will execute from top to bottom order for every object creation,

this:

- It is a keyword.
- It is a non-static variable it holds the reference of a current executing object.

USES OF THIS:

- Used to access the members of the current object.
- It is used to give the reference of the current object.
- Differentiate local Variable with non-static variable if both have same name.
- Calling a constructor of the same class is achieved with the help of this call statement.

New keyword work

Basically, by new keyword 3 works are done as follows:

1. It will create a block of memory inside heap area and provide the reference for that block.
2. It will the call the constructor to load the non-static members of the class.
   Work of the constructor for the constructor one frame will be created on the SA.
   I. All the non-static members (non-static, variable, method & multiline initializer) will be loaded into the object.
      a. If is there any non-static variable present in the class then it will store inside the object (which is created on the HA) with the default value.
      b. If is there any non-static method then it will be store in the object in a form or table which consists of 2 columns (1 column represents method signature and another represents reference of the method).
      c. If is there any non-static multi line initializer then it will be store in the object in a form of table which is previously created.

II. If is there any non-static initializer present in the class then all will be store in the object in a form of table which is previously created.
III. If is there any non-static initializer present in the class then all will be execute one by one from top to bottom manner.
[Note – for multi-line non-static initializer frame want be created on the stack area.]
IV. The programmer written instructions inside the constructor will be execute.

3. **After constructor work done then new keyword will return the reference of the created object.**

PRINCIPLE OF OOPS:

- Object-oriented programming has the following principles:
  1. Encapsulation
  2. Inheritance
  3. Polymorphism
  4. Abstraction

ENCAPSULATION:

- The process of binding the state(attributes/fields) and behaviour of an object together is known as encapsulation.
- We can achieve encapsulation in java with the help of the class, class has both state and behaviour of an object.

ADVANTAGE OF ENCAPSULATION:

- By using encapsulation, we can achieve data hiding.

DATA HIDING:

- It is a process of restricting the direct access of data members of an object and provides indirect secured access of data members via methods of the same object is known as data hiding.
- Data hiding helps to verify and validate the data before storing and modifying it.

STEPS TO ACHIEVE DATA HIDING:

STEP 1: Prefix data members of a class with the private modifier.
STEP 2: Design a public getter and setter method.

PRIVATE MODIFIER:

- private is an access modifier.
- private is a class-level modifier.
- If the members of the class are prefixed with a private modifier, then we can access that member only within the class.
  NOTE: Data hiding can be achieved with the help of a private modifier.

GETTER METHOD:

- The getter method is used to fetch the data.
- The return type of the getter method is the type of the hidden **value.**

SETTER METHOD:

- The setter method is used to update or modify the data.
- The return type of the setter method is always void.

NOTE:

- The validation and verification can be done in this method before storing the data and before reading private data members.

NOTE:

- If you want to make your hidden data member-only readable then create only the getter method.
- If you want to make your hidden data member-only modifiable then create only the setter method.
- If you want to make your hidden data member both readable and modifiable then create both getter and setter methods.
- If you want to make your hidden data member neither readable and nor modifiable then don't create a getter and setter method.

ADVANTAGES:

- Provides security to the data members.
- We can verify and validate the data before modifying it.
- We can make the data member of the class to
  - Only readable
  - Only modifiable
  - Both readable and modifiable
  - Neither readable and modifiable

What is relationship?

- The connection (Association) between two object is known as the relationship.
  Types of relationship
  1)Has-a relationship
  2)Is-a relationship

1. **Has-a relationship**

- If one object is dependent on another object is known as has-a relationship
  a) aggregation
  b) composition

What is composition?

- The dependency between two object such that one object cannot exist without another object is known as composition.
  EXAMPLE:
  Car-Engine, Human-Oxygen, etc....

Aggregation

- The dependency between two objects such that one object can exist without the other is known as aggregation.
  EXAMPLE:
  Cab-Ola, Train-Online ticket booking, Bus-Passenger, etc

2. **IS-A RELATIONSHIP**

- The relationship between two objects which is similar to the parent and child relation is known as the Is-A relationship.
- In an Is-A relationship, the child object will acquire all properties of the parent object, and the child object will have its own extra properties.
- In an Is-A relationship, we can achieve generalization and specialization.

NOTE 1

- Parents are called generalized
- Children are called specialized.

Note 2

- Private members constructors and static members are not inherited to the child class.

EXAMPLE

- With the help of the child class reference type, we can use the members of the parent's class as well as the child.
- With the help of parent class reference, we can use only the members of the parent but not the child class.

PARENT CLASS

- The parent class is also known as a superclass or base class.

CHILD CLASS

- The child class is also known as a subclass or derived class.
  Note –
  Is A relationship is achieved with the help of inheritance.

INHERITANCE

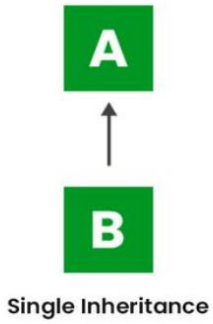
- The process of one class acquiring all the properties and behaviour from the other class is called inheritance.
  In java, we can achieve inheritance with the help of
  1. extends keyword
  2. implements keyword

Extends keyword

- extends keyword is used to achieve inheritance between two classes.
  Types of inheritance
  a. Single level inheritance
  b. Multilevel inheritance
  c. Hierarchical inheritance
  d. Multiple inheritance
  e. Hybrid inheritance

a. Single level inheritance

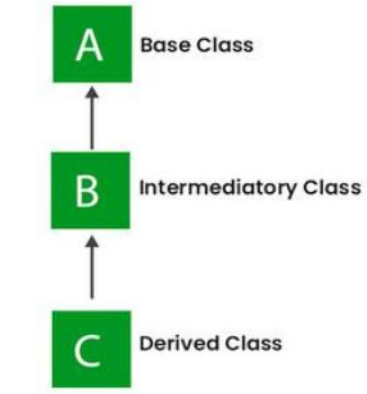
- One parent class is having one child class is known as single level inheritance.



Single Inheritance

b. Multilevel inheritance

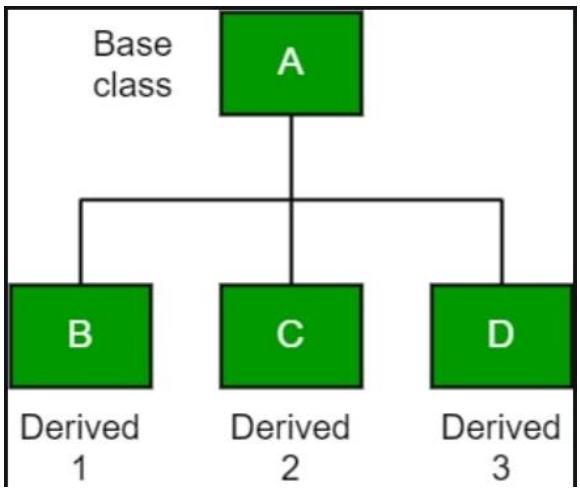
- One superclass is having one subclass and subclass is having another one sub class or one parent class having more than one sub child classes through the child is known as multilevel inheritance.



Multilevel Inheritance

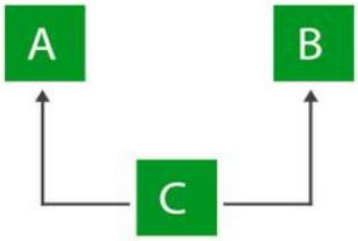
c. Hierarchical inheritance

- One parent class is having more than one child at same level or stage is known as Hierarchical inheritance.



d. Multiple inheritance

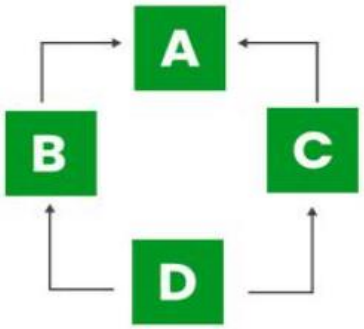
- One child is having more than one parent is known as multiple inheritance.



Multiple Inheritance

e. Hybrid inheritance

- Combination of more than one inheritance is known as hybrid inheritance.

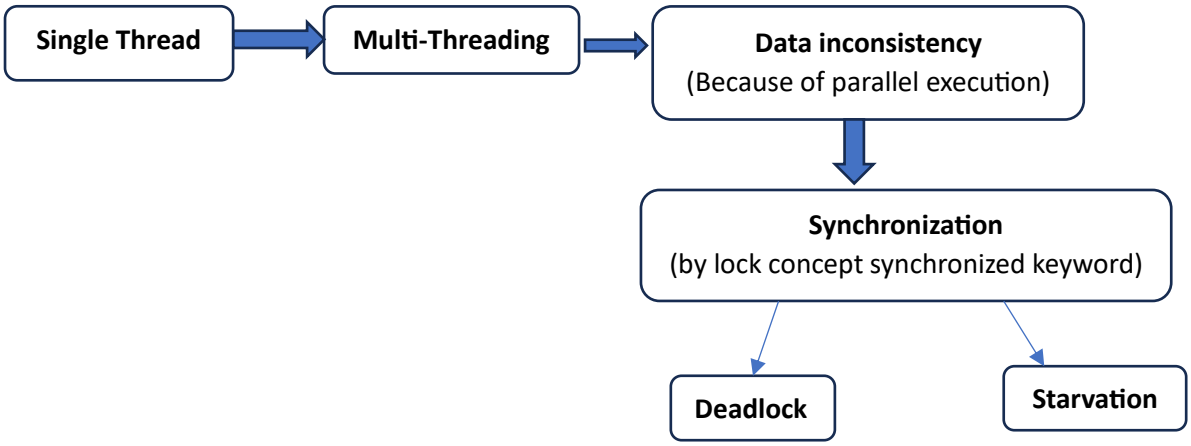


Hybrid Inheritance

NOTE

- Multiple inheritances have a problem known as the diamond problem.
- Because of the diamond problem, we can't achieve multiple inheritances with the help of class.
- In java, we can achieve multiple inheritances with the help of an interface.

Thread



Multithreading in Java is a process of exicuting multiple threads simultaneously.

Advantages of Java Multithreading

1) It doesn't block the user because threads are independent and you can perform multiple operations at the same time.

2) You can perform many operations together, so it saves time.

3) Threads are independent, so it doesn'affect other threads if an exception occurs in a single thread
[03-11-2023 09:51 PM] Jeewan: What is Thread in java

What is Thread in java

• A thread is a lightweight subprocess (the smillest executable unit of a process). It is a separate path of execution.

• Threads are independent. If there occurs exmeption in one thread, it doesn't affect other threads. It uses a share memory area.

• A process can have multiple threads.

What is process:

Process is an executing instance of an application For example, when you double click Ms Word icon in your computer, you start a process that wil run MS word application.
[03-11-2023 09:51 PM] Jeewan: Disadvantage of MultiThreaded Application

One of the major disadvantage of muti threaded system is Data inconsistency.
[03-11-2023 09:51 PM] Jeewan: Data Inconsistency

•Data inconsistency is a problem we facein multithreaded System, when multiple threads try to access the same resource(Object) at the same time.

Example:

Let us assume we have one reader thread and one writer thread to perform one task on the same Object. There is possibility of reader thread reading an invalid data(neither old data nor new.data)
[03-11-2023 09:51 PM] Jeewan: Disadvantage of MultiThreaded Application

One of the major disadvantage of muti threaded system is Data inconsistency.
[03-11-2023 09:51 PM] Jeewan: Synchronization

Synchronization:

• Synchronization is used to avoid data inconsistency occurred by interference of two threads in a sameObject.

• In java we can achieve synchronization with the help of synchronized keyword or modifier and lock concept(monitor);
[03-11-2023 09:52 PM] Jeewan: Synchronized

• It is a keyword and modifier.

• In java we can have the following

1. Synchronized non-static method

2. Synchronized static method

3. Synchronized block.

Note: A thread can be execute a synchronized part of code of an object only if the lock of the object is accrued (means if lock is not there it won't execute).
[03-11-2023 09:52 PM] Jeewan: Disadvantages of Synchronization

•Only one thread can access Synchronizel block of an Object, therefore if there are many other thread: who has to execute same block of the same object they go to wait for locks(this happens only in synchronised block).

•We generally come across two situations

1.Starvation

2.DeadLock
[03-11-2023 09:52 PM] Jeewan: Starvation

Freeze

Starvation:

Starvation describes the situation where thread is unable to join regular access to shared resource(Object

And unable to access any progress. This happens when shared resources are made unavailable for very ong time by greedy threads(High priority). This state is calledas Starvation
[03-11-2023 09:52 PM] Jeewan: Example of Stanetion

Freeze

Example:

Suppose an object provides a synchronized method whith normally takes long time to complete. If one thread invokes this method frequently o if there are many threads with high priority waiting to call the same method of the object ther a thread with low priority which also need synchronized access to the same object will be blocked for a very long time. This state of the thread become starvation.
[03-11-2023 09:53 PM] Jeewan: Deadlock

Freeze

•Deadlock describes the situation where wo or more threads are blocked forever, waiting for each other. (Adeadlock will occur due to Synchronization)

•Deadlock in Java is a part of multithreadng. Deadlock can occur in a situation when a thread is waiting for a object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread

•Since, both threads are waiting for each other to release the rock, the condition is called deadlock.
[03-11-2023 09:53 PM] Jeewan: Freeze

Deadlock

Starvation

All processes keep waiting for each other to complete and none get executed

Resources are blocked by the processes

Deadlock happens when every process holds a resource and waits for another process to hold another resource.

High priorty processes keep executing and low priorly processes are blocked

Resourcesare continuously utilized by high priority processes

Starvationhappens when a low priority program requests a system resource but cannot rur because a higher priority program his been employing that resource for a long me.
[03-11-2023 09:53 PM] Jeewan: Inter Thread Communication

Freeze

In java one thread can communicate with another tread with the help of Object class Method" such as wait(), wait(long t), wait(long t, in t1), notify() and notifyAll().

The main purpose of inter thread communication is o avoid the deadlock.

Wait(long):

It accepts long argument which behaves like millise onds.

The thread which went to wait stage when this method was called, will come back to runnable stage after the specified milliseconds of the even when it is not potiflec
Object class:

- Object class is defined in java.lang package.
- Object class is a super most parent class for all the classes in java.
- In object class there are 11 non static methods.
- One no argument constructor is there.
    1. public String toString ()
    2. public boolean equals (Object o)
    3. public int hashCode ()
    4. protected Object clone () throws CloneNotSupportedException
    5. protected vold finalize ()
    6. final public void wait () throws InterruptedException
    7. final public void wait (long l) throws InterruptedException
    8. final public void wait (long 1, int i) throws InterruptedExchan
    9. final public void notify () brows interruptedException
    10. final public void notifyAll () throws InterruptedException
    11. final public Class getClass ()

Why multiple inheritance not possible in java?

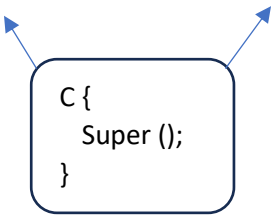
- because of Dimond ring problem of constructer.

Diamond problem because of constructor:

- assume that there are two classes A and B. both are not having user defined constructor. If class C inherits A and B then the time of creating the object of child class, we know that the child class constructor will call the parent class constructor which is called by no argument super () call statement to load then non static member of parent.

```
A {
   A ()
        }
```

```
B {
    B ()
}
```



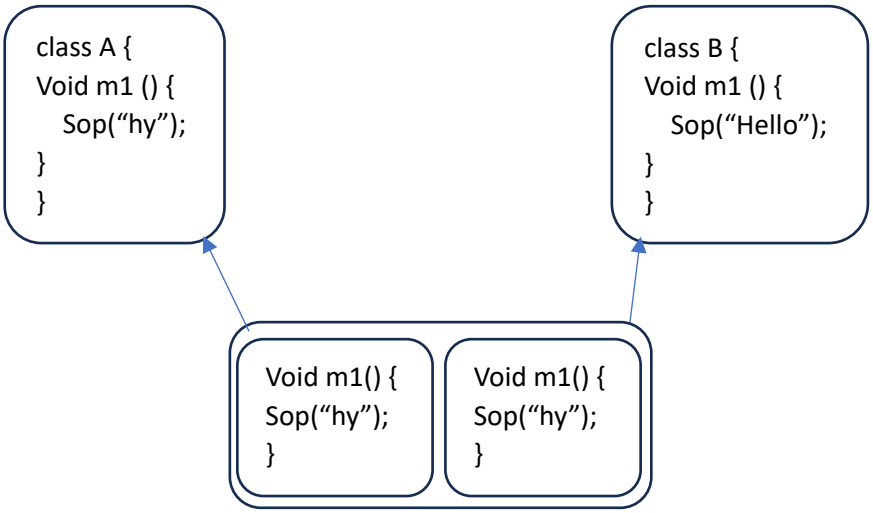
C c = new c();

Now an ambiguity arises when we try to call the superclass no argument constructor with the help of no argument super () call statement.

- This problem is known as the diamond problem.

Diamond problem because of method:

- Assume that there are two classes A and B. Both are having the method with the same signature. If class C inherits A and B then these two methods are inherited to C.



C c = new c ();

Now an ambiguity arises when we try to call the superclass method with the help of subclass reference.

- This problem is known as the diamond problem.

This keyword

- If we want differentiate between local variable and non-static variable, we have to use this keyword.

  Note:

  It is used to hold the current executing object references.

this ()

- this () is used to call one constructor from the other constructor of the same class.

  Rule
  - We have to use this () call statement in first line of constructor.

super keyword

- It is used to access the property of super class from the sub class.

super ()

- It is used to Call super class constructor from the sub class.

Rule

- We have to use super () call statement in first line of constructor.

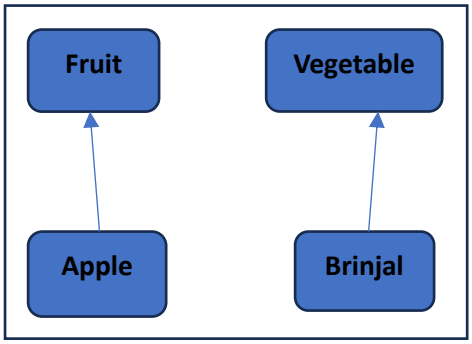
NON-PRIMITIVE TYPE-CASTING (DERIVED TYPE CASTING)

- The process of converting one reference type into another reference type is known as non-primitive or derived typecasting.

RULES TO ACHIEVE NON-PRIMITIVE TYPE CASTING:

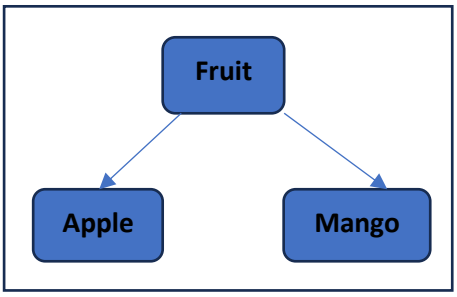
1. We can convert one reference type into another reference type only if it satisfies the following condition,
   - There must be an Is-A relation (Parent and child) exist between two references.
   - If the classes/Interfaces have a common child.

Ex 1 -



- Fruit can be converted to Apple and Apple can be converted to fruit as well as Vegetables can be converted to Brinjal and Brinjal can be converted to Vegetable.
- But Fruit and apple can't be converted to Vegetable and Brinjal as well as Vegetables and Brinjal can't be converted to Fruit and Apple.

Ex 2 –



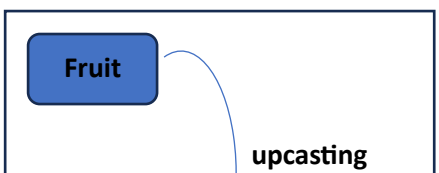
- Fruit can be converted to Apple as well as Mango and Mango and Apple can be converted into Fruit.
- But Apple can't be converted into Mango as well as Mango can't be converted into Apple.

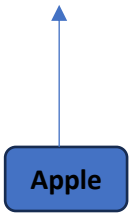
TYPES OF NON-PRIMITIVE OR DERIVED TYPE CASTING

Non-primitive type casting can be further classified into two types,
I. Upcasting
II. Down-casting

UPCASTING:

- The process of storing the child class object into a parent class reference type is known as upcasting.



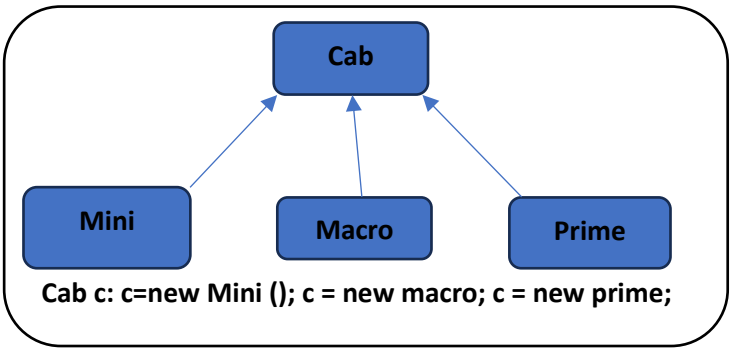


Note -

- The upcasting is implicitly done by the compiler.
- It is also known as auto upcasting.
- Upcasting can also be done explicitly with the help of a typecast operator.
- Once the reference is upcasted we can't access the members of the child.

WHY DO WE NEED UPCASTING?

- It is used to achieve generalization.
- It helps to create a generalized container so that the reference of any type of child object can be stored.
  Ex –



DISADVANTAGE

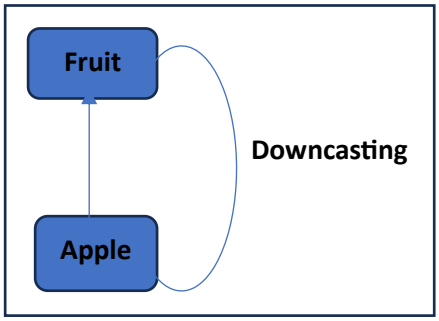
- There is only one disadvantage of upcasting that is, once the reference is upcasted its child members can't be used.
  NOTE:
  - In order to overcome this problem, we should go for down casting.

DOWNCASTING

- The process of converting a parent (superclass) reference type to a child (subclass) reference type is known as down casting.
  NOTE
  - Down casting is not implicitly done by the compiler.
  - It should be done explicitly by the programmer with the help of a typecasting operator.

WHY DO WE NEED A DOWNCASTING?

- If the reference is upcasted, we can't use the members of a subclass.
- To use the members of a subclass we need to downcast the reference to a subclass.



ClassCastException

- It is a RuntimeException.
- It is a problem that occurs during runtime while down-casting.

When and why do we get a ClassCastException?

When we try to convert a reference to a specific type(class), and the object doesn't have an instance of that type then we get ClassCastException.

EXAMPLE:

```
Case 1:
Child c = (Child)new Parent (); //ClassCastException

Case 2:
Parent p=new Parent ();
Child c=(Child)p;//ClassCastException
```

INSTANCEOF OPERATOR:

instanceof operator:

- It is a binary operator
- It is used to test if an object is of the given type.
- The return type of this operator is Boolean.
- If the specified object is of a given type, then this operator will return true else it returns false.

Syntax to use instance of operator:

```
(Object_Ref) instanceof (type)
```

Ex Class 1

```
class A {
        int x = 5;
        int y = 10;
        }
```

POLYMORPHISM

Polymorphism is derived from two different Greek words 'Poly' means Numerous; 'Morphs' means form Which means Numerous forms. Polymorphism is the ability of an object to take on multiple forms, typically using method or functions with the same name.

For Understanding:

One name → Multiple forms
One variable name -> Different values
One method name -> Different behaviour

TYPES OF POLYMORPHISM:

In java, we have two types of polymorphism,

1. Compile-time polymorphism
2. Runtime Polymorphism

COMPILE-TIME POLYMORPHISM:

- If the binding is achieved at the compile-time and the same behaviour is executed at run time is known as compile-time polymorphism.
- It is also said to be static polymorphism.

NOTE:

Binding means an association of method call to the method definition.

It is achieved by:

1. Method overloading
2. constructor overloading
3. Variable shadowing
4. Method shadowing
5. Operator overloading (does not support in java)

RUNTIME POLYMORPHISM:

- If the binding occurs at compile time but different behaviour is achieved at the runtime then it is known as runtime polymorphism.
- It is also known as dynamic binding.
- It is achieved by method overriding.

METHOD OVERLOADING

- If more than one method is created with the same name but different formal arguments in the same class are known as method overloading.

Different formal arguments mean:
Differ in no. of FA
Differ in type of FA
Differ in order of FA

EXAMPLE:

java.io.PrintStream;
println ()
println(int)
println (double d)
println (String s)

These are some of the overloaded methods (methods with the same name but different formal arguments) implemented in java.io.PrintStream class.

CONSTRUCTOR OVERLOADING

- A class having more than one constructor with different formal arguments is known as constructor overloading.

```
Class A
{
  A ()
  { }
  A (int a)
  { }
}
```

METHOD SHADOWING:

- If a subclass and superclass have the static method with the same declaration, but different in implementation is known as method shadowing.
- Which method implementation gets executed, depending on what?
- In method shadowing binding is done at compile-time, hence it is compile-time polymorphism. The execution of the method depends on the reference type and does not depend on the type of object created.

NOTE:

- The return type should be the same or it should be a covariant return type.
- Access modifier should be same or higher visibility than superclass method. Method showing is applicable only for the static method. It is compile time polymorphism
- Execution of implemented method depends on the reference type of an object.

EXAMPLE: Method Shadowing

```
class Parent {
        public static void test () {
                System.out.println("From parent");
        }

class Child extends Parent {
        public static void test () {
                System.out.println("From child");
        }
        public static void main (String [] args) {
                Parent p = new Child ();
                p.test (); // from parent // this is based on the ref variable type

                Child c = new Child ();
                c.test (); //from child // this is based on the ref variable type

                Parent p1 = new Parent ();
                p1.test (); // from parent // the based on the ref variable type
        }
}
```

VARIABLE SHADOWING

- If the superclass and subclass have variables with the same name but change in values then it is known as variable shadowing.

Which variable is used, depending on what?

- In variable shadowing binding is done at compile-time, hence it is a compile-time polymorphism. The Variable used depends on the reference type and does not depend on the type of object created.

NOTE

- It is applicable for both static and non-static variables.
- It's a compile-time polymorphism.
- Variable usage depends on the type of reference and does not depend on the type of object created.

EXAMPLE: Variable Shadowing

```
class Parent {
        int x = 10; static int y = 9;
        }
```

```
class Child extends Parent {
        int x = 20;
        static int y = 19;
        public static void main (String [] args) {
                Parent p = new Child ();
                Sopln (p.x); // 10 // this is based on the ref variable type
                Sopln (p.y); // 9

                Chic = new Child ();
                Sopln (c.x); // 20 // this is based on the ref variable type
                Sopln (c.y); // 19

                Parent p1 = new Parent ();
                Sopln (p1.x); // 10 // this is based on the ref variable type
                Sopln (p1.y); // 9
        }
}
```

METHOD OVERRIDING

- If the subclass and superclass have non static methods with the same declaration and different in implementation, it is known as method overriding.

Rule to achieve method overriding:

- Is A relationship being mandatory.
- It is applicable only for non-static methods.
- The signature of the subclass method and superclass method should be the same.
- The return type of the subclass and superclass method should be the same until the 1.4 version but, from the 1.5 version, covariant return type in the overriding method is acceptable (subclass return type should be the same or child to the parent class return type.).
- Access modifier should be some or higher visibility than superclass method.

EXAMPLE: Method Overriding

```
class Parent {
        public void test () {
                System.out.println("From parent");
        }

class Child extends Parent {
        @Override
        public void test () {
                System.out.println("From child");
        }
        public static void main (String [] args) {
                Parent p = new Child ();
                p.test (); from child // this is based on the object created

                Child c = new Child ();
                c.test (); // from child" // this is based on the object created

                Parent p1= new Parent ();
                p1.test (); // from parent // this is based on the object created
        }
```

}

EXAMPLE:

```
Child c = new Child ();
c.test(); // from child
Parent p = c;
p.test(); // from child
```

Internal runtime object is a child so child test () will get executed, it does not depend on the reference type.

NOTE: Variable overriding is not applicable.

@Override

- It is an annotation which giving the more information to the compiler about the operation we are performing which is overriding.
- Also by this one we can specify the method is declared in parent but child is only changing the implementation.
- It is not mandatory to use. But highly recommended to use.
- If we will use, compiler will check the overriding process in prior.

Abstraction?

- It is a design process of hiding the implementation and showing only the functionality (only declaration) to the user is known as abstraction.

HOW TO ACHIEVE ABSTRACTION IN JAVA?

- In java, we can achieve abstraction with the help of abstract classes and interfaces.
- We can provide implementation to the abstract component with the help of inheritance and method overriding.

ABSTRACT MODIFIER:

- The abstract is a modifier, it is a keyword.
- It is applicable for methods and classes.

ABSTRACT METHOD:

- A method that is prefixed with an abstract modifier as well as does not have the implementation and end with semicolon is known as the abstract method.
- This is also said to be an incomplete method.
- It is generally non static in nature.
- It can't be private or static or final.

Syntax to create abstract method:

abstract [access modifier] returnType methodName([For_Arg]);

NOTE:

Only child class of that class is responsible for giving implementation to the abstract method.

Abstract Class

- If the class is prefixed with an abstract keyword / modifier then it is known as abstract class. We can't create object (instance) for the abstract class.

Characteristics:

1. We can't create an instance of an abstract class.
2. We can have abstract class without an abstract method.

3. An abstract class can have both abstract and concrete method.
4. If a class has at least one abstract method which is either declared or inherited but not overridden, then it is mandatory to make the class as abstract class.

Final keyword

- Final is keyword.
- It is also known as modifier.
- It is applicable for class, variable and methods.

If class will be Final

- If class will be final we can't inherit that class (we can't make the child of this class)
- final class prevent inheritance.

Example: Demo.java

```
final class Demo {
        int x=89;
        public void m ()
                {
                        System.out.println("m ());
                }
}
```

Demo1.java

```
class Demo1 extends Demo {
        public static void main (String [] args) {
                Demo1 d= new Demo1();
                System.out.println(d.x);
                d.m();
        }
}
```

error: cannot inherit from final Demo class Demo1 extends Demo

If variable will final

- If variable will be final, we can't reassign the value of the declared variable.
- final variable prevents reassign or update the value

Convention Final Variable

- If we are taking any final variable in our class, we have to make the variable name as in UPPER CASE. For example: if variable name is age the we have declare like final int AGE = 20;

Ex:

```
class Prog {
final int AGE = 10;
public static void main (String [] args) {
        Prog p = new Prog ();
        System.out.println(p.AGE); 10
        p.AGE= 20; // reassigning the value
        System.out.println(p.AGE);
}
```

Compile Time Error:

```
error: cannot assign a value to final variable x
p.x = 20;// reassigning the value
```

If method will be Final

- If method will final, we can't override the method.
- Final method prevent method overriding

Ex:

```
class Father {
        final public void home () {
                System.out.println("Colour: Red");
        }
}

class Son extends Father
        @Override
        public void home () {
                System.out.println("Color: Yellow");
        }

        public static void main (String [] args) {
                Son s = new Son ();
                s.home();
        }
}

Compile Time Error:
        error: home () in Son cannot override home () in Father
        public void home ()
overridden method is final
```

Abstract Class

EXAMPLE:

```
abstract class Atm {
        abstract public double withdrawal ();
        abstract public void getBalance ();
        abstract public void deposit ();
}
// hiding implementation by providing only functionality
Atm a = new Atm () // CTE
```

NOTE:
Only subclass of Atm is responsible for giving implementation to the methods declared in an Atm class.

When you have to go for Abstract Method

1. When we don't have the clear idea about the implementation of the method
2. If want to leave the implementation of the method for the child who should override the method and give the implementation.

Implementation of abstract method:

- If a class extend abstract class, then it should give implementation to all the abstract method of the superclass.
- If inheriting class doesn't like to give implementation to the abstract method of superclass, then it is mandatory to make subclass as an abstract class.

- If a subclass is also becoming an abstract class then the next level child class is responsible to give implementation to the abstract methods.

STEPS TO IMPLEMENT ABSTRACT METHOD:

STEP 1: Create a class.
STEP 2: Inherit the abstract class/component.
STEP 3: Override the abstract method inherited (Provide implementation to the inherited abstract method).

Example of giving the implementation to abstract method

EXAMPLE:

```
abstract class WhatsApp {
        abstract public void send ();
}
class Application extends WhatsApp {
        public void send () {
                System.out.println("Send () method is implemented");
        }
}
```

Creating object and calling the abstract method:

```
WhatsApp w = new WhatsApp (); // not possible
Application a = new Application ();
a.send ();
```

Note:
We can't create the object of abstract class but we can create the reference variable of abstract class and store the child reference which is concrete

```
WhatsApp w = new Application ();
w.send(); //O/P: Send() method is implemented
```

Here send () is not the child class method, it is declared in the parent and only implementation is given by child. So, it is possible to access the member by the parent reference variable.

CONCRETE CLASS:

- The class which is not prefixed with an abstract modifier and doesn't have any abstract method, either declared or inherited is known as concrete class.
  NOTE: In java, we can create objects only for the concrete class.

Interface

- It is a component in java which is used to achieve 100% abstraction and multiple inheritance.
  Syntax to create an interface

```
[Access Modifier] interface InterfaceName
        {
                // declare members
        }
```

Case 1:

```
interface Demo1 {
        public static void main (String [] args) {
                System.out.println("Hello World..!!!");
```

```
        }
}
```

case 2:

```
interface Demo1 {
        int a; //CTE: Variable a is by default public, static, final. A final variable must be initialized.
}
```

EXAMPLE:

```
interface Demo {
        void m1();
        void m2();
        public void m3();
        static final int y = 7;
}
```

The Demo is an interface.
In the interface all the members by defaultly public in nature. And abstract methods are by defaultly public and abstract.

What all are the members they can be declared in an interface?

MEMBERS	CLASS	INTERFACE
Static variables	Yes	Yes, but only final static variables
Non-static variables	Yes	No
Static methods	Yes	Yes, From JDK 1.8 v. NOTE: They are by default public in nature
Non-static methods	Yes	Yes, but we can have only abstract non-static methods NOTE: Non-static methods are by default • public • abstract
Constructors	Yes	No
Initializers (Static & non-static block)	Yes	No

NOTE:
In interface, all the member are by default have public access modifier.

Why do we need an interface?

- To achieve 100% abstraction. Concrete non-static methods are not allowed.
- To achieve multiple inheritances.

What all the members are not inherited from an interface?

- Only static members of an interface are not inherited to both class and Interface.

```
interface Demo2 {
        public void test () // CTE
        {
                //statement
        }
}
```

Note: inside the interface non static concrete method is not allowed.

INHERITANCE WITH RESPECT TO INTERFACE

- An Interface can inherit any number of interfaces with the help of an extends keyword.

EXAMPLE: Single level inheritance

```
        interface I1 {
```

```
        //statements
}

interface I2 extends I1 {
        //statements
}
```

Note:

The interface which is inheriting an interface should not give implementation to the abstract methods. It should be given by any of the child class.

Note:

We can't create the object for the interface, but we can create the reference variable of interface and store the child type object reference.

Extends and Implements keyword

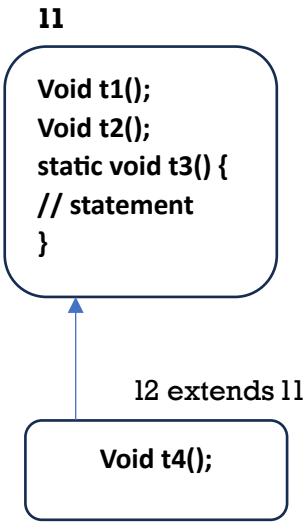
Extends:

- We can achieve inheritance in between class and class, or interface and interface by extends keyword.

Implements:

- To achieve inheritance In between interface and class we are using implements keywords.

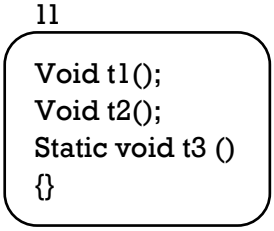
Single level inheritance

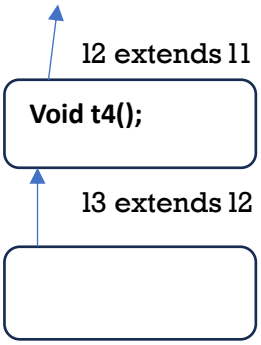


EXAMPLE 1:

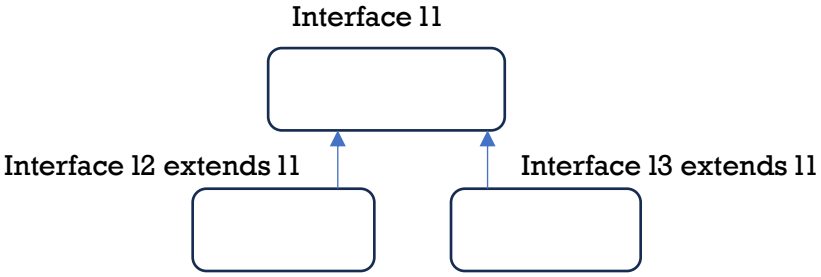
- Interface I1 have 3 methods
  2-non static abstract (t1(), t2())
  1- static (t3())

- Interface I2 have 3 methods
  2- inherited non static abstract methods (t1(), t2())
  1- declared non static abstract methods (t4())

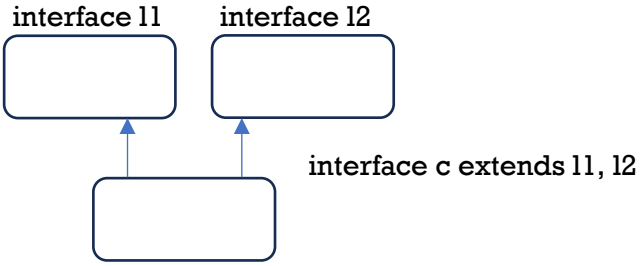




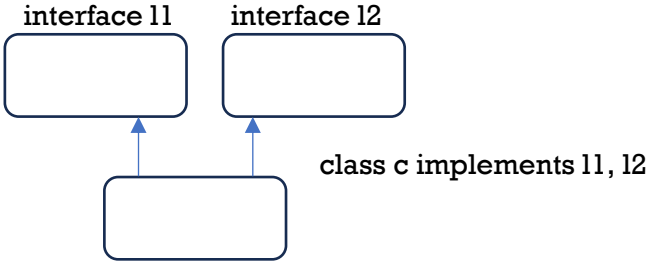
Hierarchical inheritance



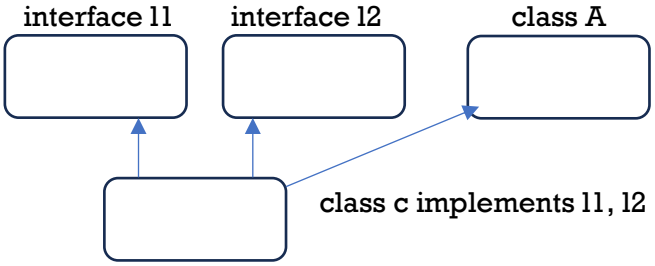
Multiple inheritance with respect to interface



Multiple inheritance with respect to class



Multiple inheritance with respect to both class and interface



NOTE:

With respect to interface there is no diamond problem.

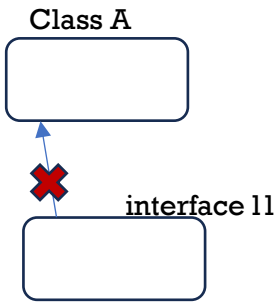
The reason,

- They don't have constructors.
- Non-static methods are abstract (do not have implementation)
- Static methods are not inherited.

INHERITANCE OF AN INTERFACE BY THE CLASS:

- Class can inherit an interface with the help of implements keywords. (class can be child of interface)
- Class can inherit more than one interface. (One class can have more than one interface as parent)
- Class can inherit a class and an interface at a time.
  Note: A class can't be a Parent of interface.

Class can't be the parent of interface.



Class can't be the parent of the interface because in every class by defaultly 11 non static concrete methods of object class are there. If we make class will be parent, then all methods will be inherited to the child but inside the interface concrete non static methods are not allowed.

NOTE:

- If a class inherits an interface, then it should give implementation to the abstract non-static methods of an interface.
- If the class is not ready to give implementation to the abstract methods of an interface then it is mandatory to make that class an abstract class.
- The next level of child class is responsible for giving implementation to the rest of the abstract methods of an interface.

Type of Interface

There are 3 types of interface.

1. Regular Interface
2. Functional Interface
3. Marker Interface

Regular Interface

- Regular interface is a interface which contains more than one abstract method.
  Ex:

```
public interface Imouse {
        public void click ();
        public void rightClick ();
        public vo doubleClick ();
}
```

Functional Interface

Functional interface is an interface which contains only one abstract method.
Ex: Comparable, Comparator, Runnable etc.
In Comparable compareTo () method is there
In Comparator compare () method is there

In Runnable run () method is there

Marker Interface
Marker interface is an interface which does not have any method in it.
It mainly used to indicate jvm about the certain activity.

We have some marker interface as follows:
1. Cloneable
2. Serializable
3. Random-access etc.

Note: After JDK 1.8 in interface static concrete and default concrete method are allowed.

When an interface is compiled, we get a class file with an extension .class only

Q. Can we make our constructor final?
Ans: No, we are using final keyword with method for preventing the override of method. As constructor is not inherited only to the child, then what is the use of make it final. That's why it is not possible.

Q. Is constructor inherited from parent to child? Why? or
Why constructor is not inherited from parent to child?
Ans: No, as constructor name same as class name, if it is inherited to the child, then in the child class, the constructor's name must be same as child class name which is not possible. And as constructor has no return type it won't considered as method also.

Q. Can we make our constructor private?
Ans: yes, we can. If we declare a constructor private, we can create object of the class within the same class only, outside the class we can't create object of the class. Creating object inside the same class only is nothing but nature of Singleton class.
Q1.1 When we should go for private constructor?
Ans:
a) If we don't want to allow to create an object of a class outside of the class.
b) When there is no subclass to be created for the class which which has the private constructor. Because a private constructor does not allow make sub class of its own class.

Q. can we make constructor as static?
Ans: No, it is by-defiantly non-static, because it will call the time of object creation.

Q. Can we make abstract method private?

Q. Can we make abstract method static?

Q. Can we make abstract method final?
Q. Can we make class private?

Q. Can we make static method final?

Q. Can we make non-static method final?

Q. Can we make main method final?

Q. What extra modifier we can use with main method?

Q. is final method inherited to the child?

Q. Can we override the final method?

Q. how many ways are there we can stop the inheritance?

Part - III

Package

A package in java is used to group a related classes, interface and subclasses. In simple word it is a folder or directory which consists of several classes and interface.
Note: Package contains only class File.

Why Package?

- Packages are used to avoid name conflict.
- It increases maintainability.
- It is used to categorize classes and interface.
- It increases the access protection.
- It is used to achieve code reusability.

Note
If we want to use package, and import statement in the same program
Then we have to follow following sequence:
1. package
2. import statement
3. class/interface

Array:

Array is a continuous block of memory which is used to store multiple homogeneous values / data.

Characteristics of An Array

- The size of an array must be defined at the time of instantiation.
- Once array is created the size of array can't be modified.
- Hence array is known as fixed size.

- In an array we can access the elements with the help of an index. Index is an integer number that starts from 0 and ends at (length of the array - 1).
- In an array we can store only homogeneous collection of an object.
  Note: In java array is an Object.

Declaring, instantiating, initializing an array in a single line

We can also declare, instantiate, and initialize an array in a single line.
Syntax:

```
Datatype [] arr_ref_variable = {element 1, element 2, element 3 etc...};
```

Ex:

```
int y= {5, 7, 9, 11, 13};
```

length variable in array

length:

- In java, the number of elements of an array can hold or the size of the array is known as length.
- You can use the length variable to find the size of an array by using the dot operator with the array name.
  Ex: int [] x = new int [5];
  System.out.println(x.length); // 5

Accessing elements from an array

We can access an element from an array with the help of array reference variable and index.
Syntax:

```
Array_ref_variable [index];
```

Ex:

```
System.out.println(x[0]);// 4
System.out.println(x[1]); // 5
System.out.println(x[2]); // 6
System.out.println(x[3]); //ArrayIndexOutOfBoundsException
```

Access the array elements by loop

By using the loop, you can access the array elements in forward direction.
Ex:

```
Int [] = {6,7,8,9);
for (int i = 0; i < x.length; i++) {
        System.out.print(x[i] +" ");
} //OP: 6789
```

Access the array elements by loop in backward direction

By using the loop, you can access the array elements.
Ex:

```
int x= {6,7,8,9};
for (int i =x.length-1 ;i>=0; i--) {
        System.out.print(x[i] +" ");
} //OP: 9876
```

Access the array elements by forEach loop

1. By using the for each loop you can access the array elements only in the forward direction.
2. It is introduced from jdk 1.5.
3. No condition is required for this loop.

Syntax:

```
for (datatype_of_array_created var: array_variable) {
        //stmt // System.out.println(var);
        }
```

Access the array elements by forEach loop

Ex:

```
int x = (6,7,8,9);
for (int i: x) {
        System.out.print(i + " ");
}//OP: 6789
```

Single line multi-dimensional array declaration, instantiation and initialization

Syntax:

```
Datatype arr_ref_variable [][] = {{(0,0), (0,1),....(n,n)}, {(1,0), (1,1).....(n,n)}};
```

Ex -

```
int x[][] = {{3,4,5), (2,4,6},{1,2,3}};
```

Access the multi-dimensional array by

```
int x [][] = {{4,4},{5,5), (6,6}};
System.out.println(x[0][0]);
System.out.println(x[0][1]);
System.out.println(x[1][0]);
System.out.prin_n(x[1][1]);
System.out.println(x[2][0]);
System.out.println(x[2][1]);
```

Access the multi-dimensional array by using the single printing statement

```
int x [] [] = {{4,4},{5,5}, {6,6}};
for (int i = 0; i<x.length;i++) {
        for (int j = 0; j<x[i].length;j++){
                System.out.println(x[i][j]);
                }
        }
```

Syntax to create or declare an array:

```
Datatype [] variable:
Or
Datatype [] variable;
Or
Datatype [] variable;
Or
Datatype variable [];
```

Examples

```
int [] x; or int [] x; or int []x; or int x [];
double [] y;
String [] name;
boolean attendance [];
char [] ch; etc.
```

How to abbreviate

- int [] x; x is single dimensional array reference variable of int type
- int [] y; y is single dimensional array reference variable of int type
- double d []; d is single dimensional array reference variable of double type.
- String [] s; s is single dimensional array reference variable of String type.

Instantiating an array

Syntax:

new datatype [Size];

Ex:

new int [5];
new String [7];
new double [4];
new boolean [3];

Note: Once array is instanced, every index of the array is assigned with the default value with respect to default value of datatype.

Initializing an Array

Add Elements on array:

We can add an element into the array with the help of array index.

Syntax

Array_ref_variable [index] = value;

Ex:

```
Int [] x;
x= new int [3];
x [0] = 4;
x [1] = 5;
x [2] = 6;
x [3] =7; // ArrayIndexOutOfBoundExotion
```

Declaring and instantiating array in single line

We can also declare the array and instantiate the array in single line.

Syntax:

Datatype [] variablename = new datatype [size];

Ex: int x = new int [5];

Strings = new String [3];

Arrays Class

- It is defined in java.util package.
- It has so many methods inside it and most of the methods static in nature.

As methods static in nature, to call the methods we have to use the Arrays class name.

For example: Arrays.sort(array_reference variable)

```
int x = {2,7, 5, 8,4,9);
Arrays.sort (x); // Sort method work is, it will sort the array element in ascending order.
```

Difference between Array and Arrays

Array is non- primitive literals. But Arrays is a predefined class in java which is declared in java.util package.

Multi-Dimensional Array

The design process of storing the elements in the form of rows and columns (Matrix format) based on index value is known as multi-Dimensional array.

Syntax to Declare Multi-dimensional array:

Datatype [] [] arr_ref_var; or Datatype [] [] arr_ref_var;

Datatype [] [] arr_ref_var; or Datatype arr_ref_var [] []; etc.

Instantiation of Multi-Dimensional Array

To create 2 d' array object:

Syntax: new datatype [row_size] [colum_size];

Ex: new int [3] [3];

It will create an array of row size 3 and column size 3. it means it will create an of size 3 which can store the address of the 3 integer elements.

Single line array declaration and instantiation

Syntax:

datatype [][] arr_ref_ var = new datatype [row_size] [column_size];

Ex:

```
double d [] [] = new double [6][6];
String s [] [] = new String [4] [ 3];
boolean b [] [] = new boolean [2][3];
char ch [] [] = new char [4] [4]; etc.
```

Initialization of multi-dimensional array

We can add the elements into array as follow the below syntax:

Syntax:

array_ref_variable [row_indx] [colum_index] = value;

Ex:

```
int arr[][] = new int[3] [3];
arr [0][0] = 65; arr[0][1] = 67;
arr[1] [0] = 68; etc.
```

Access the multi-dimensional array by using the single printing statement by for each loop

```
int x [] [] = {{4,4},{5,5}, {6,6}};
for (int [] i:x) {
        for (int j: i) {
                System.out.println(j);
        }
}
```

String Literals

- String is non-primitives a literal (deta).
- It is a group of character that is enclosed within the double quotes (**);
  Ex: "Dev", "A", "1", "*" etc.
- We can store string data or literal directly by taking inside the double quotes.
- In java it is possible to store Sting data by creating the object of the following:
  1.String class
  2.StringBuffer class
  3.StringBuilder class
- In java when we create a string, compiler will implicitly create a object String class inside the String Constant Pool or String Pool Area.

Characteristics of String Literals

- In Java when we create a string, compiler will implicitly create a object for String class inside the String Constant Pool or String pool area.
- If we will create any of the new String, then it will check 1st inside the string constant pool the given is present or not. if it will present then instead of creating the new String object, it will give existing reference of the previous object.

String Class

- String is the predefined class which is defined in the java.lang package.
- It is a final class.
- It inherits java.lang. Object class
- It is very much Case-Sensitive.
- Inside the String class toString(), equals(Object o), hashCode() methods object class are overridden.

- Its implements Comparable, Serializable and CharSequence interface.
  Ex:
  ```
  string s = "Hy";
  String s1rew String("Hello");
  ```

toString()

- toString() method returns String.
- toString() implementation of Object class returns the reference of an object in the String format.

Return Format: ClassName@HexaDecimal

EXAMPLE:

```
class Demo {
    public static void main (String [] args) {
    Demo d = new Demo();
    System.out.println(d) // d.toString() – Demo@192e014
    }
}
```

NOTE

- Java doesn't provide the real address of an object.
- Whenever programmer tries to print the reference variable toString() is Implicitly called.

PURPOSE OF OVERRIDING toString():

- We override toString() method to return state of an object instead of returning reference of an object.

EXAMPLE:

```
class Circle {
    Int radius;
    Circle (int radius) {
        this.radius = radius;
    }
    @Override
    public String toString() {
        return "radius: "+radius;
    }
    public static void main(String[] args) {
        Circle c = new Circle(5);
        System.out.println(c) // c.toString()---radius: 5
    }
}
```

equals(Object):

- The return type of equals(Object) method is boolean.
- To equals(Object) method we can pass reference of any object.
- The java.lang.Object class implementation of equals(Object) method is used to compare the reference of two objects.

EXAMPLE 1:

```
class Book {
    String bname;
        Book(String bname) {
            this.bname = bname;
        }
    }
```

Case 1

```
Book b1 = new Book("java");
```

```
Book b2=b1;
S.o.pln(b1.bname); // Java
S.o.pln(b2.bname); // Java
S.o.pln(b1==b2); // true
S.o.pln(b1.equals(b2)); // true
```

Case 2

```
Book b1 = new Book("Java");
Book b2= new Book("Java");
S.o.pin(b1.bname); // Java
S.o.pln(b2.bname); // Java
S.o.pin(b1==b2); // false
s.o.pin(b1.equals(b2)); // flase
```

NOTE

- If the reference is same == operator will return true else it returns false.
- The equals (Object) method is similar to == operator.

PURPOSE OF OVERRIDING equals(Object):

- We can override to equals(Object) method to compare the state of an two Objects instead of comparing reference of two Objects.

  NOTE:
  - If equals(Object) method is not overridden it compares the reference of two abjects similar to operator.
  - If equals(Object) method is overidden compares the state of two objech in such case comparing the reference of two objects is possible only by operator.

Design tip:

- In equals method compare the state of an current (this) object with the passed object by downcasting the passed object.

  EXAMPLE:

```
class Book {
        String bname:
        Book (String bname) {
                this.bname = bname;
        }
@Override
public boolean equals (Object o) {
        Book b= (Book)o:
        if(this.bname.equals(b.bname))
                return true;
        else
                return false;
        }
}
```

hashcode ():

- The return type of hashCode () method is int.
- The java.lang.Object implementation of hashCode() method is used to give the unique integer number for every object created.
- The unique number generated based on the reference of an object.

PURPOSE OF OVERRIDING hashcode ():

- hashcode () method should return an integer number based on the state of an object.

For more validation is required.

Note:

If the equals (Object) method is overridden, then it is necessary to override the hashcode () method. If not then both equals () and hashcode () result will be differ which is not expected.

In the above two cases it is clear that,

- If the hashcode for two object is same, equals (Object) method will return true.
- If the hashcode for two object is different, equals (Object) method will return false.

Wrapper class:

- In java every primitive datatype has a corresponding class which works like a wrapper for primitive. Hence this class is known as wrapper class.
- The wrapper class in java provides mechanism is wrap the primitive into object.
- All the wrapper classes are final classes present in java.lang package.

Primitive data type	Wrapper class
Boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
float	Float
double	double

Among these wrapper classes Byte, short, integer, long, float, double are subclasses of number class.

User of wrapper class:

- To represent primitive data in the form of corresponding wrapper object.
- To convert string type primitive data into original primitive type (Except character).

To represent primitive data in the form of corresponding wrapper object.

- Byte b = 10; //auto boxing
- System.out.println(b); // 10 [not reference]
- We are not getting reference of byte object because the to string is overridden inside wrapper class.

  Eg –

  ```
  int i = 2578;
  integer i1 = new Integer(i);
  Sopln(i); //2578
  Sopln(i1); //2578
  ```

We are not getting reference of integer object because the toString() method is overridden inside wrapper class.

Autoboxing:

- It is a process of automatically implicit conversion of primitive datatype into its corresponding non primitive datatype.

  Eg –

  byte to Byte
  short to Short
  char to Character
  int to Integer etc.
  Byte b = 50; //directly we can store the value inside the variable because auto boxing is happening

```
Boolean bi = true;
```

Unboxing / auto unboxing

- The process of automatic or implicit conversions of non-primitive wrapper datatype into its corresponding primitive datatype.
  Ex –
  Integer x1 = new Integer (520);
  Int x2=x1;
  System.out.println(x2); //520

To convert string type primitive data into original primitive type:

- Every wrapper class has a static method which is used to convert string representation of primitive to actual primitive.
  These methods are called pare method. the process is known as parsing.
  - Byte: public static byte parseByte (String s) throws NumberFormatException.
  - Short: public static short parseShort(String s) throws NumberFormatException.
  - Integer: public static int parseInt(String s) throws NumberFormatException
  - Long: public static long parseLong(String S) throws NumberFormatException
  - Float public static float parseFloat(String s) throws NumberFormatixception
  - Double: public static double parseDouble(String s) throws NumberFormatException
  - Boolean: public static boolean parseBoolean (String S)
    Ex:

```
int x = Integer.parseInt("10");
Sopln(x); //10

doubled Double.parseDouble("58.5");
Sopln(d); //58.5

boolean b Boolean.parseBoolean("true");
Sopln(b); //true
```

String to Boolean Conversion:

- It doesn't follow case sensitive and if it only contains ‘true’ then it converted into 'true'.
- Or else for any other value including false value it will convert into false.

NumberFormatException:

- Whenever we are trying to convert any String to primitive number type if the content of string is not number type
- it will throw an exception which is known as NumberFormatException.
  NOTE:
  - We can't convert String to char that's why there is no such method like parseChar (String). // it can be done by charAt(index) method
  - parseBoolean(String s) method won't throw NumberFormatException.
  - For String to primitive conversion autoboxing and auto unboxing does not work. we have to do it explicitly.

String literal:

- Anything enclosed withing the double quote “… “in java is considered as string literal.
  Characteristics of string literal:
  - When a string literal is used in a java program, an instance of java.lang.String class is created inside a string pool.
  - For the given string literal. If the instance of a string is already present then new instance is not created instead the reference of a existing instance is given.

String:

- The string is literal (data). It is a group of characters that is enclosed within the double quote'' ''.
  - It is non-primitive data.
  - In java, we can store a string a string by creating instances of the following classes.
    Java.lang.String
    Java.lang.StingBuffer
    Java.lang.StringBuilder
  - In java, whenever we create a string complier implicitly create an instance for java.lang.string in string pool area/string constant pool(scp).

```
Example 1
class Demo {
    public static void main(String[] args) {
    system.out.println(“Hello”);//String@0123
    system.out.println(“Hello”);//String@0123
    }
}

Example 2
class Demo {
    public static void main (String [] args) {
    string s1.s2;
    s1 = “Hello”;
    s2 = “Hello”;
    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s1==s2); //true
    System.out.println(s1.equals (s2)); //true
    }
}
```

Java.lang.String:

- String is a inbuilt class defined in java.lang package.
- It is a final class.
- It inherits java.lang.object.
- In a string class toString[], equals(), hashCode() methods od java.lang.Object class are overridden.

It implements:

- Comparable
- Serializable
- charSequence

constructors:

Constructors	**Description**
String ()	Creates an empty string object
String (String literals)	Creates string object by initializing with sting literals
String (StringBuffer sb)	Creates string try converting stringbuffer to string
String (StringBuilder sb)	Creates string by converting stringBuilder to string.
String (char [] c)	Converting char type array to sting type.

```
Example 3 –
class Demo {
    public static void main (String [] args) {
```

```
        String s1.s2;
        S1 = "Hello";
        S2 = new string("Hello");
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s1==s2); //false
        System.out.println(s1.equals(s2));//true
        System.out.println(s1.hashCode()==s2.hashcode());//true
        }
}
```

Return type	**Method name**	**Description**
String	toUpperCase()	Converts the specified string to upper cas.
String	toLowerCase()	Converts the specified string to Lowercase
String	Concat(String s)	Joins the specified string
String	trim()	Remove the space present before and after the string
String	Substring(int index)	Extract a character from a string object starts from specified index and ends at the end of a string
String	Substring (int start, int end)	Extract a character from a sting starts from specified index from the string
Char	Char Al(int index)	Return character of the specified index from the string
Int	indexOf(char ch)	Return the index of the character specified if not return.
Int	indexOf(char ch, int Start_Index)	Return the index of the character specified by searching from specified index if not retun.
Int	lastindexOf(char ch)	Retuns line index of the character which is occurred at last in the original string.
Int	Length()	Return length of the string
Boolean	Equals(object o)	Compares states of a two strings.
Boolean	equalsignoreCose(string s)	Compares two strings by ignoring its case (lowercase and uppercase)
Boolean	Contains (String str)	Retunes true if specified string is present else if return false.
Boolean	isEmply()	Retunes true if string is empty else tertun false.
Char[]	toCharArray()	Converts the specified string into charcter array
String[]	S(string sfr)	Break the specified string into multiple string only removing passed string and returns string array.
Byte[]	getBytes()	Converts the specified stringto byte value and returns byte array.

Characteristic:

- Instance of string class is immutable in nature. (once the object is created then the state is not modified)
- If we try to manipulate (modify) the state/date then new object is created and reference is given.

Disadvantages:

- Immutability, because for every modification separate object is get created in a memory, it reduces the performance.

  Note –

  To overcome the disadvantage of string class we can go for StringBuffer and StringBuilder.

Java.long.stringBuffer:

- It is a inbuilt class defended in java.long parckage.
- It is a final class.
- It helps to create mutable instance of string.
- stringBuffer does not have scp.
- It inherits java.long.Object class.
- In stringBuffer equals90, hashcode [] methods of java.lang.Object class are not overridden.

  It implements:

  Serializable

  charSequence

Constructors	Description
StringBuffer()	Creates emply String with initial capacity 16
StringBuffer(String str)	Creates String buffer with the specified string

Example 1:

```
Class Demo {
    public static void main (String [] args) {
        StringBuffer sb1 = new StringBuffer("Hello");
        StringBuffer sb2 = new StringBuffer("Hello");
        System.out.println(sb1);
        System.out.println(sb2);
        System.out.println(sb1==sb2); //false here also ref is
        //comparing
        System.out.println(sb1.equals(sb2)); //false here also
        //ref is comparing
    }
}
```

Example 2:

```
Class Demo {
    public static void main (String [] args) {
        StringBuffer sb1,sb2;
        sb1 = new StringBuffer("Hello");
        sb1 = sb2;
        System.out.println(sb1);
        System.out.println(sb2);
        System.out.println(sb1==sb2); //true
        System.out.println(sb1.equals(sb2)); //true
    }
}
```

Example 3:

```
Class Demo {
    public static void main (String [] args) {
        StringBuffer sb1,sb2;
        sb1 = new StringBuffer("Hello");
        sb1 = sb2;
        System.out.println(sb1); //Hello
        System.out.println(sb2); //Hello
```

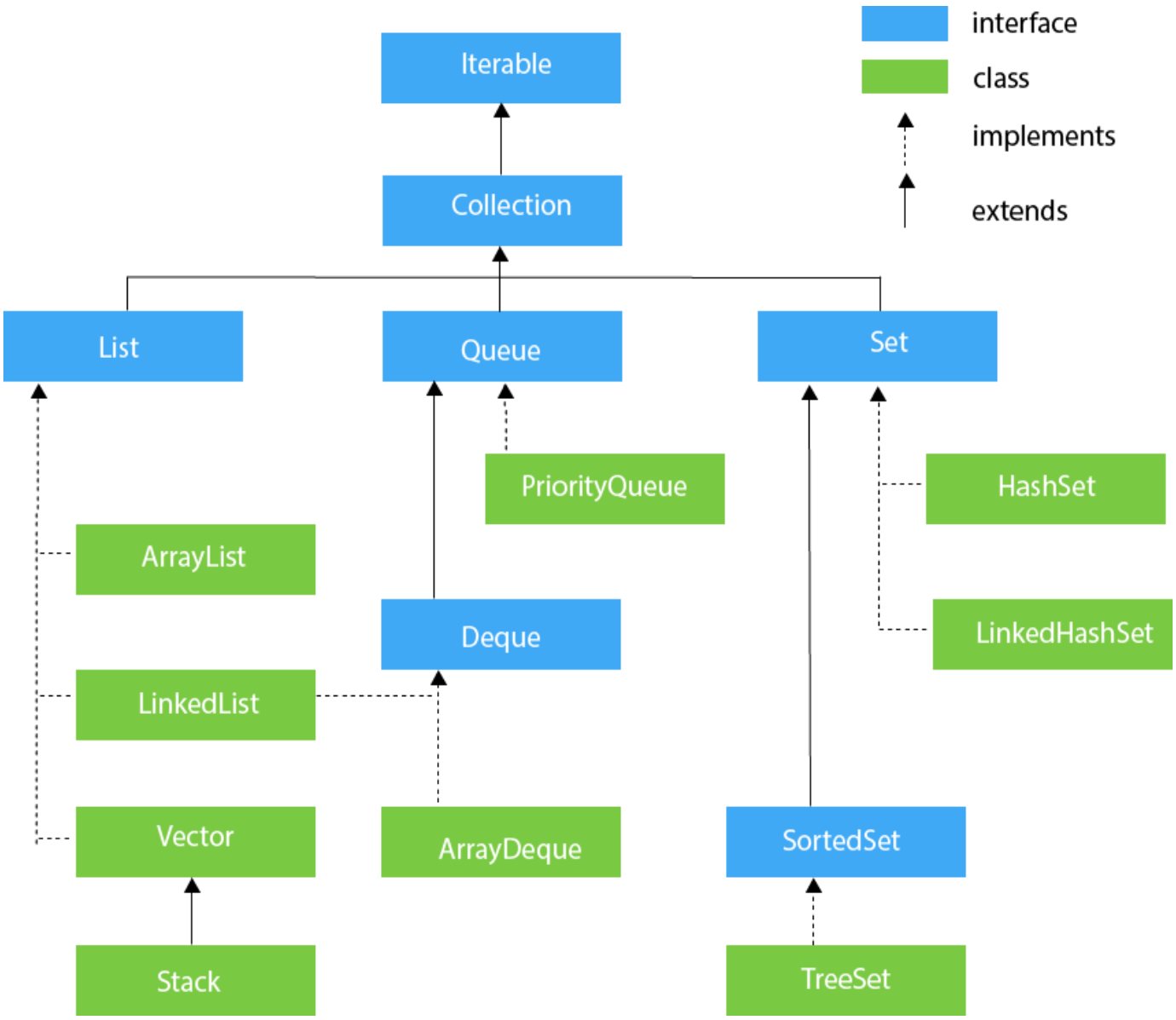
```
                sb1.oppend("world");
                System.out.println(sb1); //Hello
                System.out.println(sb2); //Hello
                System.out.println(sb1==sb2); //true
                System.out.println(sb1.equals(sb2)); //true
        }
    }
```

Disadvantages:

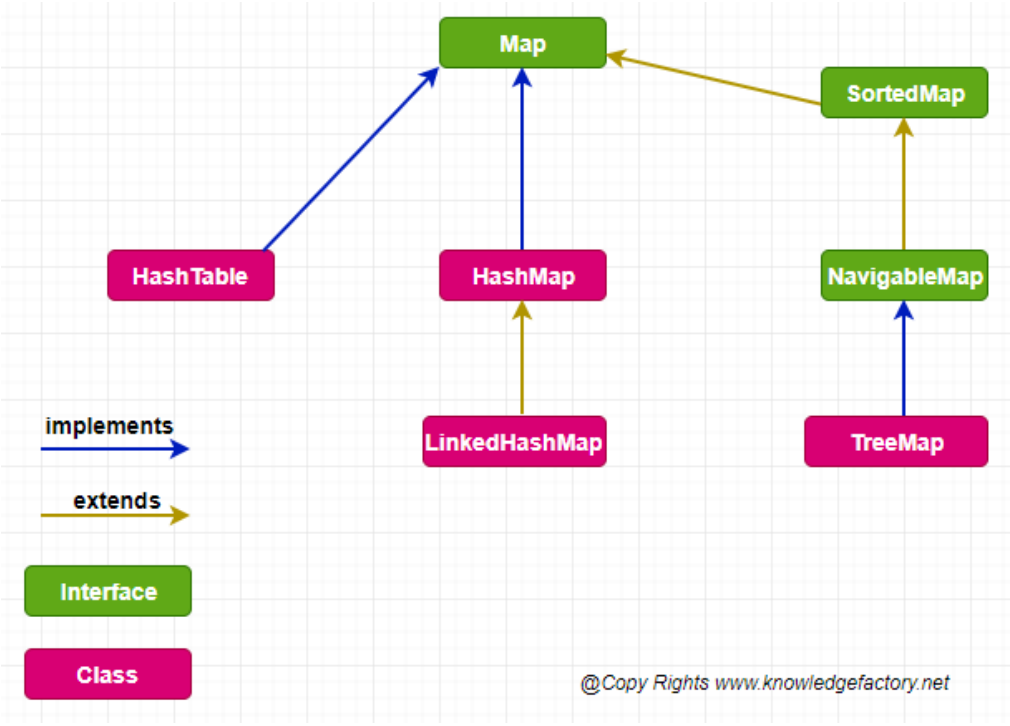
Multiple threads can't execute the StringBuffer object simultaneously because all the methods are synchronized. So, execution time is more. In order to overcome this problem, we will go for StringBuffer.

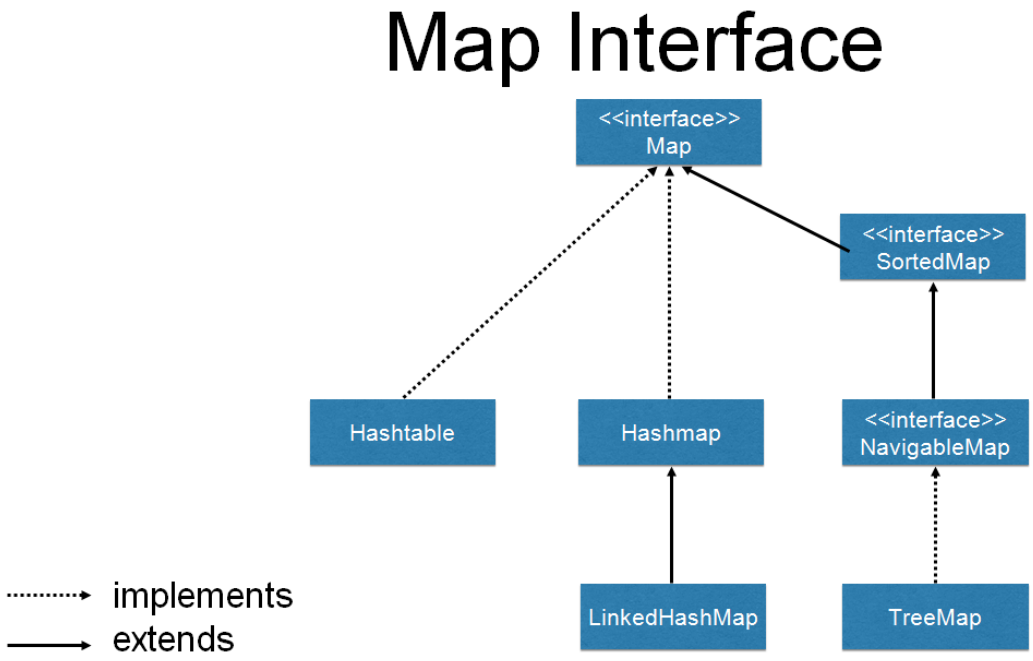
Note –

The characteristics of StringBuffer and StringBuffer are some.



Map Hierarchy





What is collection in java

- A collection represents a single unit of objects, i.e., a group.

What is a framework in java

- It provides readymade architectures.
- It represents a set of classes and interfaces.

What is collection framework?

- The collection framework represents a unified architecture for storing and manipulating a group of objects. It has;
- Interfaces and its implementations. i.e., Classes

- Algorithm (java collections can achieve all the operations that you perform on a data such as searching, sorting insertion manipulation and deletion.

Why do we need collection framework in java

- To store multiple objects or group of objects together we can generally use arrays. But arrays have some limitations.

Advantages of collection over any array:

- Array is fixed in size whereas collection is dynamic in size.
- Array is homogeneous whereas collection is heterogeneous.
- Array does not have many more method to go from corroboration. But collection having many more predefined methods go for it.
- Therererefore to avoid the limitations of the array we can store the group of objects or elements using different data structures such as:
  1. List
  2. Set
  3. Queue
  4. Maps/dictionaries
- Collection framework has many inbuilt classes, interfaces and methods which are present in java.util package, hence we need to import them before use.

Hierarchy of collection framework

Collection framework has 2 important hierarchies:

1. Collection hierarchy
2. Map hierarchy

When we have to go form map

- When we want to store the elements in the form of group of objects only we have go to collection hierarchy.
- If we want to store the element in the form group of entries, we have go for the map hierarchy. Where entry is collection of key and value pair.
  Here, key should be which provide more security to the group of objects and presents of key.

Important Methods of Collection Interface

Purpose	Return Type	Method Signature
To add the element	boolean boolean	1. add(Object) 2. addAll(Collection)
To remove element	boolean boolean boolean void	1. remove(Object) 2. removeAll(Collection) 3. retainAll(Collection) 4. clear()
To search the elements	boolean boolean	1. contains(Object) 2. containsAll(Collection)
To access the elements	iterator	1. iteration() 2. for each loop
Miscellaneous operation	int boolean Object []	1. Size 2. isEmpty() 3. toArray()

Methods of List interface

Purpose	Return Type	Method Signature
To add the element	boolean boolean	1. add(Object) 2. addAll(Collection)

	========= void boolean	============= 3. add(int index, Object) 4. addAll(int index, Collection)
To remove element	boolean boolean boolean void ========= E	1. remove(Object) 2. removeAll(Collection) 3. retainAll(Collection) 4. clear() ================== 5. remove(int index)
To search the elements	boolean boolean	1. contains(Object) 2. containsAll(Collection)
To access the elements	Iterator NA ====== List iterator Object	1. iteratior() 2. for each loop ============== 3. listIterator() 4. get(int index)
Miscellaneous operation	int boolean Object []	1. Size 2. isEmpty() 3. toArray()

Difference between List and Set	
List	Set
List is index based	Set in not index based
List can store duplicate data	Set can store only unique data
List can store multiple null value	Set cab store only single null value
List maintains insertion order	Set doesn't maintains insertion order

Difference between for and foreach loop	
For loop	For each loop
Can iterate forward as well as backward direction	Can iterate only in forward direction
Partial iteration is possible	Partial iteration is not possible
Need to know the condition to use	Can be used even without condition
Present since begin of java	Present since JDK 1.5

Difference between ArrayList and LinkedList	
ArrayList	LinkedList
Store the data in the form of array	Stores the data in the form of nodes
Data structure: growable/ resizable array	Data structure: Doubly LinkedList
3 overloaded constructors	2 overloaded constructors
Initial & incremental capacity is applicable	Initial & incremental capacity is not applicable
Implements marker interface: cloneable, serializable, RandomAccess	Implements marker interface: cloneable serializable
Has shift operation	Has no shift operation
Memory is continuous	Memory is not continuous

Difference between ArrayList & Vector	
ArrayList	Vector
ArrayList is multi-Threaded.	Vector is single threaded.
Present since JDK 1.2	Present since JDK 1.0
3 Overloaded Constrictors	4 overloaded constructors
Incremental capacity = ((current capacity * 3)/2) +1	Increments capacity = current capacity *2
Performance wise faster	Performance wise slower

Map interface

- Map is an interface which is a separate pillar or vertical of the collection framework and defined in java.util package.
- It is present since JDK 1.2.
- It doesn't inherit Collection interface.
- Map is a Collection of entries. i.e.., map stores multiple entries. Entry is a data in an associated form. i.e., keyand value pair.
- In other word Map is data structure, which heps the programmers to store the data in the form of key and values pairs. Where key and value combingly known as entry. Where every value is associated with a unique key.
- Key must always be unique, but value can beduplicated or null
- One key can be associated with only one value.
- Map helps us to access the values easily with the help of it's associated key

Note:

One key value pair is called as an entry or Mapping.
We can also make generic map and non-generic nap.

Generic Map:

Map<K, V>
Map<Integer, String>

5 = "Rohan"	7 = " Rahul "	3= "Dev"	9 = "ubham"

Map interface methods		
Return type	Method Signature	purpose
V	put (K key, V value)	1. add an entry to the map (key-value pair) 2. replace the olk value with a new value of an existing entry in the map
void	putAll(Map<? extends K,? extends V> m)	it will copy all the entries from the given map into the current map.
boolean	containsKey(Object key)	if the key is presint returns true else returns false
boolean	containsValue(Object value)	if the value is present, it returns true, else it returns false.
V	remove (Object key)	if the key is present the entry is removed from the map and the value is returned. If the key is not present nothing is removed and null is returned.
void	clear()	it removes all the entries in the map.
V	get(Object key)	it is used to access the value associated with a particular key. if the key is presentit returns the value. If the key is not present it returns null.
Collection<V>	values()	it returns a collection of values present in the map. return type Collection<v>
Set<K>	keySet()	it returns a set of keys present in the map, return type Set<k>

Set<Map.Entry<K, V>>	entrySet()	it returns a set of al the entries present in the map. return type Set<Map.Entry <k >>
int	size()	Returns the number of key-value mappings in this map.
boolean	isEmpty()	Returns true if thi: map contains no key-value mappings.

iterator() method

1. iterator() is method which belongs to Iterable nterface.
2. iterator() method creates an Iterator interface ype object and returns the reference.
3. iterator() method returns the reference in Iteraor (interface) type

Iterator Interface

- Iterator is an interface Set, List and Queue present in java.util package, it is used iterate any collection like set, list and queue.
- Iterator is not index based.
- Iterator can iterate only in forward direction. Iterator is also considered as Cursor.

Iterator Methods:

Return type	Method name	Purpose
boolean	hasNext()	it checks whether thire is an element to be accessed from the collection, if present returns true, else it returns false.
E	next()	it returns the object which is present after the cursor and move the cursor to the next position.
default void	remove()	Remove element

Work of next() method

1. Gives the element which is present at the curior position
2. Move the cursor to the next position

   By default, the cursor presents at the 1st position.

NoSuchElementException:

- In the Iterator, when we try to access an elements using next() method and if there is no element present, we get NoSuchElementException.

remove() method

- It remove the element where the cursor is present

IllegalStateException:

- Remove method can only used after the lext() method, if we try to use it any other place we will get the IllegalStateException.

Disadvantage of Iterator

1. We can iterate only once.
2. We cannot access the elements in reverse order.

listIterator() method

1. listIterator() is method which belongs to List Interface.
2. listIterator() method creates an ListIterator interface type object and returns the reference.
3. listIterator() method returns the reference in Lstiterator (interface) type.
4. In the List interface 2 overloaded methods are there for listIterator.

1. listIterator(): Creates a ListIterator interface type object and cursor will be pointing to first element.
2. listIterator(int cursor_position): Creates a Listiterator interface type object and cursor will be pointing to the position provided.

ListIterator Interface:

- ➤ It is an sub interface of Iterator interface.
- ➤it is defined in java.util package.
- ➤ ListIterator can be used to iterate only List but rot Set and Queue.
- ➤ It can iterate both in forward as well as backward direction.

Methods of ListIterator interface		
Return Type	Method Signature	Purpose
boolean	hasNext()	to check whether next element to be accessed is present or not.
E	next()	it gives the element, and hoves the cursorforward.
boolean	hasPrevious()	to check whether previous element to be accessed is present or not.
E	previous()	it gives the previous element pointed by the cursor, and moves the cursor backward.
void	remove(Object)	it removes the current object pointed, from the collection, which is under iteration.
void	add(Object)	it is used to add a new object into the collection.

Difference between Hashset or linkdHashset

Hashset don't mention the insurtation order, linkdhashset mension the insuration order.

Difference between hashmap or LinkedHashet

Hash

Hashmap having 4 overloading constructer, linkedhashet having 5 overloading

Difference between HashMap and Hashtable	
HashMap	Hashtable
It is Multithreaded	It is single threaded
Present since JDK 1.2	Present since JDK 1.0
Can strore single null key	Cannot store even single null key
Initial capacity 16	Initial capacity 11
It is faster	It is slower

Collections cass

1. It is a predefined class in java for doing the operation on Collection framework.
2. It is defined on java.util package.
3. Here so many predefined methods are there for different operation such as sort, reverse, binarysearch, replace etc. and all are static in nature.
4. As all the static in nature to call all the method we have use the Collections class name and import the java.util.Collections class.
5. One important methods we are usually using that is sort() method to sort the List.
6. There are two overloaded methods for sort() method.

TreeSet

- TreeSet is one of the implementation class of Set interface.
- Present since JDK 1.2. And defined in java.util Package
- TreeSet is mainly used for uniqueness and sorting. i.e... TreeSet doesn't store duplicate data and also implements default natural sorting order. Tre Set cannot store even single null element.

Default natural sorting order:
Ascending order...

- Number 0-9
- Alphabet A-Z & a-z

➤ TreeSet is not a hash basedlection, the data structure used is Balanced tree.

➤ TreeSet is Homogeneous i.e... TreeSet can store only one type of data.

➤ TreeSet either uses Comparable or Comparator interface.

Note: Comparable is used for default natural sorting.

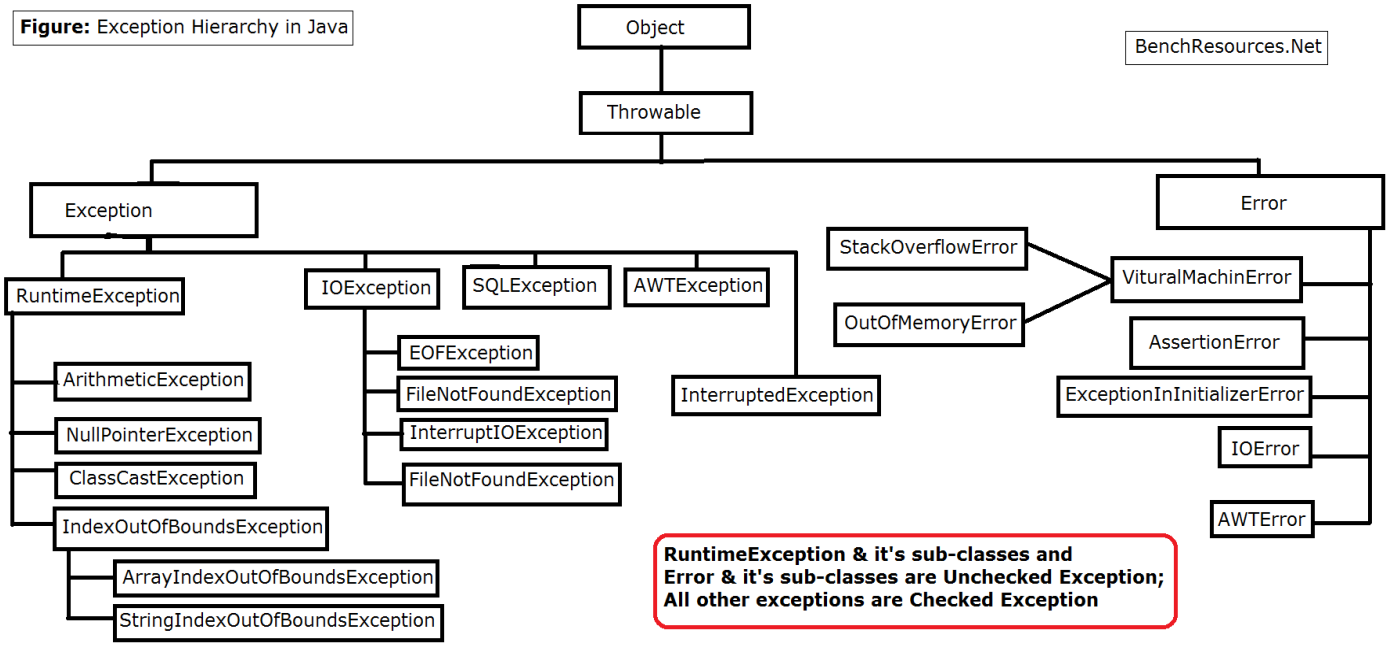
Comparator is used for Custom sorting

- The element to be added in the TreeSet must be Comparable type, else we get ClassCastException.
- All elements in the TreeSet should be same type (Homogeneous), if not we will get also ClassCastException.
- No indexing is there, therefore we can't add, remove element using index.
- It will also have all the methods of Collection interface.

TreeMap

- It is one of the implementa lass of Map interface.

- Present since JDK 1.2. And declared in java.util packag
- TreeMap is used mainly for sorting data based on key.
- TreeMap implements default natural sorting order on thekey using Comparable interface.
- The key in TreeMap must be Comparable type. If not then we will get ClassCastException.
- For custom sorting we use Comparator interface.
- It cannot store even a single null key.
- TreeMap is homogeneous.
- when we try to add null, we will get NullPointerException



Exception

- The Exception is an unexpected problem which occurs during the execution of the program (i. E nothing but RunTime). When an except on occurs, the program execution stops unexpectedly(this is known abrupt Stop of the program).
  Note:

1. Every Exceptions in java is one one of the child of Throwable class, that's why we can say that every exceptions in java 'Throwable Type.
2. Every Exceptions occur because of one Statemen
3. A statement will throw an exception during unexpected problem (ie nothing but abnormal situation).
   Example

```
class Demo {
        public static void main(String[] args) {
        int a=6;
        int b=0;
        System.out.println(5+7);
        System.out.println(a/b); //bcoz of one statement //Every Eception is java is one class apo all are Throwable Type
        }
}
```

Lists of Important Exceptions and Statements	
Statement:	Exception:
1. a/b (if b=0)	1. ArithmeticException
2. Obj reference_var.object_member	2. NullPointerException
3. (ClassName) Obj_reference_var	3. ClassCastException
4. Array ref_var[index]	4. ArrayIndexOutOfBoundsException
5. string ref_var.charAt(index)	5. StringindexOutOfBoundsException

Types Of Exception

There are 2 types of Exceptions are there.

1. Checked Exception
2. Unchecked Exception

Checked Exception:

- The compiler aware exception is known as Checked Exception. That means Compiler knows which statement is responsible for the Exception. If this is the scenario then the compiler will the force the programmer to either handle or declare the exception. If it is not done, we will get an unreported compile time error.
  Ex:
  - FileNotFoundException
  - FileOutputStream f = new FileOutputStream("path");
  - FileOutputStream f = new FileOutputStream(" d://Part3/abc.txt");

FileNotFoundException

- FileNotFoundException is defined in the java io package.
- FileNotFoundException is a checked Exception.
- We get FileNotFoundException when we try to create a file but the given path is wrong or no permission or hard disk has no sufficient memory to store the file.
- new FileOutputStream("Path/name") -> this line is responsible for FileNotfoundException.

UnChecked Exception

- The compiler unaware exception is known as unchecked Exception. i.e, the compiler doesn't know the statements which are responsible for abnormal situations (Exception).
  Hence, the compiler will not force the programmer ether to handle or declare the exception.
  Ex:

  ArithmeticException

C = a/b; (if b = 0)

FileNotFoundException

- FileNotFoundException is defined in the java.io package.
- FileNotFoundException is a checked Exception.
- We get FileNotFoundException when we try to create a file but the given path is wrong or no permission or hard disk has no sufficient memory to store the file.
- new FileOutputStream("Path/name") -> this line is responsible for FileNotfoundException.
  Note:
  1. In Throwable hierarchy Error class and its subclasses and RuntimeException class and its subclasses are all known as Unchecked Exception.
  2. All the subclasses of the Exception class except RuntimeException class, are considered as checked exception.
  3. Throwable and Exception class are partially checked and partially unchecked classes.
  4. RuntimeException and Error classes are Fully unchecked Exception classes.

Throwable class

- Throwable class is defined in the java lang package.
- It is the supermost most parent class of the Exception Hierarchy (means parent of the all the Exception).
- As Object class is the parent of all the classes in java, so for the throwable class also object is parent class.
- In the throwable class toString() method of Object class is overridden, that why all the sub class of Throwable have the overridden toString() method. That's why when any of the exception occurs we are getting he fully qualified name of the exception.
  Ex:
  java.lang.ArithmeticException etc.

Important Methods of Throwable class

1. String getMessage(): used to return the reason of the exception
2. void printStackTrace(): used to give Fully Qualified name of the exception and reason of the exception along with that flow of the exception.
   Both all the above methods are the non-static method.

Exception Handling:

- Exception handling is a mechanism used in java that is used to continue the normal flow of execution when the exception occured during runtime.

How to Handle the Exception

- In java, we can handle exception by using the try {} and catch() {} block.

Syntax to use the try and catch block:

```
try {
        //statements
}
catch(declare one variable of Throwable type) {
        //statements
}
```

If exception will occur what will happens?

- When an exception occurs:
  1. Execution of the program will be stopped.
  2. A throwable type object will be created and reference of the created object will be thrown.

try{ } block:

- Create try block and the statements which are responsible for exception should be written inside the try block.
- When an exception will be occurs

  1. Execution of try block is stopped.
  2. A throwable type object is created.
  3. The reference of throwable type object created is passed I to the catch block.

```
try {
        stmt 1;
        stmt 2; // Exception occurs

        Ox1
        Throwable type object created
        feel 3:
        stmt 4:
        //stmt 3 and stmt 4 will not execute
}
catch (variable) {

}
// Reference of the Throwable type object is thrown to the catch block.
```

The catch block is used to catch the Throwable type reference thrown by the try block.

1. If it catches, we say the exception is handled. Statements inside the catch block are get executed and the normal flow of the program will continue.
2. If it doesn't catch, we can say the exception is not handled. Statements written inside the catch block are not executed and the program i terminated.

Q. When do we say exception is handled?

We say exception is handled only if Exception is caught by catch block

Exception occurs no caught

Case 1: Exception occurs not caught:

```
try {
        sopln (100); //ArithmeticException scours
}
```

ArithmeticException@6708

```
catch (NullPointerException e) {
        //Not executed
}
//Not executed
```

Hare the exception is not caught by the catch block, because it only catch the nullPoinerException not ArithmeticException.

Exception occurs and caught:

Case 2: Exception occurs and caught:

```
try {
        sopin( 10/0); // ArithmeticException occurs
        Stmt 1; //Not executed
}
catch (ArithmeticException e) {
        stmt 2://Execute
```

```
}
Stmt 3; // Execute
```

Here it is caught by the catch block

Try with multiple catch Block

- Try block can be associated with more than one catch blocks.

Syntax:

```
try {

}
catch (.......)

}
catch (......)

}
etc
```

Note:

The exception type object is thrown from top to botten order.

Workflow of the try with multiple catch

Workflow:

```
try {

}
catch () {

}
catch () {

}
catch () {

}
```

If the exception is caught by the catch block then the try block does not throw the exception to the below catch blocks, then execution of the rest of the catch block will be skipped.

Rule:

The order of catch blocks should be maintained such that, child type should be on the top and parent type at the bottom.

Name – Jeevan Kumar Sahu

Batch code – JGU-JFFCD-A3

Q1. Difference between print and println statement example.

Print –

- The 'print' statement is used to display text without inserting a newline character at the end. If you use multiple 'print' statement one after another, their output will be concatenated on the same line.

  Ex –

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

  Output – Hello, world

Println –

- The 'println' statement is used to display text and automatically appends a newline character('\n') at the end. This means that each 'println' statement starts on a new line.

  Ex –

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

  Output – Hello,

  World

Q2. Difference between rules and conventions.

Rule –

- Rule in java are strict guidelines enforced by the language itself. Breaking a rule result in a compilation or runtime error. These rules are mandatory and must followed for the code to work correctly.

  Ex –

  Every statement must end with semicolon (;).

```
public class RuleExample
{
        public static void main (String [] args)
        {
                System.out.print(“This is a rule”);
                //correct, end with a semicolon

                System.out.print(“Breaking the rule”)
                //Incorrect, missing semicolon
        }
}
```

Conventions –

- Conventions in java are recommendations or best practices followed by developers for code readability and maintainability. Conventions are not enforced by the language but are essential for writing clean and understandable code.

  Ex –

  Using camelCase variable name

```
public class PrintExample
{
        public static void main (String [] args)
        {
                int itemCount=5;
                //correct, camelCase variable name
                int item_Count=5;
                //Incorrect, not following cameCase convention
        }
}
```

*Note –

In the above ex – the variable ‘itemCount’ follows the comelCase convention while ‘item count’ does not.

Assignment:

1. Write a Java program to print 9999.
2. Write a Java program to print 11111.
3. Write a Java program to print addition 11111+99999.
4. Write a Java program to print “Hello.”
5. Write a Java program to print hello and good morning.
6. Write a Java program to print your name, aggregate, year of pass out year.

Do all the activities by using the println statement.

1. Write a Java program to print "hello world".
2. Write a Java program to print without a main method.
3. Write a Java program to print your name.
4. Write a Java program to print hello and good morning with a println statement.
5. Write a Java program to print your name, aggregate, and year of passing out in different lines.

15, hy, a, *,
1 2 3 4 5 ……..
10 11 12 13 ……..
5 10 15 20 ……..
1 3 5 7 ……..
A B C D E ……..
100 99 98 97 ……..

1. WAJP creates a double-type variable store the character data inside it and print the variable.
2. WAJP Create an integer variable store the character inside it and print it.
3. WAJP Create a double type of variable store the integer inside it and print it.
4. WAJP Create an integer variable store floating point data inside it and print it.
5. WAJP create a character type variable store integer inside it and print it.
6. WAJP create a character type variable store the double inside it and print it.
7. WAJP create a Boolean type of variable store the integer value inside it and print it.
8. WAJP print the statement use datatypes, variable, keyword
   Name – Qspider
   Phone no – 7870794231
   Place – gurgram
   Class – java
   Room – 4
   Trainer name – Dev
   Staff member – 4
   Trainers are available – 3

9. WAJP print the statement use datatypes, variable, keyword

   Class name – employee details
   Employee name – Jeewan
   Employee id – 101
   Phone number – 123456789
   Email id – imjeewankumar@gmail.com
   Address – 1c, sec-17, gurugram
   Department – testing
   Salary – 6 lpa
   Attendance -true

10. WAJP print the statement use datatypes, variable, keyword

    Class number – personal details
    Name – jeevan
    Email – hiii@gmail.com

Age – 30
Gender – male
Place of birth – Yamuna
Designation – youtuber
Hobby – social worker

11. WAJP to store and print your name, age, date of birth and contact number.
12. WAJP to convert a character into a number.
13. WAJP to store int value into the byte, short, long, float, double, char type variable.
14. WAJP to convert the double value (56.09) into int type and print it.
15. WAJP to store char type data('A') inside the short and int type variable and print it.
16. WAJP to store Boolean type data(true) inside the int type variable and print it.
17. WAJP
18.

Find of the sum of three number
Find of the multiplication
Find of average of 5 number
Print the sum of 2 integer number.
Find the number + or not +
Find the largest amount two number
Find the smallest amount two number
Find out the number is zero or not
Find out the number is nonzero or not
Find the largest 3
Find the smallest number 3 number with conditional operator.

Method

1. What is parametrized method and when we have to fort it write an example of parameterized method.
2. What is access modifier and types of modifiers and explain private access modifier.
3. What is modifier and example of modifier and explain static modifier with example.
4. What is return type and how many types and how many types are there and explain void.
5. What is method declaration and method signature defining it.
6. What is return statement and syntax.
7. What is actual argument?
8. What is caller?
9. Why main method is static and void?
10. Write a prog to show main method overloading.
11. Create a method which accept int and float type argument and return string.
12. Create a add method which returns sum of 2 numbers to the caller.
13. Print your name infinite time without using looping statement.
14. Why main method is public?
15. What is method definition and method implementation?
16. Create a method which accepting character and string type arguments and return nothing.
17. Create a method which is accepting 2 numbers and print the product of 2 number.
18. What is no argument method? Write a java program to call no argument method, call that from caller?
19. What is modifier and explain protected access modifier.
20. What is formal argument?
21. What is called method?
22. What is access modifier and types explain final modifier?
23. What are return types and explain the rules of return statement and primitive type return statement with example.

24. Rules to call the parameterized method with an example.
25. What is method overloading and write one method overloading example.
26. What is method and characteristics syntax to create a method.

Part -II

1. What is object. Define, syntax
2. What is class
3. What is constructer. types of constructers. What is no argument and parametrize construct.
4. What are new key words
5. Difference between constructer and method
6. What is encapsulation, full steps private, geter and seter.
7. How to achieve encapsulation
8. What is data hiding
9. Stape to achieve data hiding
10. What is private
11. What is getter and setter method
12. Advantage of data hiding
13. Real time example of encapsulation
14. What is java bean class
15. What is relationships
16. Types of relationship
17. What is has a relationship and type.
18. What is composition or aggregations
19. What is a relationship how to achieve is-a relation
20. What is inheritance
21. How to achieve inheritance
22. Difference between extended and implement
23. Type of inheritance
24. What is multiple inheritance? Why it is not possible in java
25. What is Dimond ring problem.
26. How to multiple inheritance possible by the help of inheritance
27. What is non primitive type casting.
28. Explain advance disadvantage
29. The time of inheritance. what is the component are not inheritance parapet to child and why.
30. Can you make constructer privet
31. Advance or disadvantage making the construction
32. What is singleton class
33. What is polymorphism and its types.
34. What is compile time polymorphism and how to achieve it
35. What is binding
36. What is method overloading
37. What is constructer overloading
38. What is variable shadowing.
39. What is method shadowing
40. What is run time polymorphism how to achieve it.
41. What is method overriding.
42. Difference between method overloading and method overriding
43. What is abstraction? how to achieve it
44. What is abstract class when to declare the abstract class
45. What is abstract method when to make the abstract method.
46. What is the component are allowed in abstract class.
47. Characteristics of the abstract class.
48. What are interface characters of interface and what is the component allow in the interface.
49. Types of interfaces
50. Can you make abstract class static

51. Can you make abstract class private
52. Can you make abstract class final.
53. Can you make main method final.
54. Can you override the static method.
55. Is final method is inherited to parapets or child.
56. Difference between is and supper keywords.
57. Difference between this call and super call statements.
58. What are the rules for this key statements
59. What is construction Chaing how to achieve it
60. Difference between static and non-static.
61. What is final modifier
62. What is object class.

Part I

1. What is java explain features of java
2. What is jdk jre jvm
3. What is jit compiler
4. Difference between compiler and interpreter
5. Why java is platform independent or architecture natural
6. What is WORA architecture.
7. Difference between print and println statements.
8. What is programming language and types.
9. What is keywords
10. What is indemnifier and rules of identifiers
11. Difference between rule and conversion
12. What is data variable and data types and those types.
13. What is primitive type casting and types.
14. What is local variable and its characteristic
15. What is operator and its types.
16. Differ between if, if else and switch
17. Difference be while and do while
18. Syntax and workflow of for loop
19. What is break statements
20. What is method and different type of method difference and parametrise
21. What is main method.
22. Why main method is public static void
23. Can be overload the main method
24. Can interchange the position of access modifier and modifier or the main method.

Part – III

1. What is string literal.
2. How many ways creating a string in java.
3. What is string class. Explain about string class constructer and method.
4. Why sting is immutable
5. Sting is immutable is advantage or disadvantages of sting.
6. Different between string buffer, string builder, sting.
7. if it java is fully
   java is not fully object oriented language because some of the object oriented concept not supported in java as follow. 1. Multiple inheritance by the help of class is not supported in java, operate which is concept in . static member without creating a object. Primitive data type are allow in directly java.

Programs

Prime number, perfect number, palindrome number Armstrong number and Strongs number
Swipe two number without 3rd variable and without 3rd variable
Leap year
Chek the given number is odd without modules operator.
Farinose series'

Wajp to achieve the escalation
Wajp to achieve different lever or inheritance single level multiple by the help class and interface
Wajp to achieve the constructer over loading
Wajp achieve construction Chaing
Wajp achieve method mover loading
Variable showing method overriding, method shadowing
Achieve upcasting and down casting instance operator
Wajp to achieve the abstraction abstract class
Wajp to by the help of interface
Wajp to achieve multiple inheritance by the help of interface

Wajp Create a static variable inside the class and access it with the help of class name object reference and directly
Wajp create two class access one class static variable from the another class. By the help of class name, object reference.
Create a static method inside a class try to access it different way within same class as well as different class.
Wajp to design a add method which is accepting two int value and returning sum or both the value access this method from the different class and print the summation of two number.

Create the array of int long float double character Boolean string do some following operation print the reference print each element manually print all the elements bye using loop while do for.

WAJP to create student class which having two variable name and phone no. don't take any user define create five objects and print all the reference and print name and phone of individually object by using respective references

Wajp to achive single level inherent

Wajp to achive multi-level inheritance