Java Notes - TheTestingAcademy (Pramod Sir)

✅ Why Learn Java?

- Java is a general-purpose, object-oriented programming language that was designed by James Gosling at Sun Microsystems in 1995
- Compilation of the Java applications results in the bytecode that can be run on any platform using the Java Virtual Machine. Because of this, Java is also known as a WORA (Write Once, Run Anywhere)
- Java Version - https://en.wikipedia.org/wiki/Java_version_history
- High Level, Platform independent & Architecture Neutral
- Secure, Robus, Multi Thread, Distributed and OOPs language.

⚠️ Why Java is not 100% Object-Oriented?

- Primitive data types.
- Use of Static.
- Wrapper class

⚠️ How is Java Secure?

- JVM which protects from unauthorized or illegal access to system resources.
- Oops and inner classes

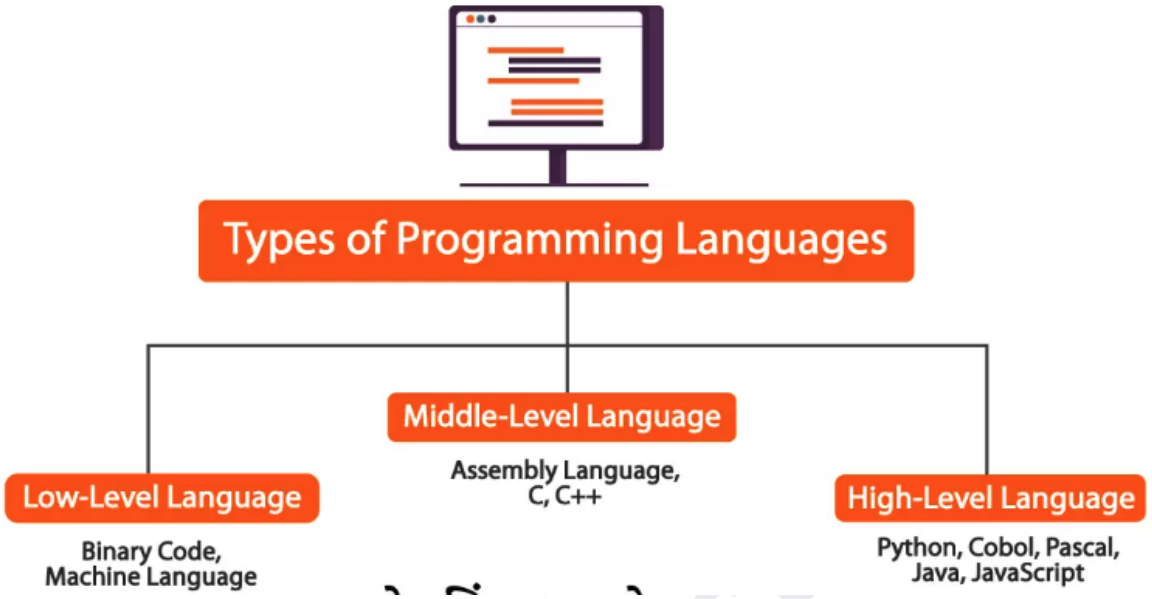
Ref - https://www.scaler.com/topics/why-java-is-not-100-object-oriented/

Important Points

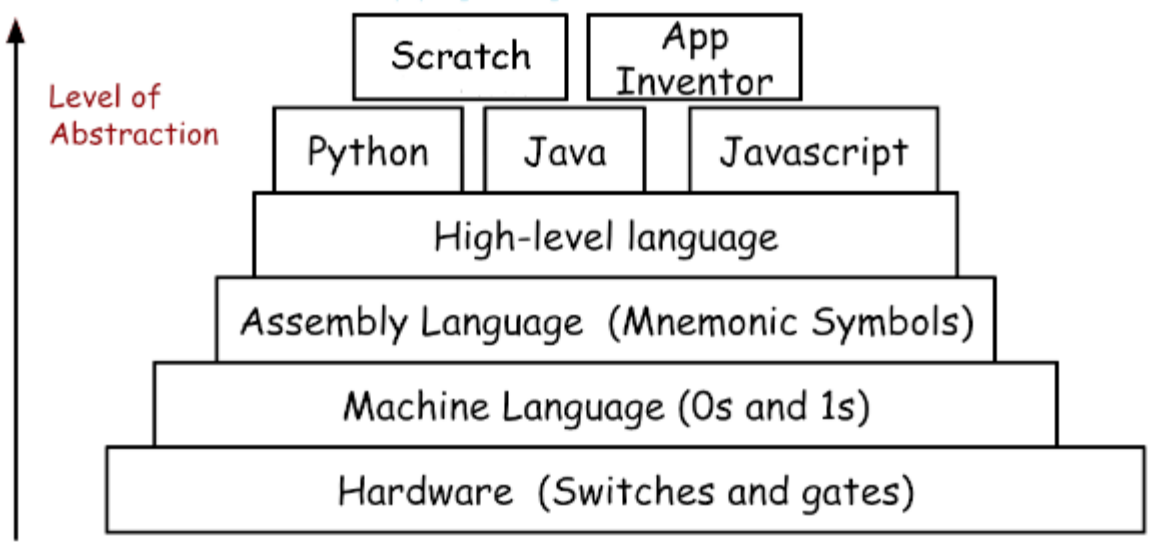
- Since Java applications can run on any kind of CPU so it is architecture – neutral.
- Java program can be executed on any kind of machine containing any CPU or any
- operating system.

TheTestingAcademy

- Java is robust because of following: Strong memory management(Garbage Collector) No Pointers Exception handling Type checking mechanism Platform Independent.
- Using RMI and EJB we can create distributed applications in java



Check the Languages, with the Level of Abstraction



Major Features of Java Programming Language

- Simple.
- Object-Oriented.
- Platform Independent.
- Portable.
- Robust.
- Secure.
- Interpreted.
- Multi-Threaded

⚠️ **What is Source Code?**

Human understandable code written using High Level Programming language is called as Source Code. (NameOfFile.java)

⚠️ **What is Byte Code?**
JVM understandable code generated by Java Compiler is called as Byte Code. Byte code is also called as Magic Value.

⚠️ **Why C and C++ are Platform Dependent?**
When you compile C or C++ program on one Operating System then compiler generates that Operating System understandable native code.Native code generated on one OS will not run on other OS directly. Window -> exe, Mac -> dmg, Linux, deb, pkg

⚠️ **What is Java Compiler?**

Java Compiler is a program developed in C or C++ programming language with the name "javac". It will check syntactical or grammatical errors of the programs. It converts source code to byte code.

⚠️ **What is Java Interpreter?**
Java Interpreter is a program developed in C or C++ programming language with the name "Java". It will convert byte code to native code line by line. It will execute that native code.

⚠️ **What is JIT Compiler?**
JIT (Just-In-Time) compiler is a component of the Java Runtime Environment. JIT Compiler compiles or translates or converts the neccessary part of the bytecode into machine code instead of converting line by line. Because of this, performance of Java program has improved.

TheTestingAcademy

Run your First Java Program

1. Install Java and Set path to the Home in Windows / Mac
2. Install IDE (IntelliJ) and you can avoid the first step.
3. Create new command line project and Add the code and run the program.

How to Set JAVA_HOME path in MAC - Click here

1. Download the JDK latest -

```
echo export "JAVA_HOME=\$(/usr/libexec/java_home)" >> ~/.bash_profile
```

2. If you're using zsh (which probably means you're running macOS Catalina or newer), then it should instead be:

```
echo export "JAVA_HOME=\$(/usr/libexec/java_home)" >> ~/.zshrc
```

3. In either case, restart your shell.

How to Set JAVA_HOME path in Widnows

Set the JAVA_HOME Variable

To set the JRE_HOME or JAVA_HOME variable:

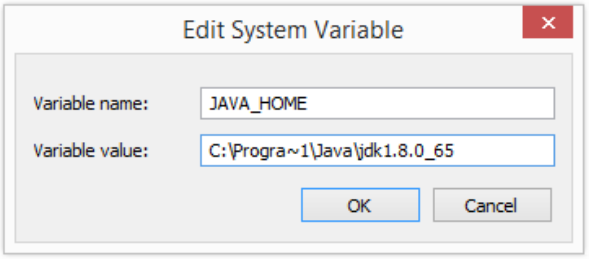
1. Locate your Java installation directory
   If you didn't change the path during installation, it'll be something like

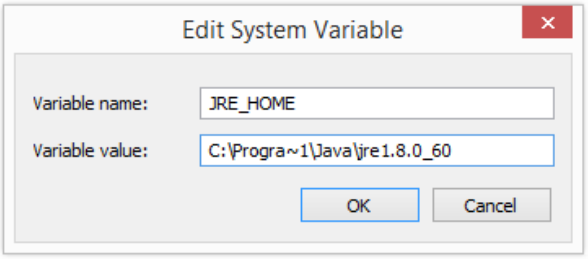
```
C:\Program Files\Java\jdk1.8.0_65
```

You can also type `where java` at the command prompt.

2. Do one of the following:
   Windows 7 – Right click **My Computer** and select **Properties** > **Advanced**
   Windows 8 – Go to **Control Panel** > **System** > **Advanced System Settings**
   Windows 10 – Search for **Environment Variables** then select **Edit the system environment variables**
3. Click the **Environment Variables** button.
4. Under **System Variables**, click **New**.
5. In the **Variable Name** field, enter either:
   - JAVA_HOME if you installed the JDK (Java Development Kit)
     or
   - JRE_HOME if you installed the JRE (Java Runtime Environment)

6. In the **Variable Value** field, enter your JDK or JRE installation path .
   If the path contains spaces, use the shortened path name. For example,
   `C:\Progra~1\Java\jdk1.8.0_65`





   Note for Windows users on 64-bit systems
   Progra~1 = 'Program Files'
   Progra~2 = 'Program Files(x86)'
7. Click **OK** and **Apply Changes** as prompted

⚠️ What is JDK (Java Development Kit) / SDK (Software Development Kit)?

It is a set of various utility programs which are required for developing and executing the java programs.
It is Platform dependent. Various JDKs are provided for various Operating Systems.
Following are various utility programs provided under JDK:
1) Java Development Tools
   i. javac
   ii. java
   iii. javap
   iv. Jar
etc
2) Source Files
3) JRE
etc

⚠️ What is JRE?

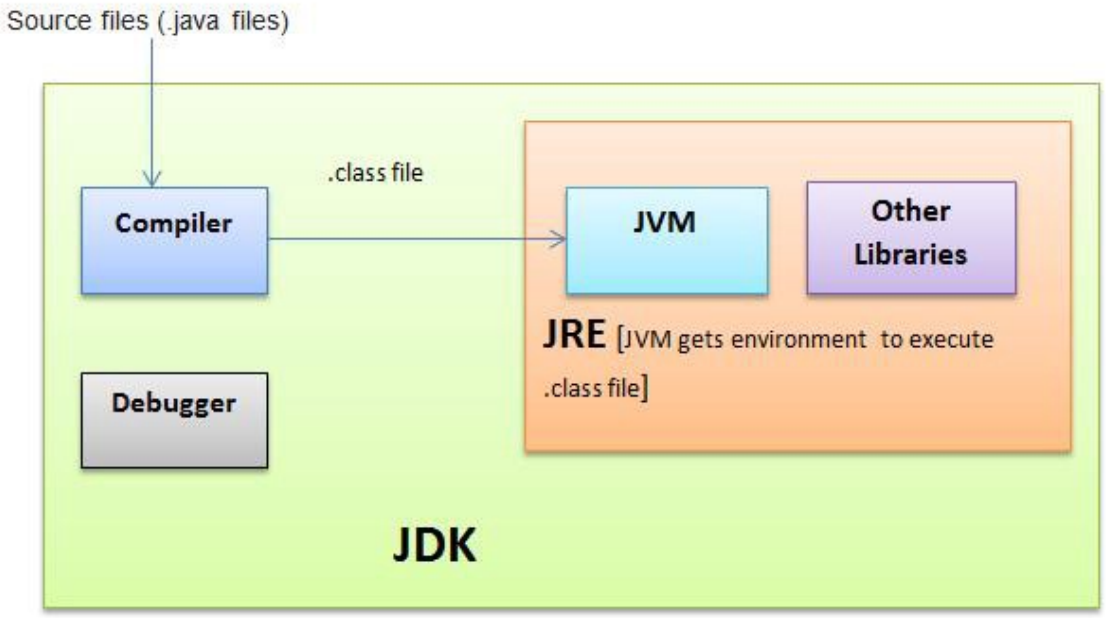
Ans: JRE stands for Java Runtime Environment. It is an implementation of JVM. It contains class
libraries, Interpreter, JIT Compiler etc. Only JRE is enough to run the Java program

⚠️ What is JVM?

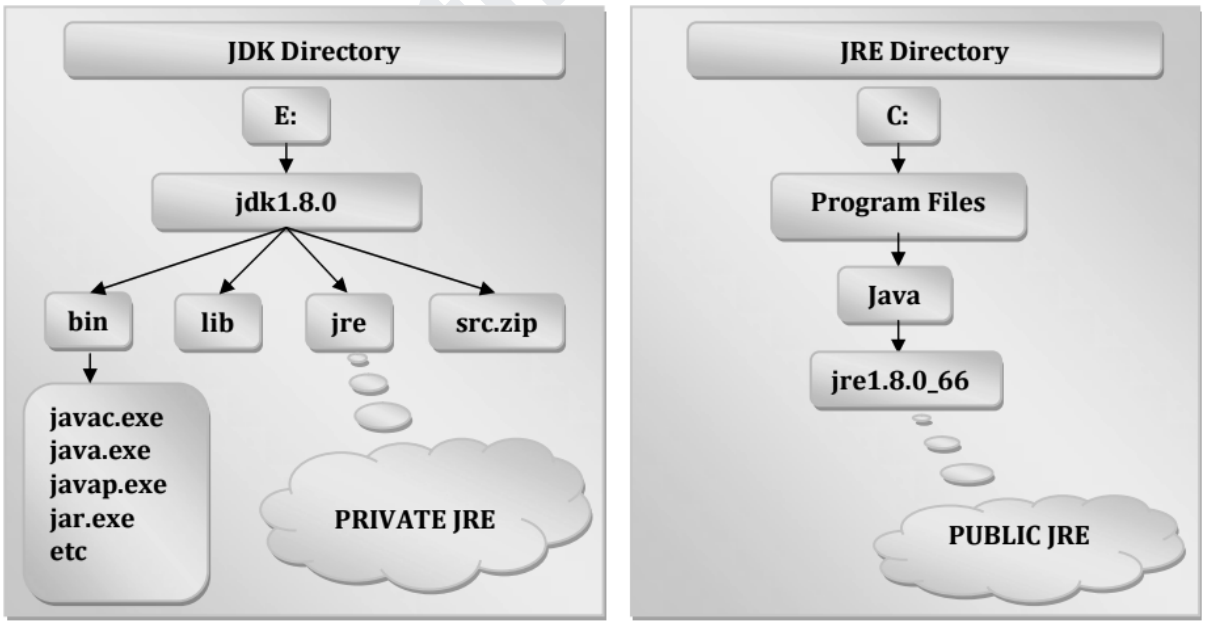
Ans: JVM stands for Java Virtual Machine. It is a specification provided by SUN Microsystem whose
implementation provides an environment to run our Java applications. JVM becomes an instance of
JRE at run time.

TheTestingAcade

Sun’s implementations of the Java Virtual Machine (JVM) is itself called as JRE. Sun’s JRE is availabe
as a part of JDK and also as a separate application.
Many vendors has implemented JVM. Some of them are SUN JRE, IBM JRE, Oracle JRE etc



JDK Vs JRE



Java Editions

1. Enterprise Edition - Servlets, JSP, JDBC etc.
2. Micro Edition - micro devices like Mobiles,Setup Box etc
3. Standard Edition - Used to develop standalone applications using applet and swing.

Edition	Description
Java SE (Standard Edition)	**The main edition of Java that is used for desktop and server applications. It includes the Java language, JVM, class libraries, and tools for developing, testing, and deploying Java applications.**
Java EE (Enterprise Edition)	**A set of specifications and APIs for developing enterprise applications, including web and mobile applications, distributed systems, and cloud computing. It includes components like servlets, JSPs, EJBs, JMS, JPA, and more.**
Java ME (Micro Edition)	**A platform for developing applications for mobile and embedded devices, like phones, PDAs, and set-top boxes. It includes a subset of the Java SE APIs and tools, as well as additional libraries for mobile-specific features like user interfaces and networking.**
Java FX	**A platform for developing rich client applications and user interfaces, with features like graphics and media support, animation, and web integration. It includes a set of APIs and tools for developing desktop, mobile, and web applications.**

Note that Java EE has been rebranded as Jakarta EE, as Oracle transferred ownership of the Java EE specification to the Eclipse Foundation.

Java Versions

Java Version	Key Features
Java 1.0	Garbage collection, multi-threading, network programming
Java 1.1	Inner classes, JavaBeans, JDBC
Java 1.2 (Java 2)	Collections framework, reflection, Swing GUI toolkit, JNDI
Java 1.3 (Java 2)	HotSpot JVM, Java Sound API, Bluetooth support
Java 1.4 (Java 2)	Assertions, regular expressions, Java Web Start, JNI
Java 5 (Java 1.5)	Generics, annotations, enhanced for loop, concurrent API
Java 6 (Java 1.6)	JAXB API, Java Compiler API
Java 7 (Java 1.7)	Try-with-resources, multi-catch exceptions, Fork/Join Framework, NIO.2
Java 8 (1.8)	Lambda expressions, Stream API, java.time package, CompletableFuture
Java 9	Module system, private interface methods, JShell
Java 10	Local variable type inference, garbage collector improvements
Java 11	HttpClient API, var keyword for lambda parameters, Unicode 10.0 support

Java 12	Switch expressions, garbage collector improvements
Java 13	Text blocks, Z Garbage Collector improvements
Java 14	Record classes
Java 15	Sealed classes and interfaces, garbage collector improvements
Java 16	Pattern matching for instanceof, garbage collector and JFR improvements
Java 17	Sealed Types Preview, garbage collector improvements

Java Comments

Comment Type	Syntax	Description
Single-line comment	// comment	A comment that extends from the // marker to the end of the line. It is ignored by the compiler and used for adding notes and explanations to the code.
Multi-line comment	/* comment */	A comment that can span multiple lines, starting with the /* marker and ending with the */ marker. It is also ignored by the compiler and used for adding longer explanations or temporarily disabling code.

Documentation comment	/** comment */	A comment that begins with /** and ends with */. It is similar to a multi-line comment, but it is specifically used for generating API documentation with tools like Javadoc. It can include tags like @param, @return, @throws, etc.
Javadoc comment	/**	A special type of documentation comment that starts with /** and includes additional Javadoc tags for generating documentation. It can be used to describe classes, interfaces, methods, fields, etc. and their functionality. It is a best practice to include Javadoc comments in your code for documentation purposes.

⚠️ **Why this Kolavari DI? (Reason for this error)**
Install JDK properly

```
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

⚠️ **Difference between Java and C++**

Java	C++
1. Java doesn't support Pointers.	1. C++ supports Pointer.
2. Java does not support multiple inheritance with classes.	2. C++ supports multiple inheritance with classes.
3. Java doesn't support Global Variables (Variables declared outside the class).	3. C++ supports Global Variables.
4. Java doesn't support header files.	4. C++ supports header files.
5. const and goto keywords can't be used in Java.	5. const and goto keywords can be used in C++.
6. Java doesn't support Destructors.	6. C++ supports Destructors.
7. Java doesn't support Virtual Functions.	7. C++ supports Virtual Functions.
8. Java doesn't support Operator Overloading.	8. C++ supports Operator Overloading.
9. Java doesn't support Scope Resolution Operator.	9. C++ supports Scope Resolution Operator (::).
10. Java has Built-in API for Multithreading and Network Programming.	10. C++ doesn't have Built-in API for Multithreading and Network Programming.
11. Java has a Garbage Collector to clean the memory automatically.	11. No automatic memory cleaning process in C++.

⚠️ Differences between Oracle JDK and OpenJDK

Both OpenJDK and Oracle JDK are created and maintained currently by Oracle only.

Most of the vendors of JDK are written on top of OpenJDK by doing a few tweaks to [mostly to replace licensed proprietary parts / replace with more high-performance items that only work on specific OS] components without breaking the TCK compatibility.

https://stackoverflow.com/questions/22358071/differences-between-oracle-jdk-and-openjdk

⚠️ What is the use of setting the PATH?

Setting the PATH is important for the operating system to know where to look for executable files when a command is executed. In the case of Java, setting the PATH is essential for the system to locate the Java compiler and runtime environment (JRE) executable files.

⚠️ What is the reason for the following error?

- Javac : file not found Hello.java
- Error Could not find or load main class Hello
- java.lang.NoClassDefFoundError : hello(Wrong name Hello)
- java.lang.ClassFormatERROR : Incompatible magic value (change in .class)
- java.lang.UnSupporttedClassVersion : Hello (Compiled with Old)

⚠️What are the ways available to set the PATH?

There are several ways to set the PATH, including using the command line, editing system environment variables, and using third-party tools or scripts. The specific method used may vary depending on the operating system being used.

⚠️ What is the difference between PRIVATE JRE and PUBLIC JRE?

A PRIVATE JRE is a Java runtime environment that is installed and used by a specific application or user, and is not shared with other applications or users on the same system. A PUBLIC JRE is a Java runtime environment that is installed and made available to all applications and users on the system.

⚠️ Which JRE will be used while compiling the Java Source File?

The JRE is not used for compiling Java source files, but rather the Java Development Kit (JDK) is used, specifically the Java compiler (javac) which is included in the JDK.

⚠️ What will happen when I am compiling Java program without PRIVATE JRE?

Compiling a Java program does not require a PRIVATE JRE. However, if the program requires a specific version of the JRE to execute, and that version is not installed on the system, the program may not run properly.

⚠️ Which JRE will be used while executing Java application?

TheTestingAcademy

The JRE that is used to execute a Java application depends on several factors, including the system's PATH settings, the application's CLASSPATH settings, and any environment variables that may be set.

⚠️ **What will happen when I am executing Java program without PUBLIC JRE?**
If a PUBLIC JRE is not installed on the system, the Java application will not be able to run. The user will need to install a JRE before the application can be executed.

⚠️ **Can we execute java program without setting PATH?**
It is possible to execute a Java program without setting the PATH, but it requires specifying the full path to the Java executable file every time the command is executed.

⚠️**What is the difference between JVM and JRE?**
JVM (Java Virtual Machine) is a virtual machine that runs Java bytecode, while JRE (Java Runtime Environment) is a software package that includes the JVM as well as other libraries and components required to run Java applications.

⚠️ **What is the difference between JIT Compiler and Java Interpreter?**
A JIT (Just-In-Time) compiler is a component of the JVM that dynamically compiles bytecode into machine code at runtime, while a Java Interpreter executes bytecode directly without compiling it.

⚠️ **What is the difference between Byte code and Native code?**
Bytecode is a machine-independent code that is generated by the Java compiler and is designed to be executed by the JVM. Native code, on the other hand, is machine-specific code that is directly executable by the CPU.

⚠️ **Is it possible to execute Java program without having JDK in machine?**
It is possible to execute a Java program without having the JDK (Java Development Kit) installed on the machine, but a JRE (Java Runtime Environment) must be installed.

⚠️ **Can I install multiple JDK versions in my machine?**
Yes, it is possible to install multiple JDK (Java Development Kit) versions on the same machine. It is important to manage the PATH and other environment variables correctly to ensure that the correct version is used when compiling or executing Java programs.

⚠️**Can I install multiple JRE versions in my machine?**
Yes, it is possible to install multiple JRE (Java Runtime Environment) versions on the same machine. It is important to manage the PATH and other environment variables correctly to ensure that the correct version is used

TheTestingAcademy

Explain the main method in Java

- The main method in Java is the entry point for any Java program.

- It is a predefined method that is executed when a Java program is run, and is required in every Java program.
- The main method has a specific signature, which includes the keyword "public", the keyword "static", the return type "void", and a parameter of type "String" array named "args". Here is an example of the main method signature:
- public static void main(String[] args) {
- // code to be executed
- }
- The main method takes an optional argument, "args", which is an array of strings that ca
- n be used to pass command-line arguments to the program.

Steps that the JVM takes to call the main method:

- Loads the necessary classes for the program.
- Locates the entry point class specified on the command line.
- Locates the main method in the entry point class.
- Sets up the environment and executes the code inside the main method.
- When the main method completes, the JVM terminates the program.

JDK 11

✅ Keywords & Identifiers

- Simple English words which are having predefined meaning in Java Programming Language.
- Keywords are also called as Reserved Words.
- All the keywords are defined in Lower Case.
- We can't use keywords as names for variables, methods, classes, or as any other identifiers.

TheTestingAcademy

List Of Keywords			
Data types (8)	boolean short float	byte int double	char long
Class & Object (9)	class extends super	interface implements new	enum this instanceof
Access Modifier (3)	private	protected	public
Modifier (9)	final synchronized static	native transient const *	abstract volatile strictfp
Package (2)	package	import	
Control Statements (12)	if case while continue	else default break return	switch do for goto *
Exception Handling (6)	try throw	catch throws	finally assert
Other Data type (1)	void		

Identifiers - names those will be used to identify the programming elements like classes, methods, variables, etc uniquely

Rules to follow when you define an Identifier:

1. Identifier can contain Alphabets, Digits, and two special symbol i.e. Dollar ($),
2. Underscore (_).
3. First character of an identifier must be an Alphabet or Dollar ($) or Underscore (_).
4. Keywords or Reserved words can't be used as Identifier.

Valid Identifiers	Invalid Identifiers
Hello Int getClassName studentEmail TOTAL_FFE	1stClass true student number student-email int

✅ Variables and Data Types in Java

Variables

- A variable is a container (storage area) used to hold data.
- Each variable should be given a unique name (identifier).
- Memory will be allocated for the variable while executing the program
- Value of the variable can be changed any number of times during the program execution.

Types of Variables

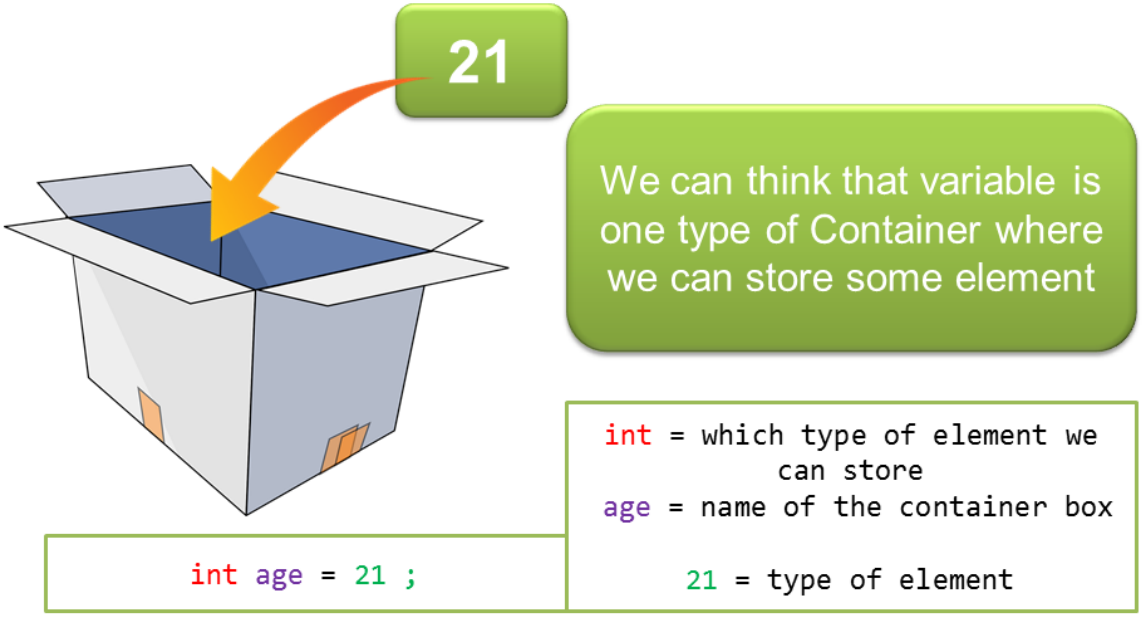
There are two types of variables based on data type used to declare the variable.

1) Primitive Variables

2) Reference Variables

```
1 package demo;
2
3 public class Demo {
4
5     public static void main(String[] args) {
6
7         int     result =    100 ;
8     }
9 }
```

datatype variable_name variable_value



Some Points

- A variable is a storage location paired with an associated symbolic name (an identifier), which contains some known or unknown quantity of information referred to as a value.

- Variables in Java are strongly typed; thus they all must have a data type declared with their identifier

- There are three rules to create an identifier:
    - Characters from A to Z, as well as their lowercase counterparts, can be used.
    - Numbers from 0-9 can be used.
    - Special characters that can be used are $ (the dollar sign) and _ (underscore).

```
int $  = 34;    // 44 - Yes
//        int _pramod = 34; // Yes
//        int 123  = 34; // NO
//        int 123_name  = 34; // NO
//        int 1name  = 34; // no
        //int #1  = 34; // No
//       System.out.println($);
```

Primitive Variables

- Variables declared with primitive data types are called as primitive variables.
- int a; int b = 99; double d1; double d2=9.9;

Reference Variables - Variables declared with user defined data types are called as reference variables.

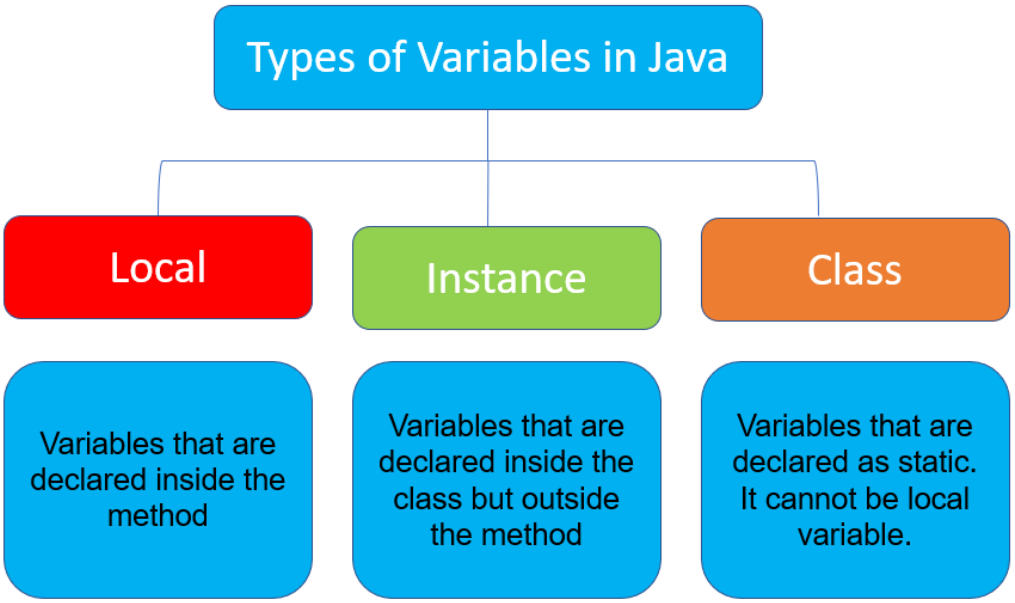
String str1;

String str2 = “TheTestingAcademy”;

Based On SCOPE variables are 3 types

- **Instance Variables :** Variables declared in the class without using static keyword are called as Instance Variables.
- **Static Variables :** Variables declared in the class using static keyword are called as Static variables.
- **Local Variables -** Variables declared in the member of the class like method etc are called as Local variables.

TheTestingAcademy




```
class Hello{
        int a;          → Instance Variable
        static int b;   → Static Variable
        void show(){
                int c;  → Local Variable
        }
}
```

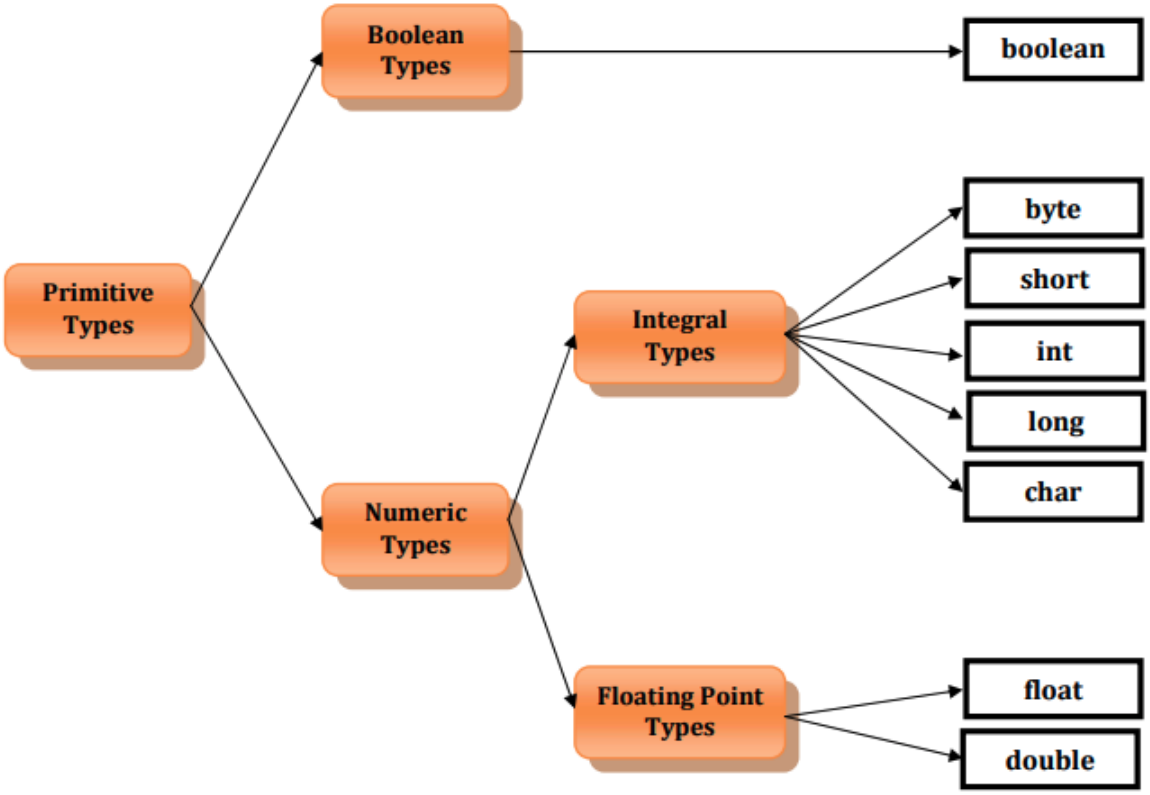
- Declare variables with default values
- Multiple variable declare.

Data Types

- Type of data you want to use
- Amount of memory allocation required for your data.
    - Size, Range

There are two types of data types
1) Primitive Data Types
2) User Defined Data Types

Primitive data types
They are the basic data types in Java and are used to store simple values.



There are eight primitive data types in Java, which are categorized into four groups: integer, floating-point, character, and boolean.

Data Type	Size (bits)	Default Value	Example
byte	8	0	byte b = 10;
short	16	0	short s = 100;

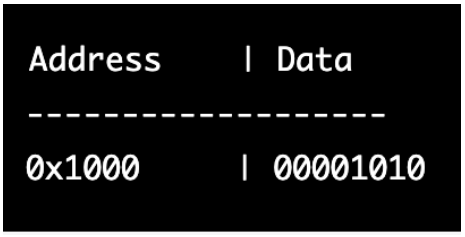
int	32	0	int i = 1000;
long	64	0L	long l = 100000L;
float	32	0.0f	float f = 1.23f;
double	64	0.0d	double d = 1.23;
char	16	'\u0000'	char c = 'A';
boolean	1	false	boolean b = true;

How byte b = 10 is stored in JVM?

When the statement byte b = 10 is executed in Java, the JVM creates a variable named b of type byte and assigns it the value 10.

Internally, the value 10 is represented in binary format as 00001010 (since byte data type is 8 bits in size), and it is stored in memory as a sequence of 8 bits. The JVM allocates a specific memory location for the variable b, and the binary representation of the value 10 is stored at that memory location.

To illustrate this, let's assume that the memory location allocated for the variable b is **0x1000**. The binary representation of the value 10 is 00001010, which can be stored in a single byte of memory. Therefore, the value 10 is stored at memory location 0x1000 as follows:



Reference data types are used to store complex objects, such as arrays and classes. These data types are created using predefined or custom classes.

User Defined Data types

Four types of User Defined Data types:

- Class type - String
- Interface type
- Enum type (From JAVA 5)
- Annotation type (From JAVA 5)

Type	Size		Default Value	Min Value	Max Value
	Byte	Bits			
boolean	N/D	N/D	false	true or false	
byte	1	8	0	-2^7 (-128)	2^7 -1 (127)
char	2	16	ASCII – 0 Unicode - \u0000	0	2^{16}-1 (65535)
short	2	16	0	-2^{15} (-32,768)	2^{15}-1 (32,767)
int	4	32	0	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
long	8	64	0	-2^{63}	2^{63}-1
float	4	32	0.0	1.4E-45	3.40E38
double	8	64	0.0	4.9E-324	1.79E308
Any Reference Type	8	64	null	Reference of the corresponding type object	

Types of Variables There are two types of variables based on data type used to declare the variable.

1) Primitive Variables - Variables declared with **primitive data types** are called as primitive variables.

byte b; int a; double d; char ch;

2) Reference Variables - Variables declared **with reference data types** are called as reference variables

String str; Hello h;

✅ Constants

- Special variable whose value can't be modified during the program execution.
- Constant is also called as final variable.

final int A=99;
final String STR="TTA";
final double D1=999.99;

Variables & Constants

1) Default value of char is either ASCII - 0 or UNICODE \u0000.
2) Default value of char is not space.
3) JVM will not provide default value for Local Variable.
4) Local variable must be initialized before using .
5) JVM will provide default value for static Variable.
6) Static variable can be accessed from Static method (main method).
7) JVM will provide default value for instance Variable.
8) Instance variable cannot be accessed directly from Static method (main method).
9) You can't declare multiple variables of different data types in single variable declaration statement.
10) You can declare multiple variables of same data types in a single statement.
11) You can assign same value to multiple variables in a single statement.
12) You can declare and initialize multiple variables of same data type in a single statement.
13) We can't declare two variables with same name in same scope.
14) Value of the variable can be changed any number of times during program execution.
15) Value of the final variable can not be changed.
16) We can't use const keyword to declare constant.

TheTestingAcademy

Q1	How many keywords are available prior to Java 5?	48

Q2	What is the keyword added newly in Java 5?	enum
Q3	Which of the following are valid Java keywords? A) instanceof B) goto C) null D) assert E) struct F) true G) Long H) false I) virtual J) signed	B) goto E) struct I) virtual J) signed
Q4	What are the keywords available which can't be used in Java Program?	goto, const
Q5	What is an identifier? What are the rules to follow to define an identifier?	An identifier is a name assigned to a variable, method, class, or package. The rules to follow while defining an identifier are: i) It should start with a letter or underscore, ii) No whitespace or special characters allowed except underscore, iii) It should not be a reserved keyword.
Q6	What is data type? How many types of data types available?	Data type defines the type of value a variable can hold. There are two types of data types: i) Primitive data types, ii) Reference data types.
Q7	How many primitive data types available?	8
Q8	What is the Integral data type that will not allow negative value?	char

Q9	What is the default value of char data type?	'\u0000'
Q10	Why the range of short and char are different even both required 2 bytes of memory?	The range of short and char are different because short is signed and char is unsigned.
Q11	How many types of User defined data types available up to JDK1.4?	2
Q12	How many types of User defined data types available from Java 5?	1
Q13	Is String a data type?	No, it is a class.
Q14	What is the default value of reference data type?	null
Q15	What are the possible values can be used for boolean type?	true or false
Q16	What is a variable?	A variable is a container that stores values.
Q17	How many types of variables are available as per the data type?	2

Q18	How many types of variables are available as per the scope?	3
Q19	What are the differences between Primitive variables and Reference variables?	Primitive variables hold the actual values, while reference variables hold the memory location of the object.
Q20	What is the default value of local variable?	There is no default value for a local variable. It must be initialized before use.
Q21	How can I declare multiple types of variables in one statement?	Not possible. Variables must be declared separately for each data type.
Q22	How can I declare multiple variable of same type in one statement?	Multiple variables of the same type can be declared in one statement by separating each variable name with a comma.
Q23	How can I declare and initialize a variable in one statement?	Declare and initialize a variable in one statement by assigning a value to it when it is declared.
Q24	How can I declare and initialize multiple variables of same type in one statement?	Declare and initialize multiple variables of the same type in one statement by separating each variable name with a comma and assigning a value to each variable when it is declared.

Q25	Can I change the value of variable during program execution?	Yes
Q26	When will the memory be allocated for the variables?	Memory for variables is allocated at runtime, while the program is executing.

UNICODE Characters

- UNICODE stands for UNIversal CODE.
- Every character will have UNICODE value.
- UNICODE Notation
- Syntax:
    - \uXXXX - X will be hexadecimal digit
- Starts with \u followed by four hexadecimal digits.
- UNICODE Range
- \u0000 (0) to \uFFFF (65535)

TheTestingAcademy

Character	ASCII	OCTAL	UNICODE	Character	ASCII	OCTAL	UNICODE
A	65	101	\u0041	a	97	141	\u0061
B	66	102	\u0042	b	98	142	\u0062
C	67	103	\u0043	c	99	143	\u0063
D	68	104	\u0044	d	100	144	\u0064
E	69	105	\u0045	e	101	145	\u0065
F	70	106	\u0046	f	102	146	\u0066
G	71	107	\u0047	g	103	147	\u0067
H	72	110	\u0048	h	104	150	\u0068
I	73	111	\u0049	i	105	151	\u0069
J	74	112	\u004A	j	106	152	\u006A
K	75	113	\u004B	k	107	153	\u006B
L	76	114	\u004C	l	108	154	\u006C
M	77	115	\u004D	m	109	155	\u006D
N	78	116	\u004E	n	110	156	\u006E
O	79	117	\u004F	o	111	157	\u006F
P	80	120	\u0050	p	112	160	\u0070
Q	81	121	\u0051	q	113	161	\u0071
R	82	122	\u0052	r	114	162	\u0072
S	83	123	\u0053	s	115	163	\u0073
T	84	124	\u0054	t	116	164	\u0074
U	85	125	\u0055	u	117	165	\u0075
V	86	126	\u0056	v	118	166	\u0076
W	87	127	\u0057	w	119	167	\u0077
X	88	130	\u0058	x	120	170	\u0078
Y	89	131	\u0059	y	121	171	\u0079
Z	90	132	\u005A	z	122	172	\u007A

TheTe

Practice QnA

Give the output of the following in the following

1. It will not compile.
2. At runtime error
3. Condition 1
4. Condition 2

Question 1
int enum=9;
System.out.println(enum);

Question 2
char char='A'; System.out.println(char);

TheTestingAcademy

✅ Literals

- Literals are the actual values assigned
- Literals can be Numeric and Non Numeric.

Literal Type	Description
Integer literals	Whole numbers without decimal points, such as 42 or -123
Floating-point literals	Numbers with decimal points, such as 3.14 or -0.0025
Character literals	A single character enclosed in single quotes, such as 'a'
Boolean literals	A value of either true or false
String literals	A sequence of characters enclosed in double quotes, such as "hello world"
Null literals	A special literal that represents a null reference or a null value

Types of Literals
1) Boolean Literals - true, false
2) Character Literals -
3) String Literals
4) Integral Literals
5) Floating Literals
6) null Literal

1. Boolean Literals

There are two boolean literals 1) true 2) false

2. Character Literals

A char type variable can hold following:

- Single character enclosed in single quotation marks
- Escape Sequence
- ASCII Value
- UNICODE Character
- Octal Character

Diff between /n and /r
https://stackoverflow.com/questions/1761051/difference-between-n-and-r

Escape Sequence - **Task_12.java**

Escape Sequence	Description
\t	Tab Space.
\b	Backspace.
\n	Newline.
\r	Carriage return.
\f	Formfeed.
\'	Single quote character.
\"	Double quote character.
\\	Backslash character.

ASCII stands for American Standard Code for Information Interchange.

- Every character enclosed in single quotation marks will have an integer equivalent value
- called as ASCII value.
- ASCII Value Range is 0 – 255.
- ASCII Value can be assigned to a char type variable

Octal Value as char type (0)

061 -> 49
061 = $(0 \times 8^2) + (6 \times 8^1) + (1 \times 8^0) = 49$

https://www.rapidtables.com/convert/number/octal-to-decimal.html

TheTestAcademy

Character	ASCII	OCTAL	UNICODE	Character	ASCII	OCTAL	UNICODE
0	48	060	\u0030	5	53	065	\u0035
1	49	061	\u0031	6	54	066	\u0036
2	50	062	\u0032	7	55	067	\u0037
3	51	063	\u0033	8	56	070	\u0038
4	52	064	\u0034	9	57	071	\u0039

Character	ASCII	OCTAL	UNICODE	Character	ASCII	OCTAL	UNICODE
A	65	101	\u0041	a	97	141	\u0061
B	66	102	\u0042	b	98	142	\u0062
C	67	103	\u0043	c	99	143	\u0063
D	68	104	\u0044	d	100	144	\u0064
E	69	105	\u0045	e	101	145	\u0065
F	70	106	\u0046	f	102	146	\u0066
G	71	107	\u0047	g	103	147	\u0067
H	72	110	\u0048	h	104	150	\u0068
I	73	111	\u0049	i	105	151	\u0069
J	74	112	\u004A	j	106	152	\u006A
K	75	113	\u004B	k	107	153	\u006B
L	76	114	\u004C	l	108	154	\u006C
M	77	115	\u004D	m	109	155	\u006D
N	78	116	\u004E	n	110	156	\u006E
O	79	117	\u004F	o	111	157	\u006F
P	80	120	\u0050	p	112	160	\u0070
Q	81	121	\u0051	q	113	161	\u0071
R	82	122	\u0052	r	114	162	\u0072
S	83	123	\u0053	s	115	163	\u0073
T	84	124	\u0054	t	116	164	\u0074
U	85	125	\u0055	u	117	165	\u0075
V	86	126	\u0056	v	118	166	\u0076
W	87	127	\u0057	w	119	167	\u0077
X	88	130	\u0058	x	120	170	\u0078
Y	89	131	\u0059	y	121	171	\u0079
Z	90	132	\u005A	z	122	172	\u007A

Syntax: \DDD - D will be octal digit
OCTAL Range Range in Decimal 0 - 255 Range in Octal \0 - \377

Question	Answer
What is a literal?	A literal is a fixed value that is directly used in a program without needing to be computed or evaluated.
How many types of literals are available?	There are six types of literals available in programming languages: numeric literals, character literals, boolean literals, string literals, array literals, and null literals.
How many boolean literals are available?	There are only two boolean literals available, which are true and false.
What will happen when I assign 1 to boolean type variable?	When you assign the value 1 to a boolean type variable, it will be interpreted as true because any non-zero value is considered as true in boolean expressions.
Can we store empty character in char type variable?	No, you cannot store an empty character in a char type variable because it requires at least one character.
How to store single quote in char variable?	To store a single quote in a char variable, you need to use the escape sequence ' because the single quote is a reserved character in programming languages.
What is Escape Sequence?	An escape sequence is a combination of characters used to represent special characters or non-printable characters in a string literal. It usually starts with a backslash () character.
What will be displayed when UNICODE value is found in String Literal? String str="UNICODE of A is \u0041";	The string "UNICODE of A is A" will be displayed because \u0041 represents the Unicode value of the letter A.

What will be displayed when Octal representation is found in String Literal? String str="Octal of A is \101";	**The string "Octal of A is A" will be displayed because \101 represents the octal value of the letter A.**
What will happen when Escape Sequence is found in String Literal? String str="Hello\nGuys";	**The string "Hello" will be displayed on the first line, and "Guys" will be displayed on the next line because \n represents the newline character.**

Integer Literals

- Decimal Literals- // Base of 10
- Octal Literals - int b=0101; // Base of 8
- Hexadecimal Literals - int c=0Xface; // base 16 rather than base 10
- Binary Literals (From Java 7) - int b2 = 0b101;

Scientific / Exponent N Literals

```
float f = 129.8763e2F;
double d1 = 129.8763e+2;
double d2 = 12987.63e-2;
System.out.println(f);
System.out.println(d1);
System.out.println(d2);
```

Null literal

- null is a value.
- It is default value for any reference variable.
- If the value of reference variable is null then it indicates that address/reference is not
- available in the variable.

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

TheTestingAcademy

✅ Operators

Operators are used to perform operations by using operands.
There are three types of operator depending on the number of operands required.

- 1) Unary Operator
  - Only one operand is required.
- 2) Binary Operator
  - Two operands are required.
- 3) Ternary Operator
  - Three Operands are required.

Types of operator depending on the operation:

- Arithmetic Operators
  - i. Unary Arithmetic Operators
  - ii. Binary Arithmetic Operators
- String Concatenation Operator
- Assignment Operator
  - i. Simple Assignment Operator
  - ii. Compound Assignment Operators
- Increment & Decrement Operators
- Relational Operators
- Logical Operators
- new Operator
- instanceof Operator
- Conditional or Ternary Operator
- Bitwise Operators.

TheTestingAcademy

Unary Arithmetic Operators

- unary minus(-)
- increment(+)
  - Pre
  - post
- decrement(-)
  - Pre
  - post
- NOT(!) - boolean and condition
- Bitwise Complement (~) - unary operator returns the one's complement representation of the input value or operand,
  a = 5 [0101 in Binary]
  result = ~5

- Addressof operator(&)
- sizeof()

Binary Arithmetic Operators

Operator	Description
+	Addition (SUM)
-	Subtraction (DIFFERENCE)
*	Multiplication (PRODUCT)
/	Division (QUOTIENT)
%	Modulus (REMAINDER)

Remember this

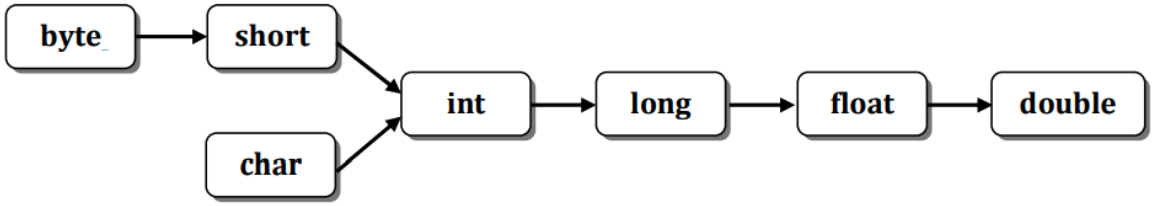
1st Operand Type	2nd Operand Type	Result Type
byte	byte	int
char	char	int
short	byte	int
int	int	int
int	long	long
byte	double	double
float	int	float
float	long	float
char	double	double
int	double	double
char	int	int
short	char	int
short	float	float

String Concatenation Operator (+)

+ operator can be used for two purposes:
1) Arithmetic Addition
2) String Concatenation

Simple Assignment Operator (=)

It is a binary operator.
It is used to assign the value to a variable. =
o Operand1 must be a variable.
o Operand2 can be a variable, value or expression.



Type Casting

Source type is not same as destination type then source type must be converted to destination type.

Type casting can be done in two ways:
o Implicit casting
o Explicit Casting

Type casting can be done in two ways:
o Implicit casting
o Explicit Casting

Implicit casting:
When type casting is happening automatically by the compiler then it is called as Implicit Casting.

Explicit Casting:
When type casting is happening explicitly by the programmer then it is called as Explicit Casting.
There are two types of conversions:
o Widening
o Narrowing

Widening

Widening is the process of converting lower type to higher type. This is safe conversion.
Ex:
byte b=10;
int a=b; // VALID – Implicit Casting
int a= (int)b; // VALID – Explicit Casting

Narrowing is the process of converting higher type to lower type.
Ex:
int a=300;
byte b=a; // INVALID – Implicit Casting
byte b = (byte)a; // VALID – Explicit Casting

int a=300; -> int means 32 bits memory required
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0
byte b1=(byte)a; -> byte means 8 bits memory required
0 0 1 0 1 1 0 0
Value is
0 * 27 + 0 * 26 + 1 * 25 + 0 * 24 + 1 * 23 + 1 * 22 + 0 * 21 + 0 * 20
0 + 0 + 32 + 0 + 8 + 4 + 0 + 0 = **44.**

Compound Assignment Operators

int a=90; a+=9; => a = a+9;

Operator	Description
+=	Add right operand to left operand and assign the result to left operand.
-=	Subtract right operand from left operand and assign the result to left operand.
*=	Multiply right operand to left operand and assign the result to left operand.
/=	Divide right operand to left operand and assign the result to left operand.
%=	Calculate modulus using two operands and assign the result to left operand.

Increment (++) / Decrement (--) Operators

- Increment operator will be used to increase the value of the variable by 1.
- Decrement operator will be used to decrease the value of the variable by 1.
- Increment / Decrement operator can't be applied for value/constant.

Relational Operators

Operator	Description	Operands
>	Greater than	Numeric or char
<	Less than	Numeric or char
>=	Greater than or equal to	Numeric or char
<=	Less than or equal to	Numeric or char
==	Equal to	Numeric or char
!=	Not equal to	Numeric or char

Logical Operators

Operator	Description
!	Logical NOT
&&	Logical AND
||	Logical OR

Logical OR and Logical AND (|| , &&

Logical OR:

Operand 1	Operand 2	Result
false	false	false
false	true	true
true	false	true
true	true	true

Logical AND:

Operand 1	Operand 2	Result
false	false	false
false	true	false
true	false	false
true	true	true

new Operator

- new operator is used to create the new object for class.
- It returns of the address of newly created object.
- String s1 = new String(“Pramod”);

instanceof Operator

- It is used to check whether the given object belongs to specified class or not.
- Result of instanceof operator is boolean value.
- It returns true if given object belongs to specified class otherwise false.
- It is also called aCOMPARISONERATOR.
- System.out.println(s1 instanceof Object);

Conditional Operator

- It is ternary operator. ? : Operand1 must be of boolean type.
- If Operand1 is true then Operand2 will be returned otherwise Operand3 will be returned.

Bitwise Operators

Operator	Description
~	Bitwise NOT
&	Bitwise AND
|	Bitwise OR
^	Exclusive OR (XOR)
<<	Left Shift
>>	Right Shift

Bitwise AND o Bitwise OR o Exclusive OR (XOR)

https://bit-calculator.com/bit-shift-calculator

---

1. 7+ (6 + 5 *3) - 4/2
2. boolean b = (boolean)1; b)
3. String str=(String)99;

TheTestingAca

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= & = ^= |= <<= >>= >>>=

Interview QnA

Question	Answer
What is the use of bitwise operators?	Bitwise operators are used to perform bitwise operations on binary representations of integers.
What is the result type of String concatenation operation?	The result type of String concatenation operation is always a String.
What will happen when I try to assign int type to String type variable? Ex: String str=99;	The code will not compile because you cannot assign an int value to a String variable directly.
What are types of operands allowed for comparison operators?	Comparison operators can be used with numeric and boolean operands.

What is the difference between prefix and postfix forms of increment operator?	The prefix form increments the value before it is used, while the postfix form uses the value and then increments it.
What is type casting and what are the types available?	Type casting is a way to convert a value from one data type to another. The types available are widening conversion (implicit) and narrowing conversion (explicit).
What is the use of new operator?	The new operator is used to create an instance of a class or an array.
What is the difference between String Concatenation and Arithmetic Addition operator?	The String concatenation operator (+) is used to concatenate two strings, while the arithmetic addition operator (+) is used to add two numeric values.
What are the bitwise operators available?	The bitwise operators available are: ~ (bitwise NOT), & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), << (left shift), >> (right shift), and >>> (unsigned right shift).
What are equality operators available?	The equality operators available are: == (equal to) and != (not equal to).
What is the difference between Implicit casting and Explicit Casting?	Implicit casting is done automatically by the compiler, while explicit casting requires the programmer to specify the conversion explicitly.
What is the use of assignment operator?	The assignment operator is used to assign a value to a variable or an array element.

What is 2's Complement?	2's complement is a way of representing negative integers in binary. It involves flipping all the bits and adding 1 to the result.
What are types of operands allowed for arithmetic operators?	Arithmetic operators can be used with numeric operands (int, long, float, double) and char operands.
What is the difference between unary arithmetic and binary arithmetic operators?	Unary arithmetic operators act on a single operand, while binary arithmetic operators act on two operands.
What is the syntax of Explicit casting?	The syntax of explicit casting is: (datatype) value.
What are types of operands allowed for bitwise operators?	Bitwise operators can be used with integer operands (int, long, short, char, byte).
What is the use of Logical NOT operator?	The logical NOT operator (!) is used to negate a boolean value.
What is the difference between prefix and postfix forms of decrement operator?	The prefix form decrements the value before it is used, while the postfix form uses the value and then decrements it.
What is the use of instanceof operator?	The instanceof operator is used to check if an object is an instance of a particular class or interface.
What is result type when you add two byte variables?	The result type when you add two byte variables is int.

What is result type when you add long and float variables?	The result type when you add long and float variables is float.

Question	Answer
What is an operator?	An operator is a symbol that represents a specific operation to be performed between operands.
What are the valid conditions for "Infinity" output?	Division by a positive number or multiplication by a positive infinity.
What is the use of arithmetic operators?	Arithmetic operators are used to perform mathematical calculations between operands.
What is the result type when you add long and float variables?	The result type is a float.
What is the difference between String Concatenation and Arithmetic Addition operator?	String concatenation is used to join two or more strings together, while arithmetic addition is used to add numeric values.
What is the result of new operator?	The new operator is used to allocate memory for an object and returns a reference to that object.
What is the result of instanceof operator?	The instanceof operator returns true if an object is an instance of a particular class, or a subclass of that class.

What is the use of bitwise operators?	Bitwise operators are used to perform operations at the bit level.
What is type casting and what are the types available?	Type casting is the process of converting one data type to another. The two types of type casting are implicit casting and explicit casting.
What will happen when I try to assign int type to String type variable?	It will result in a compilation error because they are incompatible types.
What is the use of assignment operator?	The assignment operator is used to assign a value to a variable.
What is the difference between unary arithmetic and binary arithmetic operators?	Unary operators work on a single operand, while binary operators work on two operands.
What is the use of Logical NOT operator?	The logical NOT operator is used to reverse the truth value of a boolean expression.
What is the difference between Implicit casting and Explicit Casting?	Implicit casting is done automatically by the compiler, while explicit casting is done manually by the programmer.
What are compound assignment operators available?	Compound assignment operators combine an arithmetic or bitwise operation with an assignment.
What is 2's Complement?	2's complement is a binary representation of a signed number that uses the most significant bit as a sign bit.

What are types of operands allowed for bitwise operators?	Bitwise operators can be applied to integer types: byte, short, int, and long.
What is the syntax of Explicit casting?	The syntax of explicit casting is: (target_type) expression.
What are the relational operators?	The relational operators compare two values and return a boolean result: <, >, <=, >=.
What is result type when you add byte and int variables?	The result type is an int.
What are the bitwise operators available?	The bitwise operators are: &,
What is the difference between prefix and postfix forms of decrement operator?	The prefix form evaluates and returns the value after the operation, while the postfix form evaluates and returns the value before the operation.
What is the result type of String concatenation operation?	The result type is a String.
What is the use of instanceof operator?	The instanceof operator is used to test whether an object is an instance of a particular class or not.
What is the difference between Widening and Narrowing?	Widening casting occurs when you convert a smaller type to a larger type, while narrowing casting occurs when you convert a larger type to a smaller type.

✅Conditions and Loop

If
condition -> true or fasle

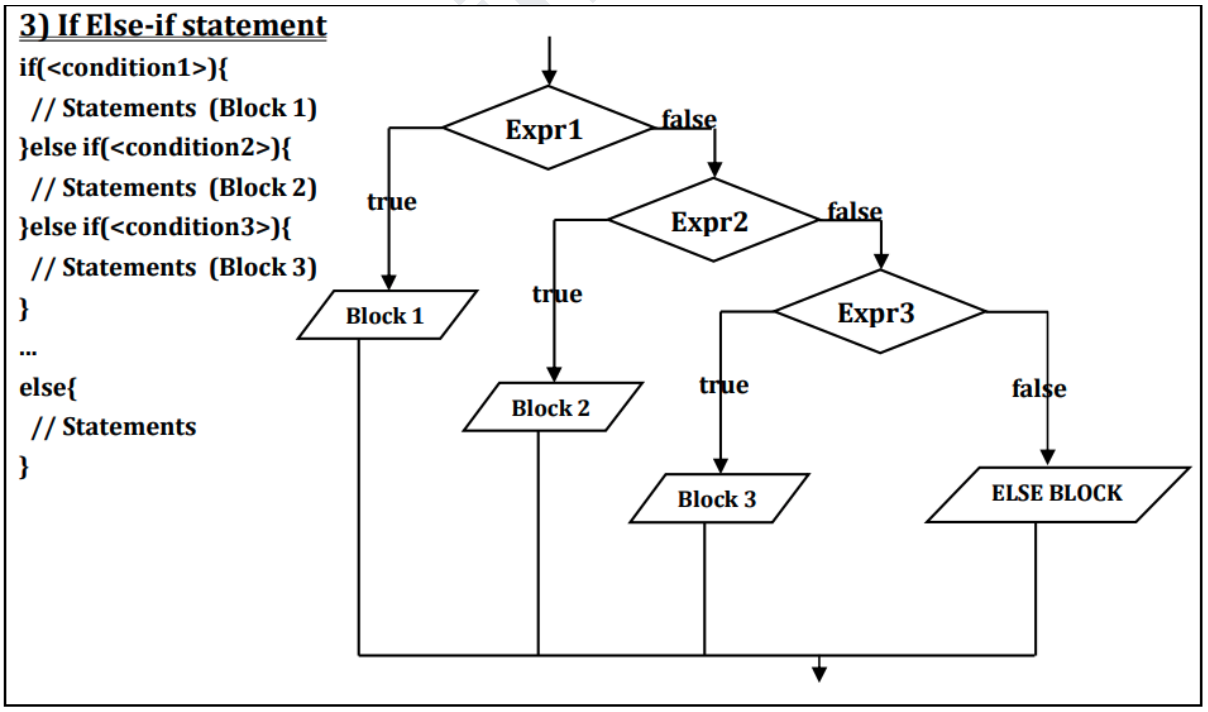
if(condition)
{ // Statements (IF BLOCK) }

Problem to find the Even and Odd with % Mod

```
if(a % 2==0)
 System.out.println("Value is EVEN");
else
 System.out.println("Value is ODD");
}
```

Problem to find the MAX three

```
if(a>b && a> c)
 max=a;
else if(b> a && b>c)
 max = b;
else
 max=c;
System.out.println("Max value is "+max);
```



Switch multi-way branch statement

- [ ] Java switch statement executes one statement from multiple conditions.
- [ ] Java switch expressions must be of **byte, short, int, long(with its Wrapper type), enums and string**. Beginning with JDK7, it also works with enumerated types (Enums in java), the String class, and Wrapper classes.
- [ ] case/s values are not allowed.
- [ ] The value for a case must be constant or literal. **Variables are not allowed.**
- [ ] break can't be used with elseif condition

```
// switch statement
switch(expression)
{
  // case statements
  // values must be of same type of expression
  case value1 : expression = value 1 ->
    // Statements
    break; // break is optional

  case value2 :
    // Statements
    break; // break is optional

  // We can have any number of case statements
  // below is default statement, used when none of the cases is true.
  // No break is needed in the default case.
  default :
    // Statements
}
```

```
String browser = "chrome"
switch(browser){
  case "chrome" :
                        sop("chrome started!!");
    break;

}
```

TheTestingAcademy

- **Default fall through due to missing break - JDK 13**

Much improved switch accepts multiple values per case.

```
switch (itemCode) {
   case 001, 002, 003 :
      System.out.println("It's an electronic gadget!");
      break;

   case 004, 005:
```

```
        System.out.println("It's a mechanical device!");

}
```

- **yield is used to return a value**

A new keyword **yield** has been introduced. It returns values from a switch branch only.

We don’t need a break after yield as it automatically terminates the switch expression.

```
int val = switch (code) {
    case "x", "y" :
        yield 1;
    case "z", "w" :
        yield 2;
}
```

Switch can be used as an expression

```
String text = switch (itemCode) {
    case 001 :
        yield "It's a laptop!";
    case 002 :
        yield "It's a desktop!";
    case 003 :
        yield "It's a mobile phone!";
    default :
        throw new IllegalArgumentException(itemCode + "is an unknown device!");
}
```

Switch with arrows

```
switch (itemCode) {
    case 001 -> System.out.println("It's a laptop!");
    case 002 -> System.out.println("It's a desktop!");
    case 003,004 -> System.out.println("It's a mobile phone!");
}
```

TestingAcademy

Case label can be any of the following:

Literals	Final Variables	Expression with Literals or Final Variables

case 9: … ;
case 'A': … ;
	final int A=99;
	case A: … ;
		case 10 + 20: … ;
		
		final int A=90;
		final int B=9;
		case A+B : … ;

TheTestingAcademy

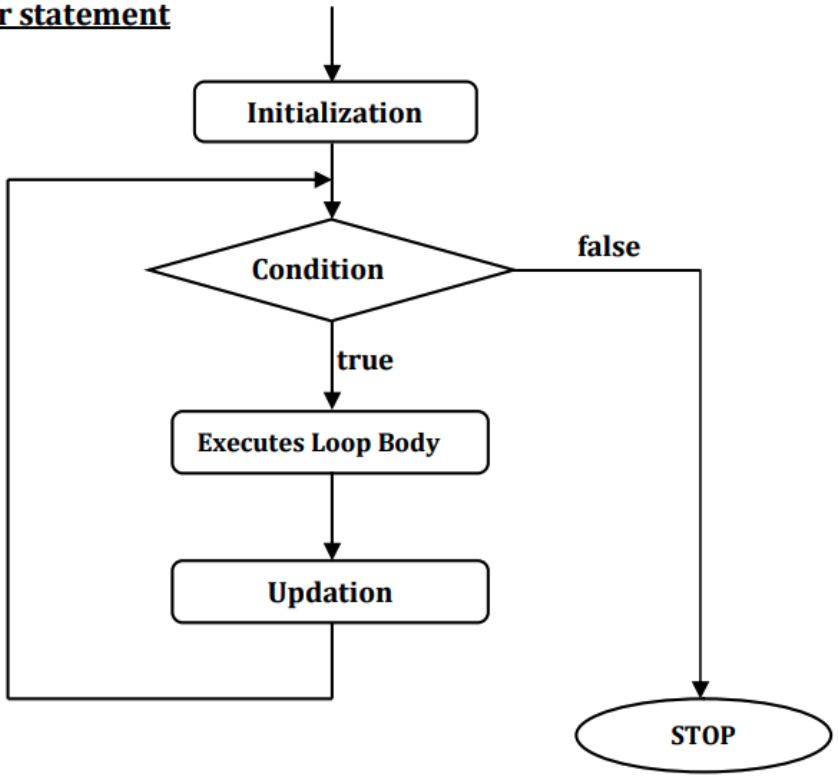
Case label can't be the following:

Variable	Blank Final Variables	Expression with Variables or Blank Final Variables

int a=99;
 	
case a: ... ;
	final int A;
	A = 99;
	
	case A: ... ;
		int a=90;
		int b = 9;
		case a + b: ... ;
		
		final int A;
		A = 90;
		final int B=9;
		
		case A+B : ... ;

For

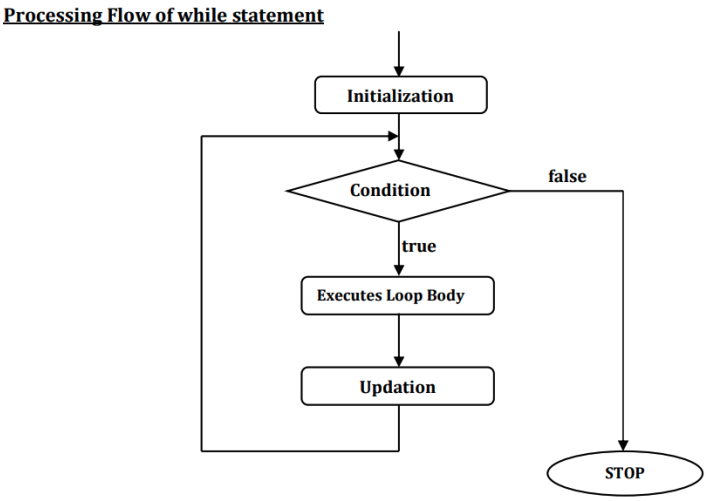
Processing Flow of for statement



1.

While

- condition of while statement is mandatory and must be boolean type



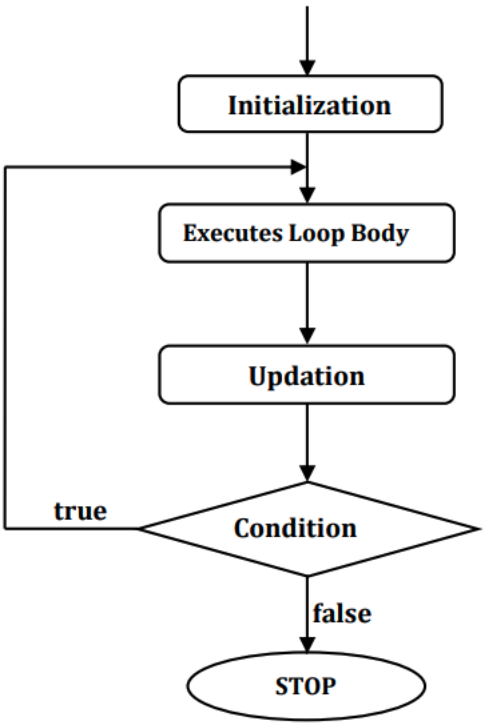
Do While

- When you are using for statement or while statement then it verifies the condition before executing the block.
- So in the case of for statement and while statement, when first time condition is false
- then the block will not be executed.
- for and while statement are also called as **Entry Controlled Loop.**
- If you want to execute the block at least once then use do-while statement

In the case of do-while first block of statements will be executed and then condition will be verified.

TheTestingAcademy

Processing Flow of while statement



// While -> Condition -> Body -> Increment

// Do While -> Body -> Condition -> Increment

break Statement and continue

- break is a keyword.
- It can be used within switch or any looping statement.
- It is used to terminate the execution of the current looping/switch statement.
- break can be used in two ways:
- break;
- break <label>;

Continue

- continue is a keyword.
- It can be used within any looping statements.
- It is used to continue the execution of the current looping statement with next iteration.
- continue can be used in two ways:
    - continue;
    - continue <label>;

✅ Arrays

- Array is a collection of data which is of similar type.
- Array is also called as Homogeneous data structure.
- Elements of an array will be stored in contiguous memory locations.
- Arrays are objects in Java.
- Three tasks to remember while you are working with arrays:
    - o Array Declaration
    - o Array Construction
    - o Array Initialization
- Arrays can be constructed with multiple dimensions i.e 1-D Array, 2-D Array etc.
- length
- Use loop to traverse.
- Array size is mandatory while constructing an array object. a. int a[] =new int[3]; //VALID b. int a[] =new int[]; //INVALID
- You can't specify the size of an array at the time of declaration. a. int a[] //VALID b. int a[1] //INVALID.
- When you are specifying -ve value as array size then java.lang.NegativeArraySizeException will be thrown at runtime

int arr[5]; String names[3];

int arr[]={10,20,30}; String names[]={"Sri","Manish","DK"};

What happens if we try to access elements outside the array size?
ArrayIndexOutOfBoundsException

Multidimensional Arrays

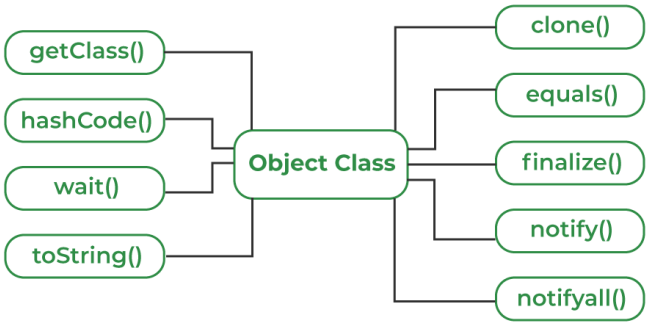
```
int [][] arr= new int[3][3];
    // 3 row and 3 column
```

TheTestingAcademy

✅ Object Class in Java

- Object class is present in java.lang package.
- Every class in Java is directly or indirectly derived from the Object class.
- Object class methods are available to all Java classes.

Ref image - geeksforgeeks



toString() method
In Java, the toString() method is a method defined in the Object class that returns a **string representation of an object.**

```
public class Person {
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  @Override
  public String toString() {
    return "Person[name=" + name + ", age=" + age + "]";
  }
}

Person p = new Person("John", 30);
System.out.println(p); // prints "Person[name=John, age=30]"
```

hashCode() method

In Java, the hashCode() method is a method defined in the Object class that returns an integer hash code value for an object.

The hashCode() method is implemented by converting the internal address of the object into an integer value. If two objects are equal according to the equals() method, then they should have the same hash code.

```
public class Person {
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  @Override
  public int hashCode() {
    int result = 17;
    result = 31 * result + name.hashCode();
    result = 31 * result + age;
    return result;
  }
}

Person p1 = new Person("John", 30);
Person p2 = new Person("John", 30);

System.out.println(p1.hashCode()); // prints a number
System.out.println(p2.hashCode()); // prints the same number as p1
```

equals(Object obj) method

In Java, the equals(Object obj) method is a method defined in the Object class that compares the current object to the specified object.

The method returns true if the objects are equal, and false if they are not.

The default implementation of the equals() method in the Object class compares the memory addresses of the objects

```
public class Person {
  private String name;
  private int age;
```

```
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  @Override
  public boolean equals(Object obj) {
    if (obj == this) return true;
    if (!(obj instanceof Person)) return false;
    Person other = (Person) obj;
    return name.equals(other.name) && age == other.age;
  }
}

Person p1 = new Person("John", 30);
Person p2 = new Person("John", 30);
Person p3 = new Person("Jane", 30);

System.out.println(p1.equals(p2)); // prints true
System.out.println(p1.equals(p3)); // prints false
```

getClass() method,

```
public class Test {
    public static void main(String[] args)
    {
        Object obj = new String("GeeksForGeeks");
        Class c = obj.getClass();
        System.out.println("Class of Object obj is : "
                           + c.getName());
    }
}

Output - Class of Object obj is : java.lang.String
```

finalize() method

This method is called just before an object is garbage collected. It is called the Garbage Collector on an object when the garbage collector determines that there are no more references to the object.

```
// Java program to demonstrate working of finalize()

public class Test {
```

```
	public static void main(String[] args)
	{
		Test t = new Test();
		System.out.println(t.hashCode());

		t = null;

		// calling garbage collector
		System.gc();

		System.out.println("end");
	}

	@Override protected void finalize()
	{
		System.out.println("finalize method called");
	}
}
```

clone()method

It returns a new object that is exactly the same as this object. For clone() method refer Clone().

```
public class Person implements Cloneable {
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  @Override
  public Person clone() {
    try {
      return (Person) super.clone();
    } catch (CloneNotSupportedException e) {
      // This should never happen, since we are Cloneable
      throw new InternalError(e);
    }
  }
}

Person p1 = new Person("John", 30);
Person p2 = p1.clone();
```

```
System.out.println(p1 == p2); // prints false
```

Note that the clone() method only creates a shallow copy of the object, which means that it only copies the references to the object's fields, rather than creating new copies of the objects themselves. If you want to create a deep copy of an object, you need to create a new copy of each object contained within the original object.

📙 Arrays in Java

- A group of like-typed variables referred to by a common name.
- Arrays are stored in contiguous memory.
- The variables in the array are ordered, and each has an index beginning from 0.
- The size of an array must be specified by int or short value and not long.
- The size of the array cannot be altered(once initialized).
- JVM throws ArrayIndexOutOfBoundsException to indicate that the array has been accessed with an illegal index

```
MyClass myClassArray[];
Object[]  ao,
```

int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

// Array of Objects
Student[] arr = new Student[5];

```
int[] numbers = new int[5];

// set the values of the array elements
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;

// print the values of the array elements
for (int i = 0; i < numbers.length; i++) {
  System.out.println(numbers[i]);
}

// prints 10, 20, 30, 40, 50
```

[Assignment] - highest and lowest

Java program that reads a list of integers from the user and then prints the highest and lowest values in the list.

```
Enter a number: 5
Enter a number: 3
Enter a number: 8
Enter a number: 2
Enter a number: -1

The highest number is 8
The lowest number is 2
```

import java.util.Scanner;

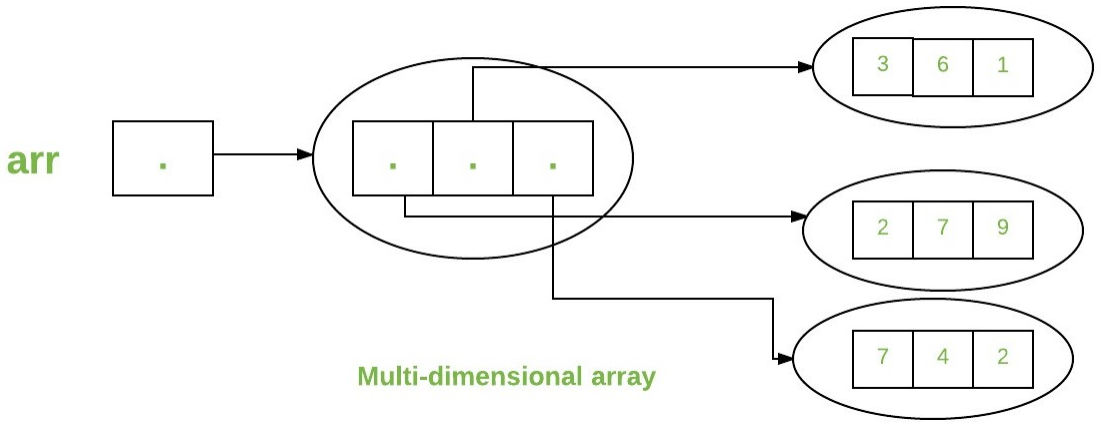
```
public class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // create an array to store the numbers
    int[] numbers = new int[100];
    int size = 0;

    // read the numbers from the user
    int n;
    do {
      System.out.print("Enter a number: " , -1 to stop entering”);
      n = scanner.nextInt();
      if (n >= 0) {
        numbers[size] = n;
        size++;
      }
    } while (n >= 0);

    // find the highest and lowest numbers
    int highest = numbers[0];
    int lowest = numbers[0];
    for (int i = 1; i < size; i++) {
      if (numbers[i] > highest) {
        highest = numbers[i];
      }
      if (numbers[i] < lowest) {
        lowest = numbers[i];
      }
    }

    // print the highest and lowest numbers
```

```
    System.out.println("The highest number is " + highest);
    System.out.println("The lowest number is " + lowest);
  }
}
```

For Each - For collections and arrays only

Multidimensional Arrays:
Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other arrays.

```
int [][] arr= new int[3][3];
```




```
public class multiDimensional {
      public static void main(String args[])
      {
            // declaring and initializing 2D array
            int arr[][]
                  = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

            // printing 2D array
            for (int i = 0; i < 3; i++) {
                  for (int j = 0; j < 3; j++)
                        System.out.print(arr[i][j] + " ");

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

🧵 String in Java

- Sequence of Characters.
- String is a **built-in class in java.lang package.**
- String is final class, so you can't define the subclass for String class.
- String class implements the following interfaces:
    - java.io.Serializable
    - java.lang.Comparable
    - java.lang.CharSequence
- String class has following variable to hold data.
    - private final char value[];
- String objects are immutable objects. It means once the object is created then the content or data of the object can't be modified.
- When you try to modify the contents of object then new object will be created as a result.

How String can be represented?

1) String class
2) StringBuffer
3) StringBuilder
4) Array of Characters
5) ArrayList of Characters

Why String is Immutable in Java?

The reason for making strings immutable in Java is to increase efficiency, security, and thread-safety.

Here is an example that demonstrates the immutability of strings in Java:

```
String str1 = "Hello";
String str2 = str1.concat(" World");
System.out.println(str1); // Output: Hello
System.out.println(str2); // Output: Hello World
```

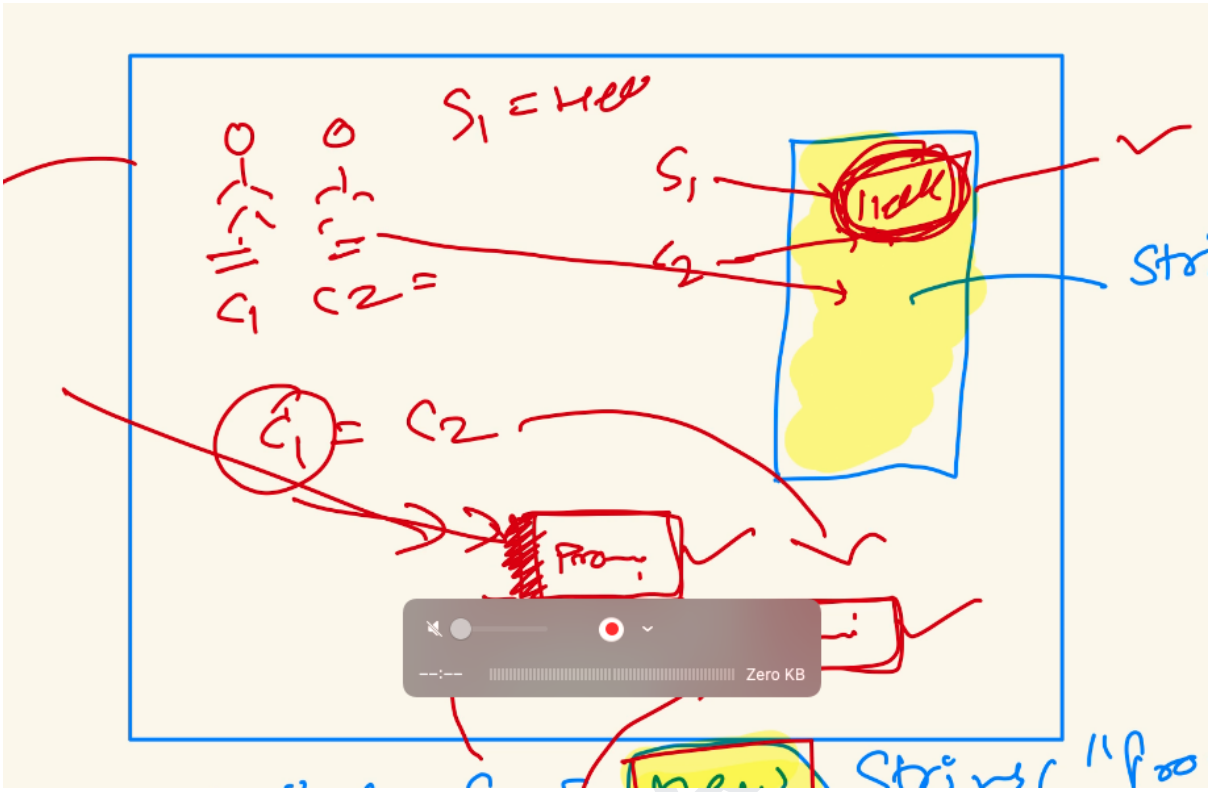
In this example, we create a string str1 with the value "Hello". We then concatenate the string " World" to str1 using the concat() method, and assign the result to str2. The concat() method does not modify str1 but creates a new string with the value "Hello World".

The original string str1 remains unchanged, and the new string str2 is assigned the concatenated value. This demonstrates the immutability of strings in Java.

https://www.digitalocean.com/community/tutorials/string-immutable-final-java

TheTestingAcademy

Concept of String Pool and New Operator



String class provides a wide range of functions to manipulate strings.

Here are some of the most commonly used functions of the String class:

charAt(int index): Returns the character at the specified index in the string.
Example: "hello".charAt(0) returns 'h'.

concat(String str): Concatenates the specified string to the end of the original string.
Example: "hello".concat(" world") returns "hello world".

contains(CharSequence s): Returns true if the string contains the specified sequence of characters, otherwise false.
Example: "hello world".contains("world") returns true.

equals(Object obj): Returns true if the string is equal to the specified object, otherwise false.
Example: "hello".equals("world") returns false.

equalsIgnoreCase(String str): Returns true if the string is equal to the specified string, ignoring case differences, otherwise false.
Example: "HELLO".equalsIgnoreCase("hello") returns true.

indexOf(int ch): Returns the index of the first occurrence of the specified character in the string, or -1 if the character is not found.

Example: "hello".indexOf('l') returns 2.

length(): Returns the length of the string.
Example: "hello".length() returns 5.

replace(char oldChar, char newChar): Returns a new string resulting from replacing all occurrences of the specified oldChar with the specified newChar.
Example: "hello".replace('l', 'w') returns "hewwo".

split(String regex): Splits the string into an array of substrings based on the specified regular expression.
Example: "hello world".split(" ") returns ["hello", "world"].

substring(int beginIndex, int endIndex): Returns a new string that is a substring of the original string, starting from the specified beginIndex and ending at the endIndex (exclusive).
Example: "hello".substring(1, 3) returns "el".

toLowerCase(): Returns a new string with all characters converted to lowercase.
Example: "HELLO".toLowerCase() returns "hello".

toUpperCase(): Returns a new string with all characters converted to uppercase.
Example: "hello".toUpperCase() returns "HELLO".

== operator checks if two string references point to the same memory location.

equals method compares the actual content of the strings, checking if they contain the same sequence of characters.

StringBuilder and StringBuffer in Java

- StringBuilder and StringBuffer are classes that provide mutable sequences of characters.
- They are designed for efficient string manipulation operations, such as appending, inserting, or deleting characters from a string.
- The main difference between StringBuilder and StringBuffer is that StringBuilder is not thread-safe, while StringBuffer is thread-safe.
- StringBuilder is faster and more efficient in single-threaded environments, while StringBuffer is safer to use in multi-threaded environments.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb.toString()); // Output: Hello World
```

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
```

TheTestingAcademy

System.out.println(sb.toString()); // Output: Hello World

StringBuffer and StringBuilder

- equals() is not overriden in StringBuilder and StringBuffer class.
- hashCode() is not overriden in the StringBuilder and StringBuffer class.
-

Table summarizing the most commonly used functions of StringBuilder and StringBuffer classes in Java:

Function	Description
`append()`	Appends the specified character(s) or object to the end of the string.
`insert()`	Inserts the specified character(s) or object at the specified index in the string.
`delete()`	Deletes the characters from the string between the specified start and end indices.
`deleteCharAt()`	Deletes the character at the specified index in the string.
`replace()`	Replaces the characters in the string between the specified start and end indices with the specified string.
`reverse()`	Reverses the order of the characters in the string.

`substring()`	Returns a substring of the string between the specified start and end indices.
`charAt()`	Returns the character at the specified index in the string.
`length()`	Returns the length of the string.
`setCharAt()`	Sets the character at the specified index in the string to the specified character.
`capacity()`	Returns the current capacity of the string builder.
`ensureCapacity()`	Ensures that the capacity of the string builder is at least the specified minimum capacity.
`trimToSize()`	Trims the capacity of the string builder to its current length.

Note that **StringBuilder** and StringBuffer share most of their functions and have very similar syntax. The main difference between them is that StringBuilder is not thread-safe, while StringBuffer is thread-safe.

[Assignment] String Problems

Reverse a String
Input - s = abc
O/P- s = cba

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

        String str= "Pramod", nstr="";
        char ch;

      System.out.print("Original word: ");
      System.out.println("Pramod"); //Example word

      for (int i=0; i<str.length(); i++)
      {
        ch= str.charAt(i); //extracts each character
        nstr= ch+nstr; //adds each character in front of the existing
string
      }
      System.out.println("Reversed word: "+ nstr);
    }
```

Palindrome

```
Input : str = "abba"
Output: Yes

Input : str = "pramod"
Output: No
```

```
public static boolean isPalindrome(String str)
    {
        // Initializing an empty string to store the reverse
        // of the original str
        String rev = "";

        // Initializing a new boolean variable for the
        // answer
        boolean ans = false;

        for (int i = str.length() - 1; i >= 0; i--) {
            rev = rev + str.charAt(i);
        }

        // Checking if both the strings are equal
        if (str.equals(rev)) {
            ans = true;
```

```
        }
        return ans;
    }
```

Using the String Builder

```
class ReverseString {
    public static void main(String[] args)
    {
        String input = "Pramod Sir is Owner of TheTestingAcademy";

        StringBuilder input1 = new StringBuilder();

        // append a string into StringBuilder input1
        input1.append(input);

        // reverse StringBuilder input1
        input1.reverse();

        // print reversed String
        System.out.println(input1);
    }
```

Reverse Words in String

Input: s = “i love programming very much”
Output: s = “much very programming love i”
// Java program to reverse a String

```
import java.util.*;
class TheTestingAcademy {

      // Reverse the letters
      // of the word
      static void reverse(char str[], int start, int end)
      {
            // Temporary variable
            // to store character
            char temp;

            while (start <= end) {
                  // Swapping the first
                  // and last character
                  temp = str[start];
                  str[start] = str[end];
```

```
                str[end] = temp;
                start++;
                end--;
        }
    }
    // Function to reverse words
    static char[] reverseWords(char[] s)
    {
        // Reversing individual words as
        // explained in the first step

        int start = 0;
        for (int end = 0; end < s.length; end++) {
            // If we see a space, we
            // reverse the previous
            // word (word between
            // the indexes start and end-1
            // i.e., s[start..end-1]
            if (s[end] == ' ') {
                reverse(s, start, end);
                start = end + 1;
            }
        }

        // Reverse the last word
        reverse(s, start, s.length - 1);

        // Reverse the entire String
        reverse(s, 0, s.length - 1);
        return s;
    }

    // Driver Code
    public static void main(String[] args)
    {
        String s = "i like this program very much ";

        // Function call
        char[] p = reverseWords(s.toCharArray());
        System.out.print(p);
    }}
```

🧵 Funtions in Java

a function is called a method. Methods in Java are blocks of code that perform a specific task and can be reused throughout your program

1. No Return Type
2. Return Type

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

int result = add(5, 3); // Calling the 'add' method with arguments 5 and 3

public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }

   public static void main(String[] args) {
      Calculator calculator = new Calculator();
      int result = calculator.add(5, 3); // Calling the 'add' method
      System.out.println("Result: " + result);
   }
}
```

```
static String appDutta(String s){
System.out.println("Return with Param");
return s+"Dutta";
}

static String appDutta2(){
System.out.println("Return and Non Param");
return "Dutta";
}

static void print(String s){
System.out.println("Non Return with Param");
System.out.println("Print some"+ s);
}
static void print(){
System.out.println("Non Return and Non Param");
}
```

TestingAcademy

📗 OOPs (Java)

1. OOPs vs POP or PPL (Procedural oriented programming)
2. Object vs Functional

Procedural Programming Language	Object Oriented Programming Language
1. Program is divided into functions.	1. Program is divide into classes and objects..
2. The emphasis is on doing things.	2. The emphasis on data.
3. Poor modeling to real world problems.	3. Strong modeling to real world problems.
4. It is not easy to maintain project if it is too complex.	4. It is easy to maintain project even if it is too complex.
5. Provides poor data security.	5. Provides strong data Security.
6. It is not extensible programming language.	6. It is highly extensible programming language.
7. Productivity is low.	7. Productivity is high.
8. Do not provide any support for new data types.	8. Provide support to new Data types.
9. Unit of programming is function.	9. Unit of programming is class.
10. Ex. Pascal , C , Basic , Fortran.	10. Ex. C++ , Java , Oracle.

PPL Details

Using the Global Struct to create a Complex Data Type and X global primitive

```
Struct Customer {

Int id,
String name
String age

}
```

Int x = 99;

- UpdateCustomer()
- FetchCustomer()
- Show() can also use the Customer and Primitive x (NO ENCAPSULATION)

- Data Security issues (global, local variables)

☯️ No Relation between the Struct(Global) and Functions.

How to Use Oops to Build Any Modern-Day Software

Consider this scenario: We are developing an Automation Tester Batch System.

1. Identify the objects
    a. Like Student, Course, Payment.
2. Describe those with details
    a. Data - Student -> name, id, age, address, email, course taken
    b. Operations -> addStudent, deleteStudent
    c. Bind them with Encapsulation (Class)
3. Establish the relationship with each other
    a. Student -> payment, Student -> Course, Course -> Payment
4. Now implement it via Class and Objects

Abstraction Example

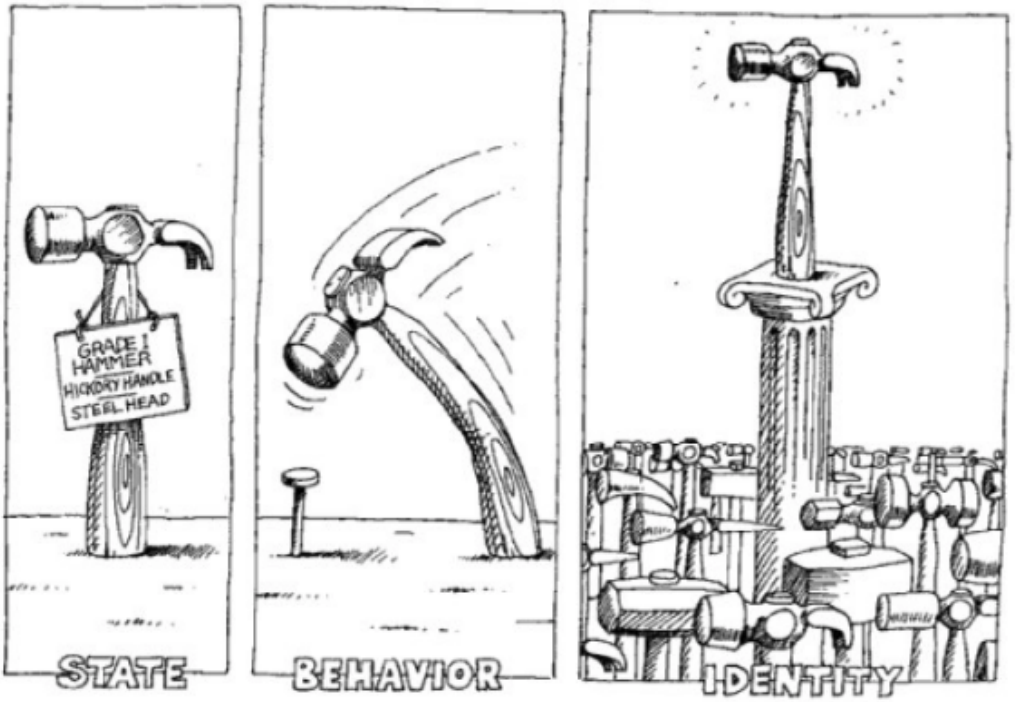
- It is a design level concept.
- Example - PC Computer
    a. User
    b. Service Engineer (Hide price)

Object Oriented Programming

- Objects are the main building blocks of OOPS i.e your applications will be divided into multiple objects.
- **A class with**
    - **Data members**
    - **Methods -**
- OOPs concepts apply now.
    - o Abstraction
    - o Encapsulation
    - o Inheritance
    - o Polymorphism
- Everything in the world is an object.

TheTestingAcademy

- **Grady Booch - Father of OOPS defined the object as follows:**



An object has state, exhibits some weii-defined behavior, and has a unique identity.

Abstract class

An abstract class, on the other hand, can contain both abstract methods (methods with no implementation) and concrete methods (methods with an implementation).

```
abstract class Shape {
  // fields (variables) and concrete methods can be defined in an
abstract class
  protected String color;

  public Shape(String color) {
    this.color = color;
  }

  // concrete method
  public String getColor() {
    return this.color;
  }

  // abstract method
  public abstract double getArea();
}
```

```
class Rectangle extends Shape {
  private double width;
  private double height;

  public Rectangle(double width, double height, String color) {
    super(color);
    this.width = width;
    this.height = height;
  }

  // implementation of the abstract method defined in the superclass
  @Override
  public double getArea() {
    return this.width * this.height;
  }
}

class Circle extends Shape {
  private double radius;

  public Circle(double radius, String color) {
    super(color);
    this.radius = radius;
  }

  // implementation of the abstract method defined in the superclass
  @Override
  public double getArea() {
    return Math.PI * this.radius * this.radius;
  }
}
```

In this example,

- the Shape class is an abstract class that defines an abstract method getArea() and a concrete method getColor().
- The Rectangle and Circle classes are concrete subclasses of Shape that must implement the getArea() method because it is abstract in the superclass.
- The Rectangle and Circle classes can also use the getColor() method inherited from the Shape class.

[Assignment]
20+ Examples of Abstract Class and Interface (in Demo)

Interface
An interface is a collection of abstract methods that must be implemented by any class that implements the interface

Inheritance

1. Example with Student -> Manual , Automation
2. Extra Data members, Operations or Methods

PolyMorphism

- + operator
- Remote or on/off button poly behavior

Class and Objects

Some Questions for Revision :

Question	Answer
Q1) What is the purpose of any programming model?	The purpose of a programming model is to provide a systematic approach to develop computer programs. It defines the rules, principles, and guidelines to be followed while designing and implementing software solutions. It also helps programmers to organize their code and create efficient, reliable, and maintainable applications.
Q2) What are the programming models available to develop applications?	The programming models available to develop applications include procedure-oriented programming (POP), object-oriented programming (OOP), functional programming (FP), event-driven programming (EDP), and aspect-oriented programming (AOP).

gAcademy

Q3) What are the main components of the Procedure Oriented Programming model?	The main components of the Procedure Oriented Programming model include procedures or functions, which are a set of instructions that perform a specific task, and data structures, which are used to organize and store data. POP follows a top-down approach, which means the program flow starts from the main function and then calls other functions or procedures as required.
Q4) What are the limitations of the Procedure Oriented Programming model?	The limitations of the Procedure Oriented Programming model include a lack of data security, difficulty in maintaining and modifying code, and low reusability of code. POP is also not suitable for large-scale applications as the program logic becomes complex and difficult to manage.
Q5) What is OOPs concept?	OOPs (Object-Oriented Programming) is a programming concept that emphasizes the use of objects, which are instances of classes, to represent real-world entities and their behavior. OOPs focuses on encapsulation, inheritance, and polymorphism to create reusable and modular code.
Q6) What are the main components of Object-Oriented Programming model?	The main components of Object-Oriented Programming model include classes, objects, encapsulation, inheritance, and polymorphism. Classes are used to define the attributes and behavior of objects, and objects are instances of classes that represent real-world entities. Encapsulation provides data security by hiding the implementation details of classes, while inheritance allows the creation of new classes by extending existing classes. Polymorphism enables the use of the same code for different types of objects.
Q7) Who is the father of OOPs?	Alan Kay is considered the father of OOPs, as he introduced the concept of object-oriented programming in the 1960s.

Q8) What are the major principles of Object-Oriented Programming?	The major principles of Object-Oriented Programming include inheritance, encapsulation, abstraction, and polymorphism. Inheritance allows the creation of new classes by extending existing classes, encapsulation provides data security by hiding the implementation details of classes, abstraction allows the creation of complex systems by hiding unnecessary details, and polymorphism enables the use of the same code for different types of objects.
Q9) What is abstraction?	Abstraction is a programming concept that involves hiding unnecessary details and exposing only essential information. It allows the creation of complex systems by breaking them down into smaller, manageable parts. In OOPs, abstraction is achieved through the use of abstract classes and interfaces.
Q10) What is encapsulation?	Encapsulation is a programming concept that involves hiding the implementation details of classes and exposing only their interfaces or public methods. It provides data security by preventing unauthorized access to data and allows for easy modification of code without affecting other parts of the program.
Q11) What is inheritance?	Inheritance is a programming concept that allows the creation of new classes by extending existing classes. It enables code reuse and promotes the creation of modular and scalable applications

Assignment - What happens if we do it

```
Class Book{
     Int id;
    String name;
    String author;

}
```

```
Book s1 = new Book();
```

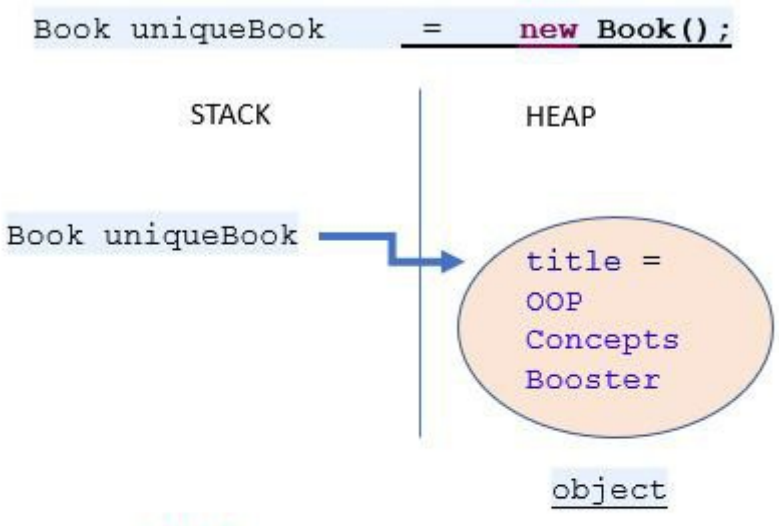
1. JVM allocates 8 bytes for each reference variable.
2. All variables are assigned a memory.
3. Referq

Abstract classes and methods

- Override the methods if the child extends the Parent or adds itself as Abstract.
- No object for Abstract, only reference.

Interfaces

1. Full Abstract - no implantation of any method apart from default.
2. Public static final variables
3. Public abstract methods.




```
class Book{
String title;
Book(){
title = "OOP Concepts Booster";
}
}
```

ThingAcademy

Important Class Members

1. Variables
2. Blocks
3. Constructors
4. Methods
5. Inner Class

```
package atb.classdemo;

public class Book {
    // Instance Variables
    int id; // -> Instance Primitive variable
    String name; // -> Instance Reference variable
    String author;

    // Block
    {
        System.out.println("I am block");
    }

    // Default constructor
    Book(){
        System.out.println("Default Con");
    }

    // Parameters constructor
    public Book(int id, String name, String author) {
        this.id = id;
        this.name = name;
        this.author = author;
    }

    // Method
    void show(){
        System.out.println("Data");
    }

    @Override
    public String toString() {
```

```
        return "Book{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", author='" + author + '\'' +
                '}';
    }
```

```
    // Inner Classe
    class Publisher{
        String name;
        void show(){
            System.out.println(Book.this.name);
        }
    }

}
```

Type of Variables

1. Instance Variable
    a. Primitive variable
    b. Reference variable
2. Local Variables
3. Static Variables

```
package atb.classdemo.staticdemo;

public class StaticDemo {
    int a = 10;
    static  int b= 20;
    void m1(){
        System.out.println(a);
        System.out.println(b);
    }
    static void m2(){
        //System.out.println(a);
        System.out.println(b);
    }
}
```

Static members can be accessed with Class Name.
Instance members can't be accessed with class Name.

```java
package atb.classdemo.staticdemo;

public class Main {
    public static void main(String[] args) {
        StaticDemo.m2(); // How?
    }
}
```

Class Loaders

Moving the file from src file to memory
Class loaders are responsible for loading Java classes dynamically to the JVM (Java Virtual Machine) during runtime.

TestingAcademy

Encapsulation

- Refers to the bundling of data and methods that operate on that data within a single unit, or object.
- Encapsulation helps to promote the principle of "data hiding".
- This is achieved by declaring the **object's data fields as private** and providing public getter and setter methods to access and modify the data.

```java
public class Student {
  // data fields are private
  private String name;
  private int age;
  private String address;
  // public getter and setter methods for each field
  public String getName() {
    return this.name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return this.age;
  }
```

```
  public void setAge(int age) {
    this.age = age;
  }
  public String getAddress() {
    return this.address;
  }

  public void setAddress(String address) {
    this.address = address;
  }
}
```

Access Modifier

	default	private	protected	public
same class	yes	yes	yes	yes
same package subclass	yes	no	yes	yes
same package non-subclass	yes	no	yes	yes
different package subclass	no	no	yes	yes
different package non-subclass	no	no	no	yes

The

Access modifiers in Java:

Access Modifiers

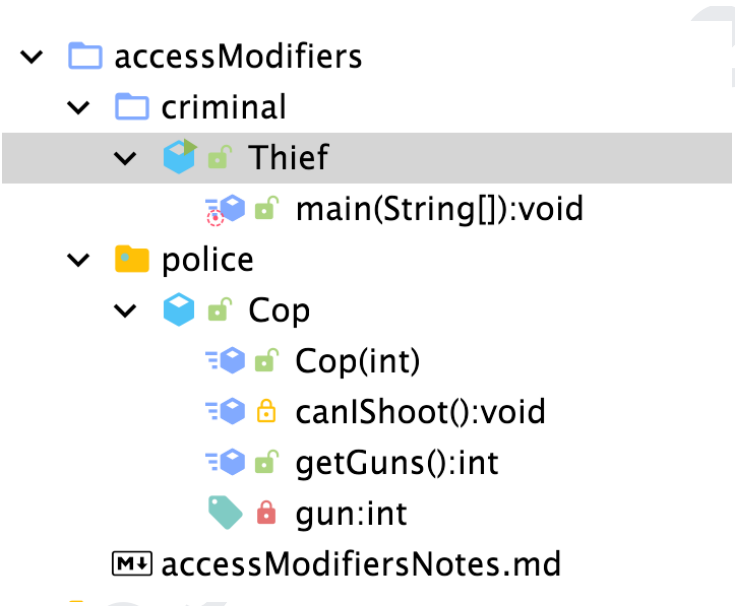
They are used to impose access restrictions on different data members and member functions.

Three types of access modifiers in Java

- Private : A private member cannot be accessed directly from outside the class
- Public : members can be directly accessed by anything which is in the same scope as the class object.
- Protected :access level to the protected members lies somewhere between private and public. protected data members can be accessed inside a Java package
- Default : The default access is similar to the protected. It also has package-level access

Example of Cop and Police

Private Gun, public CheckGun, Shoot is Private or Protected!



Polymorphism

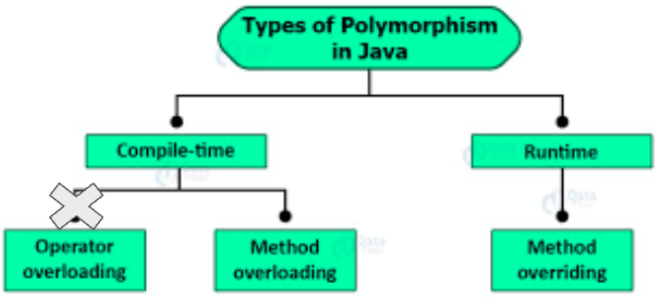
TheTe

Polymorphism

Polymorphism is the ability of an object to take on many forms

Each of these classes will have different underlying data. Precisely, Poly means 'many' and morphism means 'forms'.

when a parent class reference is used to refer to a child class object



Polymorphism

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

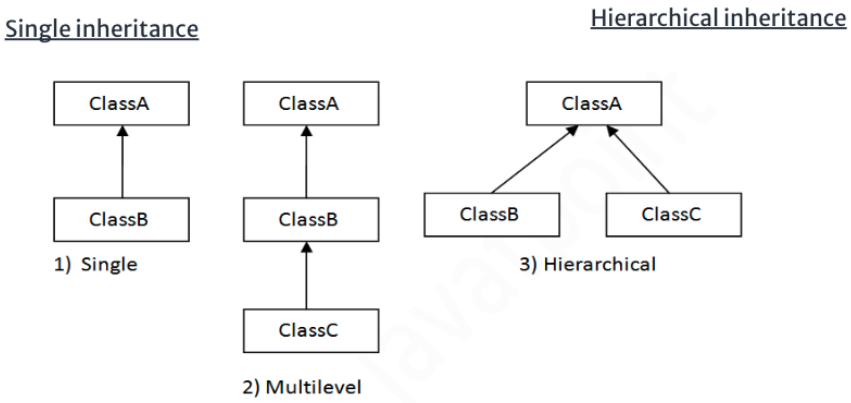
Method overloading	Method overriding
1. The process of defining functions having the same name with different parameter list is called as **overloading method**.	1. When methods of the subclass having same name as that of superclass, overrides the methods of the superclass then it is called as **overriding methods**.
2. It is a relationship between methods in the same class.	2. It is a relationship between subclass method and a superclass method.
3. It is static binding –at the compile time.	3. It is dynamic binding—at the runtime.
4. It is the concept of **compile time polymorphism or early binding.**	4. It is the concept of **run-time polymorphism or late binding.**
5. It may or may not be observed during inheritance.	5. It is observed during inheritance.
6. Return types do not affect overloading.	6. Return types, method names and signatures, of both overridden methods must be identical.

Assignment

1. Create the Class of ATB
2. Create an Array of ATB Students and add toString method.
3. Create Single, Multilevel, and Heriichca. Inteheri
4. Create both Overloading and OverRIDING Examples

Inheritance

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.



Abstraction

Abstract classes and methods

- In Java **Abstract is the keyword.**
- Remember, **abstract can be a class or method both.**
- If the class doesn't have any abstract method, then it is a concrete class.
- Abstract class cannot be final.
- An abstract class cannot be instantiated i.e. one cannot create an object of an abstract class.
- Although it is not required, an abstract class may have the declaration of one or more abstract methods.
- Since the body of an abstract method cannot be implemented in an abstract class.
- Non-abstract/normal methods can be implemented in an abstract class.
- To use an abstract class it needs to be inherited from.
- The class which inherits from the abstract class must implement all the abstract methods declared in the parent abstract class.
- An abstract class can have everything else as same as a normal Java class has i.e. constructor, static variables and methods.
- You cannot use the following modifier with abstract methods:
  Private, static, final, strictfp, synchronized, native.

Interface

1. Interface variables will be inherited to subclasses.

2. interface is a keyword which is used to define User Defined Datatypes.
3. Interfaces can be used to achieve multiple inheritance in Java.
4. One interface can extend one or more interfaces.
5. One class can implement one or more interfaces.
6. Interface can contain the following members:
    a. public final static variables
    b. public abstract methods
    c. public static inner classes
7. Variables declared in the interface are by default public final and static.
8. Methods declared in the interface are by default public and abstract.

Default Methods in interfaces

- Default methods are methods that can have a body.
- They provide additional functionality to a given type without breaking down the implementing classes.
- If a new method was introduced in an interface then all the implementing classes used to break.

```
default return type name () {}
```

- Solve the Diamond problem by Interface.super.methodName

Static methods in interfaces

static methods in interfaces are similar to default methods but the only difference is that you can't override them.

Problem to discuss to check Interface

```
interface I1{}
interface I2{}
class A{}
class B{}Interface
class Test1 extends A{ } //0K
class Test2 extends A,B{ } //Not OK
class Test3 implements I1{ }//0K
class Test4 implements I1,I2{ }//0K
class Test5 extends A implements I1,I2{ }//OK
class Test6 implements I1 extends A{ } / /Not OK


interface I3 extends A{}
interface I4 implements A{}
interface I5 extends A,B{}
```

TestingAcademy

```
interface I6 extends I1,I2{ }
```

Interfaces	Abstract Classes
Support multiple inheritance	Don't support multiple inheritance
All members are `public`	Can have `private`, `protected` and `public` members
All data members are `static` and `final`	Can have non-static and non-final members too
Can't have constructors	Constructors can be defined

✅ Problem Statement

Book class which has an abstract method getDetails(),name, author, price.
PrintMyBook class that inherits from the Book class.

asbtract

Book myBook = new PrintMyBook("Harry Potter", "J.k. Rowling", "120");

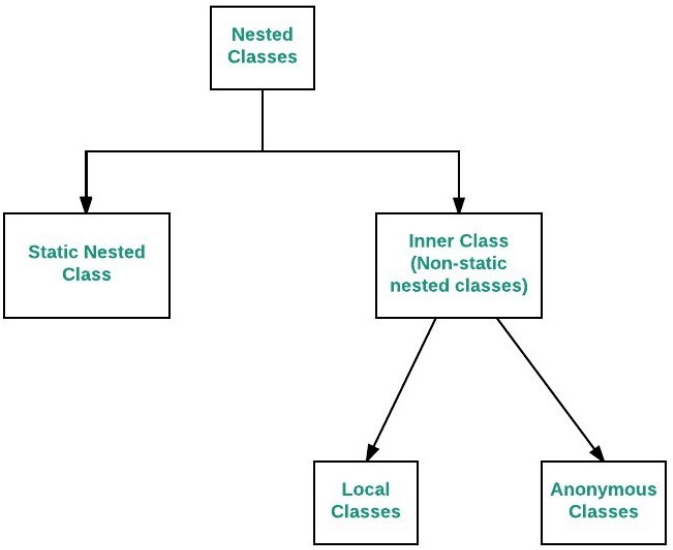
// Output
"Harry Potter, J.k. Rowling, 100"

Nested Class

Class within Class is called Nested class

```
class OuterClass
{
...
   class NestedClass
   {
      ...
   }
}
```

TheTestingAcade



Static Nested Class

- Only a static member of the outer class can be accessed directly.
- Inside static nested class, static members can be declared.

```
class OO{
  static  int o =100;
  int a = 900;
  static class SNCI{
     void show(){
        System.out.println(o);
        //Only a static member of outer class can be accessed directly.
       // System.out.println(a);
     }
  }
}

public class SNC {

  public static void main(String[] args) {

     OO.SNCI oo = new OO.SNCI();
     oo.show();

  }
}
```

TheTestingAcademy

Inner Class

An inner class in Java is a class that is defined within another class. Inner classes are useful for organizing code and for creating classes that are tightly coupled to their outer class.

1. Inner classes can use the outer data and members.
2. Inner class members can't be used directly, use Inner class object.
3. Inner class use outside with
   `Outer.Inner io2 = new Outer().new Inner();`

4. Static declaration is not allowed in the inner class
5. Static Inner class, Instance Inner class, Local Inner class, Anonymous inner class

There are 4 ways you can create a Class

1. Inner Class
2. Method Level Inner Class
3. Nested Class
4. Anonymous Class

```java
public class Car {
  // outer class field
  private String make;
  private String model;

  // inner class
  public class Engine {
    // inner class fields
    private int horsepower;
    private String fuelType;

    // inner class constructor
    public Engine(int horsepower, String fuelType) {
      this.horsepower = horsepower;
      this.fuelType = fuelType;
    }

    // inner class method
    public void start() {
      System.out.println("Engine starting...");
    }
  }

  // outer class constructor
  public Car(String make, String model) {
    this.make = make;
    this.model = model;
```

```
  }

  // outer class method
  public void drive() {
    System.out.println("Driving the car...");
  }
}

//
Car myCar = new Car("Toyota", "Camry");
Car.Engine engine = myCar.new Engine(200, "Gasoline");
engine.start();
```

Upcasting and Downcasting in Class

You can assign a subclass object to the super type reference variable directly. This process is called UPCASTING.

```
class Parent{}
class Child extends Parent{}

Parent parent = new Parent();  // p1


Child child = new Child();
parent=child;
```

// Valid -> Upcasting

```
class Hello{}
class Child extends Hello{}

Hello hello = new Child();
Child hai=hello; // Invalid
Child hai= (Child)hello; // Valid ->Downcasting
```

Anonymous Inner class

It is an inner class without a name and for which only a single object is created.

An **anonymous inner class** can be useful when making an instance of an object with certain “extras” such as overriding methods of a class or interface, without having to actually subclass a class.

```
Test t = new Test()
{
  // data members and methods
  public void test_method()
  {
    .....
    .....
   }
}
```

- An anonymous class has access to the members of its enclosing class.
- An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final.
- constructors can not be declared in an anonymous class.
-

Normal Java Class	Anonymous Class
1. It can extend only one class at a time. 2. It can implement any number of interfaces simultaneously. 3. It can extend a class & can implement any no. of interfaces simultaneously. 4. Here we can write any number of constructors. 	It also can extend only one class at a time. It can implement only one Interface at a time. It can extend a class or can implement an interface, but not both simultaneously. Here we can't write any constructor because explicitly because name of class & constructor must be same but anonymous Inner classes don't have any name.

Types of Anonymous Inner Class

1. Anonymous Inner class that extends a class
2. Anonymous Inner class that implements an interface
3. Anonymous Inner class that defines inside method/constructor argument

Anonymous inner classes are generic created via below listed two ways as follows:

1. Class (may be abstract or concrete)
2. Interface

```
package thetestingacademy.oops.anonymousclass;

public class Anony02 {
   public static void main(String[] args) {
//            MyStudent myStudent = new MyStudent();
//            myStudent.setId();

        Student student = new Student() {
            @Override
            public void setId() {
                System.out.println(id);
            }
        };
        student.setId();
   }
}


interface Student{
   int id = 11;
   void setId();
}

class MyStudent implements Student{

   @Override
   public void setId() {
        System.out.println(id);
   }
}
```

Wrapper Classes

- A Wrapper class is a class whose object wraps or contains primitive data types.
- They convert primitive data types into objects.
- Data structures in the **Collection framework**, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
- An object is needed to support synchronization in multithreading.

TheTestingAcademy

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Primitive -> Wrapper , Primitive -> String ,Wrapper -> Primitive

1. Primitive to String
    1. Using valueOf()
    2. toString() of wrapper
2. String to Primitive
    a. A) using parseXO;
3. Primitive to Wrapper Object
    a. A) Using Constructor of Wrapper class
    b. B) Using valueOf method of Wrapper class
4. Wrapper Object to Primitive
    a. A) Using xxValue0 method of Wrapper classes
5. String to Wrapper Object
    a. A) Using constructor of Wrapper classes
    b. B) Using valueOf method of Wrapper classes
6. Wrapper Object to String
    a. A) Using toString(method

Comparison of Autoboxed Integer

```
Integer x = 400, y = 400;
      if (x == y)
        System.out.println("Same");
      else
        System.out.println("Not Same");

// Les than 127

Integer x = 40, y = 40;
      if (x == y)
        System.out.println("Same");
```

TheTestingAcademy

```
        else
          System.out.println("Not Same");
```

// JVM - values from -128 to 127 are cached, so the same objects are returned.

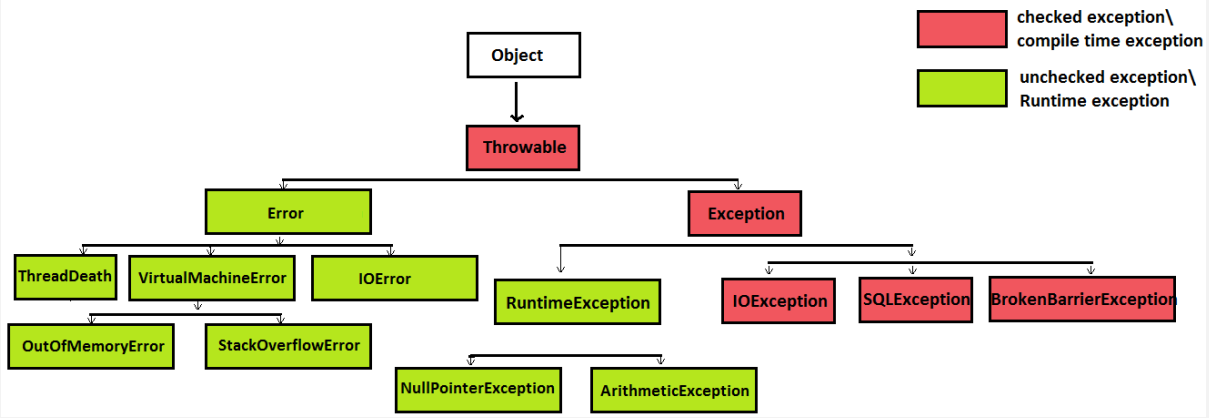
```
Integer x = new Integer(40), y = new Integer(40);
        if (x == y)
          System.out.println("Same");
        else
          System.out.println("Not Same");


Integer X = new Integer(10);
    Integer Y = 10;

    // Due to auto-boxing, a new Wrapper object
    // is created which is pointed by Y
    System.out.println(X == Y);
```

📕 Exceptions

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

P.S. - If no exceptions are handled, control is passed to the JVM, which terminates the program.

Exception Hierarchy



TestingAcademy

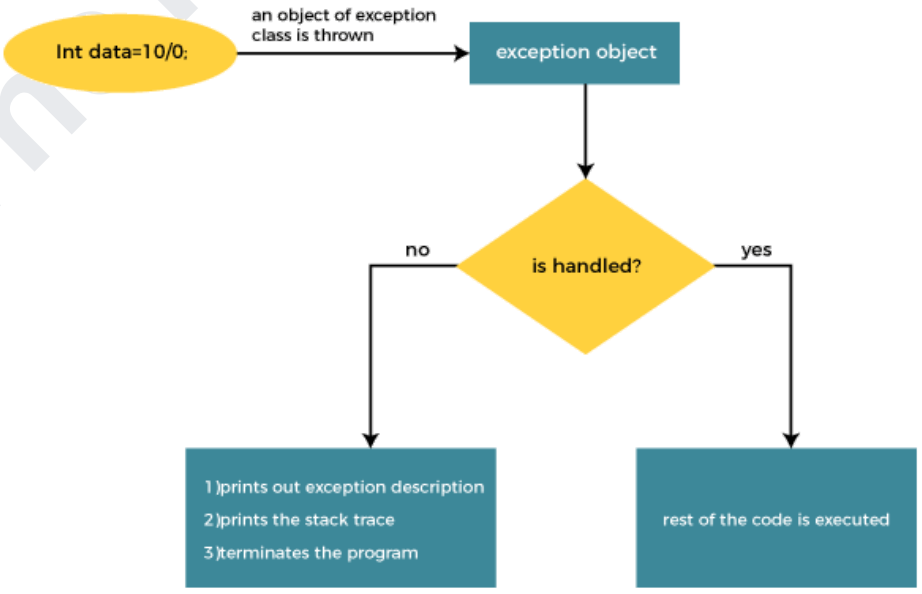
```java
package thetestingacademy.exceptions;  // 0

public class Ex01 {
    // 3 PROBLEMs in this problem
    public static void main(String[] args) {
        String sh = args[0];
        int x = Integer.parseInt(sh);
        int a = 10/x;
        System.out.println(x);
        System.out.println(a);
    }
}
```

Solve the three problems in the above program.

1. Array.
2. Number format
3. Arithmetic.

Exception Handling in Java is a mechanism for handling runtime errors so that the application's normal flow can be maintained.

Try and Catch
Attempt to catch the block.



TheTestingAcademy

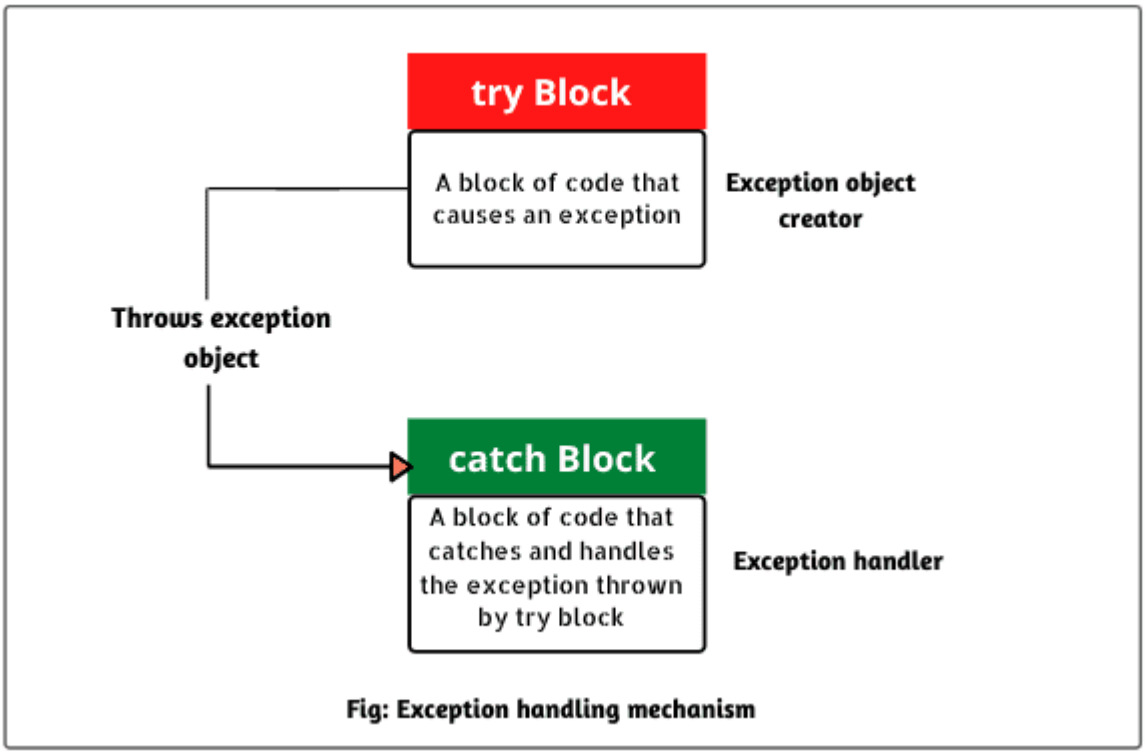


Fig: Exception handling mechanism

When Catch will be executed
Only when an exception in public class Ex02 is raised

```
    public static void main(String[] args) {
        try {
            String name = null;
            name.trim();
        }catch (Exception e){
            e.printStackTrace();
        }
        System.out.println("I am done");
    }
}
```

Flow of the Exception Program

1. JVM calls Main Method.
2. If No Exception Execute and Skips Catch
3. If Exception, It created an Object of Exceptions.
4. Continue

⚠️Questions

1. Throwable in Catch Block
2. String in Catch ?

```
    public static void main(String[] args) {
        try {
            String name = null;
            name.trim();
        }catch (Exception e){
//        }catch (Throwable e){
//        }catch (Error e){
            e.printStackTrace();
        }
        System.out.println("I am done");
    }
}
```

Multiple Catch

You can catch multiple exceptions; their execution depends on the execution problem.

```
package thetestingacademy.exceptions;

public class Ex03 {

    public static void main(String[] args) {
        try {
            String ip = args[0];
            int a = Integer.parseInt(args[0]);
            int b = 10 / a;
        } catch (NumberFormatException exception) {
            exception.printStackTrace();
        } catch (ArithmeticException exception) {
            exception.printStackTrace();
        } catch (ArrayIndexOutOfBoundsException exception) {
```

```
            exception.printStackTrace();
        }
    }
}
```

Matching Catch it executes
If NumberFormat -> Number Ex
If Arth - Arth
If Array -> Array
// Keep the Biggest at the last Exception E (bigger basket)
⚠️ P.S. - Please do not write anything between Try and Catch.

Please find the problem in this Code.

```
package thetestingacademy.exceptions;

public class Ex04 {
    public static void main(String[] args) {
        try {
            int a = Integer.parseInt(args[0]);
        }
        catch (NumberFormatException exception) {
            exception.printStackTrace();
        }
        try {
            int b = 10 / a;
        }
        catch (ArithmeticException exception) {
            exception.printStackTrace();
        }
        try {
            String ip = args[0];
        }
        catch (ArrayIndexOutOfBoundsException exception) {
            exception.printStackTrace();
        }
    }
}
```

Or Catch Concept
After Java 7, We can write them in OR condition with pipe char |

```
try {
            String ip = args[0];
            int a = Integer.parseInt(args[0]);
            int b = 10 / a;
```

```
        } catch (NumberFormatException  | ArithmeticException |
ArrayIndexOutOfBoundsException exception) {
            exception.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Finally Block

The finally block is a block of code that follows a try-catch block and is always executed, regardless of whether an exception is thrown or caught. The finally block is typically used to release resources or perform other cleanup tasks.

```
try {
   // code that might throw an exception
} catch (ExceptionType e) {
   // code to handle the exception
} finally {
   // code to be executed after the try-catch block
}
```

```
public class Main {
   public static void main(String[] args) {
      FileInputStream file = null;
      try {
         file = new FileInputStream("file.txt");
         // read from the file
      } catch (FileNotFoundException e) {
         System.out.println("File not found!");
      } finally {
                        file.close();
                }
   }
}
```

If the file cannot be found, a FileNotFoundException is thrown, which is caught by the catch block.

Regardless of whether the exception is thrown or not, the finally block is always executed, and it closes the file.

Attempting to catch in the final is also permitted.

Yes

```
public static void main(String[] args) {
      FileInputStream file = null;
      try {
         file = new FileInputStream("file.txt");
         // read from the file
      } catch (FileNotFoundException e) {
         System.out.println("File not found!");
      } finally {
         if (file != null) {
            try {
               file.close();
            } catch (IOException e) {
               // handle the exception
            }
         }
      }
   }
```

Only in one case, If exit(0) is found, finally will not execute.

Problem 1

```
public class Ex06 {
    public static void main(String[] args) {
        ProblemF problemF = new ProblemF();
        int x = problemF.show();
        System.out.println(x);
    }
}

class ProblemF{
    int a = 10;
    int show(){
        try{
            System.out.println("In class -> "+ a);
            return a;
```

eTestingAcademy

```
        }catch (Exception e){
            System.out.println("Catch");
            a = 20;
            return a;
        }finally {
            System.out.println("I am Final");
        }
    }}
```

Problem #2

```
package thetestingacademy.exceptions;

public class Ex06 {
    public static void main(String[] args) {
        ProblemF problemF = new ProblemF();
        int x = problemF.show();
        System.out.println(x);
    }
}

class ProblemF{
    int a = 10;
    int show(){
        try{
//            a =  10/0;
            System.out.println("In class -> "+ a);
            return a;
        }catch (Exception e){
            System.out.println("Catch");
            a = 20;
            return a;
        }finally {
            System.out.println("I am Final");
            //int[] a1 = new int[-1]; fix it
        }
    }
}
```

Problem #03

```
package thetestingacademy.exceptions;

public class Ex08 {
    public static void main(String[] args) {
        try {
            int a = 10 / 10;
        } catch (Exception a) {
            System.out.println("I am Catch 01");
        } finally {
            System.out.println("Finally 01");
        }
        try {
            int a = 10 / 0;
        } catch (Exception a) {
            System.out.println("I am Catch 02");
        } finally {
            System.out.println("Finally 02");
        }
        try {
            int a = 10 / 0;
        } catch (Exception a) {
            System.out.println("I am Catch 03");
            System.exit(-1);
        } finally {
            System.out.println("Finally 03");
        }
        try {
            int a = 10 / 20;
        } catch (Exception a) {
            System.out.println("I am Catch 04");
        } finally {
            System.out.println("Finally 04");
        }
    }
}
```

Exception Passing Methods

Navigating the Exception Flows from One Method to OTHER

```java
package thetestingacademy.exceptions;

public class Exception02 {
    public static void main(String[] args) {
        extracted2();
        System.out.println("MAIN Execute");
    }

    private static void extracted2() {
        extracted1();
        System.out.println("extracted2 Execute");
    }

    private static void extracted1() {
        extracted();
        System.out.println("extracted1 Execute");
    }

    private static void extracted() {
        try {
//            String name = null;
//            name.length();

            Integer[] i = new Integer[2];
            System.out.println(i[3]);
        } catch (NullPointerException e) {
            System.out.println("Null");
            e.printStackTrace();
        }catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Out");
            e.printStackTrace();
        }catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Throw and Throws

- `Till Now JVM created an Exception for us.`
- `Now We will do it`

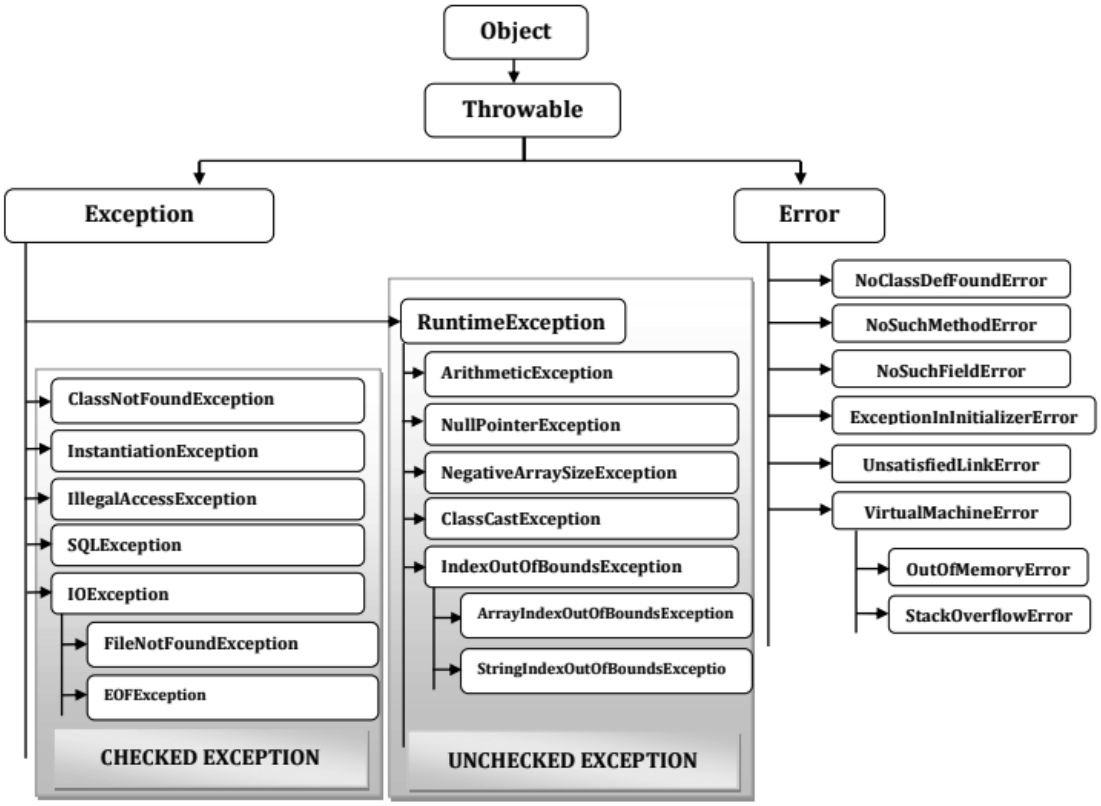
Throw exObject.
Throws Keyword at Method LEVEL

Custom Exception

"Exception" class or any of its subclasses, such as "RuntimeException". Custom exceptions allow you to create specific types of exceptions to represent error conditions that are specific to your application.

For example, you might create a custom exception to represent an error condition such as "insufficient funds" in a bank account application. This allows you to handle this error condition in a specific way, rather than using a more general exception type such as "Exception" or "RuntimeException".

TheTestingAcademy



Important Points

```
throw new Exception();
throw new Error();
throw new Throwable();

throw new Object(); //Not Ok

catch(Exception){}
catch(Error){}
catch(Throwable){}

catch(Object){} //Not Ok

void m1() throws
Exception
void m1() throws Error
void m1() throws
Throwable
```

```
void m1() throws Object   //Not Ok
```

Some Errors Asked in the Interview :
1) NoSuchMethodError:main

(if main() method is not found upto jdk1.6.)

2) NoClassDefFoundError

(if class name is wrong while executing.)

3) StackOverflowError

In recursive method call, if memory is not available in stack.)

4) OutOfMemoryError

(While creating the object, if there is no memory in heap.)
int arr[]=new int[54453453];

5) UnsupportedClassVersionError

(If JVM version is lower then compiler version.)

6) ExceptionInInitializerError
(if exception occurs at static variable or static block while loading the class.)
static int p;
static {p=Integer.parseInt("SRI");}
}

---

1) ClassCastException
(If you try to convert superclass object into subclass object whitout having the ref. of subclass.)

Object ob1 = new Object();
String st1=(String)ob1;

2) IllegalStateException

Thread t= new Thread();
t.start();

TheTestingAcademy

t.start();//Here this err will occur

3) IllegalArgumentException
t.setPriority(30);
Thread priority must be b/w 1-10.

Try Out

```
package thetestingacademy.exceptions;

public class Ex11 {
    public static void main(String[] args) {
        Hello h = new Hello();
        h.process();
        System.out.println("Pramod");
    }
}

class Hello {
    public int process() {
        throw new NullPointerException();
    }
}
```

TheTestingAcademy

Collection Framework

Focus on Main Business Logic rather than Low Level Logics

Problem with Arrays

- Since we have to give size, memory is wasted.
- Insertion and deletion are heavy.
- No built-in support for sorting, searching.

We prefer below classes and Interfaces in place of Arrays.

Legacy APIs are provided

Legacy Classes
- Vector
- Stack
- Properties
- Hash table
- Dictionary

Legacy Interface
- Enumeration (cursor)

Enumeration

It is an interface used to get elements of legacy collections(Vector, Hashtable)
Vector access with Enumeration
Vector v - new Vector();
Enumeration e = v.elements(); and hasNext functions

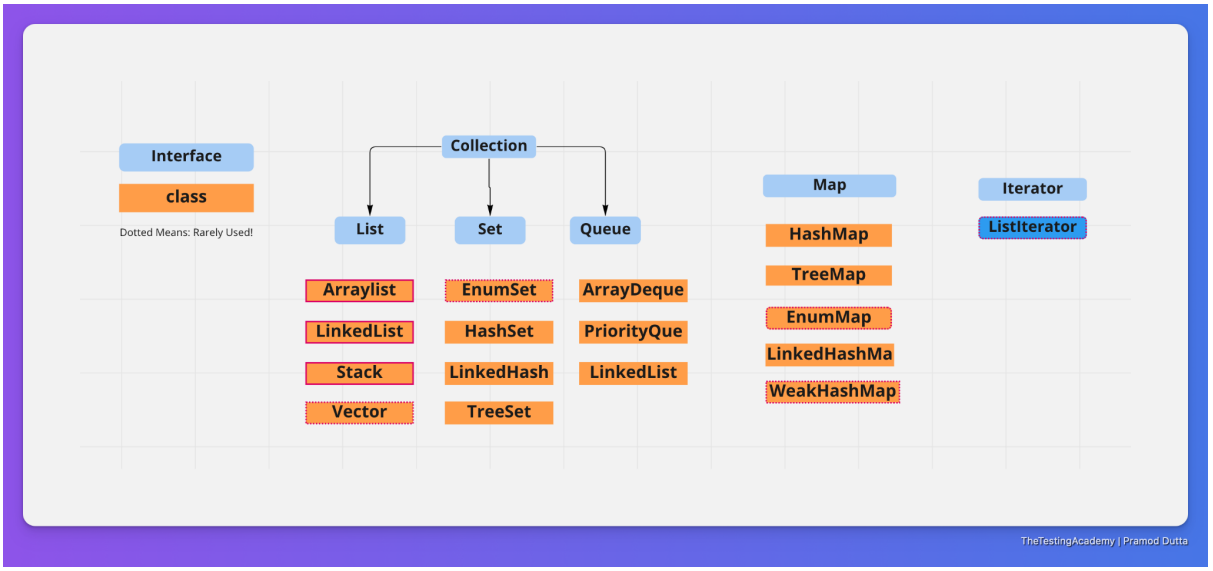
Enumeration	Iterator
Introduced in Java 1.0	Introduced in Java 1.2
Legacy Interface	Not Legacy Interface
It is used to iterate only Legacy Collection classes.	We can use it for any Collection class.
It supports only READ operation.	It supports both READ and DELETE operations.
It's not Universal Cursor.	It is a Universal Cursor.
Lengthy Method names.	Simple and easy-to-use method names.

TheTestingAcademy | Pramod Dutta

TheTestingAcademy

Collection vs Collections

- Collection Framework" has been defined in JDK 1.2 which holds all the collection classes and interfaces in it.
- Collection interface (java.util.Collection) and Map interface (java.util.Map) are the two main "root" interfaces of Java collection classes.



What is a Framework?
A framework is a set of classes and interfaces which provide a ready-made architecture.

Advantages of the Collection Framework

- Reduces programming effort
- Increases program speed and quality
- Consistent API - add(), remove(), contains()

List

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html

TheTestingAc

The time complexity comparison is as follows:

	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)

- Ordered Collection
- Control over the insertion
- List allows Duplicate.

Way to Initialize

- List<String> fruites = List.of("orange","apple");
- ArrayList - Underline Array - Get element is easy, inserting, Delete is Costly.
- LinkedList - LinkedList
- Vector

Most of the time, programmers prefer ArrayList over Vector because ArrayList can be synchronized explicitly using Collections.synchronizedList.

List Functions

Use the javap java.util.List

1. public abstract int **size();**
2. public abstract boolean isEmpty();
3. public abstract boolean contains(java.lang.Object);
4. public abstract java.util.Iterator<E> iterator();
5. public abstract java.lang.Object[] toArray();
6. public abstract <T> T[] toArray(T[]);
7. public abstract boolean **add(E);**
8. public abstract boolean r**emove(java.lang.Object);**
9. public abstract boolean **containsAll**(java.util.Collection<?>);
10. public abstract boolean addAll(java.util.Collection<? extends E>);
11. public abstract boolean addAll(int, java.util.Collection<? extends E>);
12. public abstract boolean removeAll(java.util.Collection<?>);
13. public abstract boolean retainAll(java.util.Collection<?>);
14. public default void replaceAll(java.util.function.UnaryOperator<E>);

TheTestingAcademy

15. public default void sort(java.util.Comparator<? super E>);
16. public abstract void **clear();**
17. public abstract boolean equals(java.lang.Object);
18. public abstract int hashCode();
19. public abstract E **get(int);**
20. public abstract E **set(int, E);**
21. public abstract void **add(int, E);**
22. public abstract E remove(int);
23. public abstract int **indexOf**(java.lang.Object);
24. public abstract int **lastIndexOf**(java.lang.Object);
25. public abstract java.util.ListIterator<E> listIterator();
26. public abstract java.util.ListIterator<E> listIterator(int);
27. public abstract java.util.List<E> subList(int, int);
28. public default java.util.Spliterator<E> spliterator();

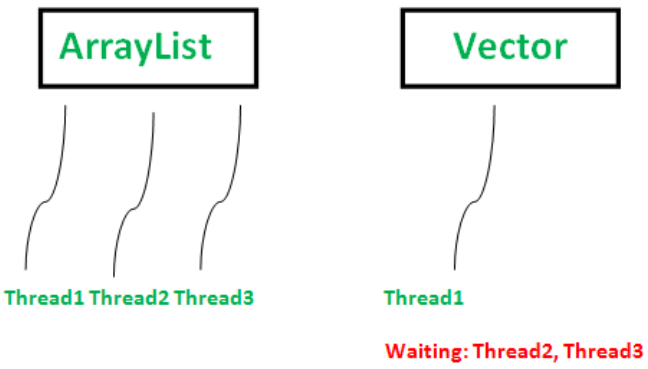
Arraylist

- ArrayList allows duplicates and it is implemented as a resizable array.
- ArrayList elements will be stored internally using indexing notation.
- This is one of the most widely used concrete class.
- It is fast to access the elements, but slow to insert and delete the elements.

Array vs Vector

Vector

- Vector allows duplicates, and it is implemented as a resizable array.
- Vector is legacy class and elements will be stored internally using indexing notation.
- Vector methods are synchronized, **so vector objects can't be accessed by multiple threads at a time.**



TheTestingAcademy

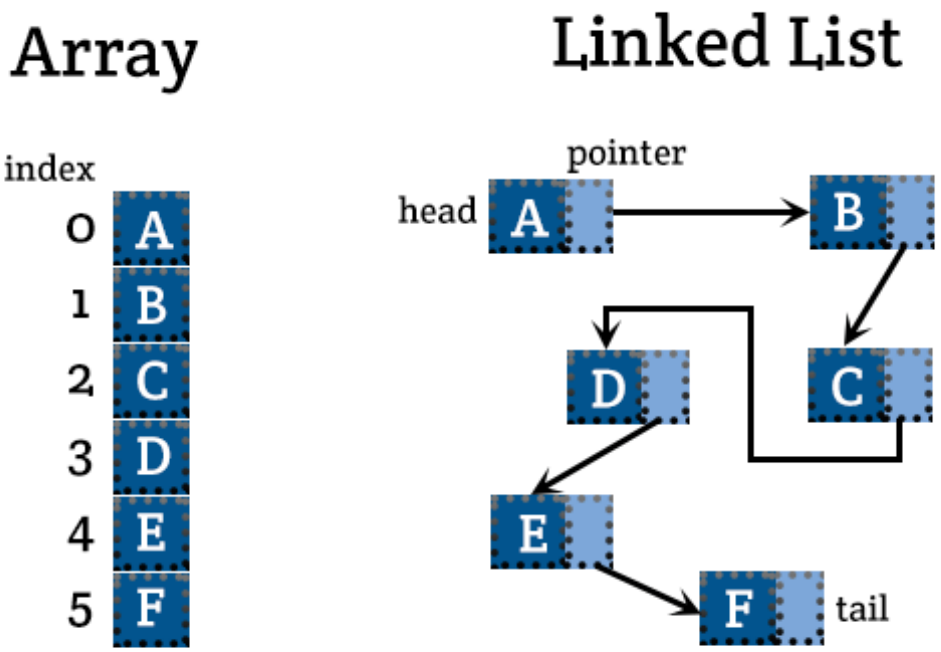
S. No.	ArrayList	Vector
1.	ArrayList is not synchronized.	Vector is synchronized.
2.	ArrayList increments 50% of the current array size if the number of elements exceeds ts capacity.	Vector increments 100% means doubles the array size if the total number of elements exceeds its capacity.
3.	ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a legacy class.
4.	ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in a runnable or non-runnable state until the current thread releases the lock of the object.
5.	ArrayList uses the Iterator interface to traverse the elements.	A Vector can use the Iterator interface or Enumeration interface to traverse the elements.

Array vs ArrayList

Actions on Array and ArrayList	**Array**	**ArrayList**
Declaration	String myList = new int[5];	ArrayList<String> myList = new ArrayList<String>();
Create new element	String a = new String("hello");	String a = new String("hello");
Add to list	myList[0] = a ;	myList.add(a);
Get size	myList.length ;	myList.size();
Get element from list	String s = myList[0];	Object o = myList.get(0);
Delete element	myList[0] = null;	myList.remove(0);

Check the given element is there or not	boolean isIn = false;
	
	for(String item : myList){
	If(a.equals(item){
	isIn = true;
	break;
	}
	}
		boolean isIn = myList.contains(a);

Array vs LinkedList



Difference between ArrayList and LinkedList in Java

ArrayList	LinkedList
ArrayList internally uses a dynamic array to store the elements	LinkedList internally uses a doubly linked list to store the elements
Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
ArrayList consumes less memory than LinkedList	A LinkedList consumes more memory than an ArrayList because it also stores the next and previous references along with the data.
An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
ArrayList is better for storing and accessing data.	LinkedList is **better for manipulating** data.

-

LinkedList

- LinkedList allows duplicates and internally implements a doubly-linked list data structure.
- LinkedList elements will be stored internally using node representation.
- It is fast to insert or delete the elements, but slow for accessing the elements.

Stack

- Stack is a legacy subclass of vector and its methods are synchronized, so stack objects can't be accessed by multiple threads at a time.
- Stack has important operations like peek(), pop(), push(), search() and other operations are same as Vector. It is a and not in use.

Task	ArrayList	LinkedList	Vector Stack
Order	User added	User added	User added
Duplicate Vaue	Yes	Yes	Yes
Null Value	Yes	Yes	Yes
Random Access	Yes	No	Yes
Sequential Access	Yes	Yes	Yes
Iterator (used?)	Yes	Yes	Yes
List Iterator	Yes	Yes	Yes
Enumeration	No	No	Yes
Index Representation	Yes	No	Yes
Node Representation	No	Yes	No
DeQue API can be used	No	Yes	No
Serializable	Yes	Yes	Yes
Cloneable	Yes	Yes	Yes

TheTestingAcademy

For Each vs Iterator

In for-each loop, we can't modify collection, it will throw a ConcurrentModificationException on the other hand with iterator we can modify collection.

If we have to modify the collection, we can use Iterator.

https://stackoverflow.com/questions/2113216/which-is-more-efficient-a-for-each-loop-or-an-iterator

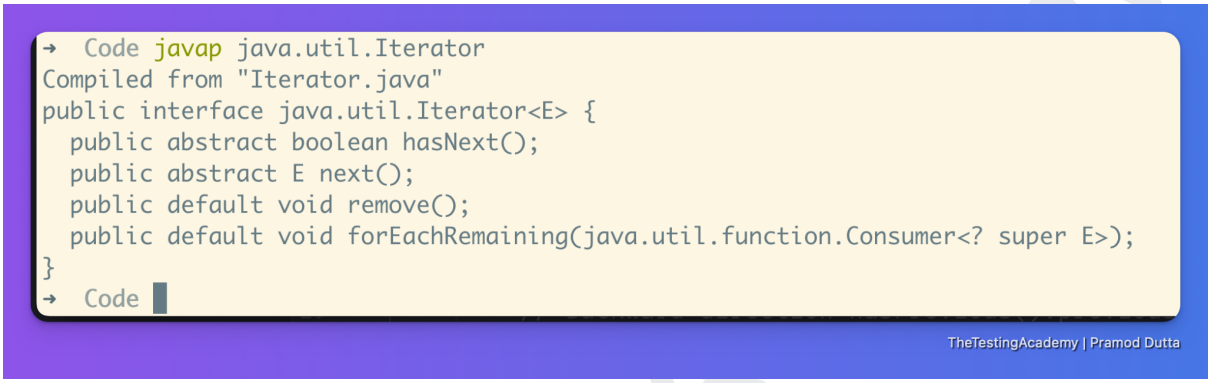
Iterator

- used to iterate or traverse or retrieve a Collection or Stream object's elements one by one and called a Java Cursor is an Iterator.

three cursors in Java.

1. Iterator
2. Enumeration
3. ListIterator

Methods in Iterator



Problem with Iterator

- In CRUD Operations, it does NOT support CREATE and UPDATE operations.
- only Forward direction.
- only Sequential iteration.

ListIterator

Iterator that is used to traverse all types of lists including ArrayList, Vector, LinkedList, Stack, etc. It is available since Java 1.2. It extends the iterator interface.

- bi-directional traversal.
- four CRUD operations(Create, Read, Update, Delete)
- Cursor concept.

TheTestingAcademy

Methods in ListIterator

```
➜  Code javap java.util.ListIterator
Compiled from "ListIterator.java"
public interface java.util.ListIterator<E> extends java.util.Iterator<E> {
  public abstract boolean hasNext();
  public abstract E next();
  public abstract boolean hasPrevious();
  public abstract E previous();
  public abstract int nextIndex();
  public abstract int previousIndex();
  public abstract void remove();
  public abstract void set(E);
  public abstract void add(E);
}
```

TheTestingAcademy | Pramod Dutta

Iterator vs **ListIterator**

Iterator	**ListIterator**
It can traverse a collection of any type.	It traverses only list collection implemented classes like LinkedList, ArrayList, etc.
Traversal can only be done in forwarding direction.	Traversal of elements can be done in both forward and backward direction.
Iterator object can be created by calling iterator() method of the collection interface.	ListIterator object can be created by calling directions listIterator() method of the collection interface.
Deletion of elements is not allowed.	Deletion of elements is allowed.
It throws **ConcurrentModificationException** on doing addition operation. Hence, addition is not allowed.	Addition of elements is allowed.
In iterator, we can't access the index of the traversed element.	In listIterator, we have nextIndex() and previousIndex() methods for accessing the indexes of the traversed or the next traversing element.
Modification of any element is not allowed.	Modification is allowed.

TheTestingAcademy | Pramod Dutta

Spliterator

- Spliterator == Splittable Iterator
- it can split some source, and it can iterate it too. (2 balanced)
- trySplit is for. Splitting is needed for parallel processing.


```java
package thetestingacademy.collections.list;

import java.util.ArrayList;
import java.util.Spliterator;
import java.util.stream.Stream;

public class SpliteratorDemo {

    public static void main(String[] args) {


        ArrayList<Integer> al = new ArrayList<>();
        // Add values to the array list.
        al.add(1);
        al.add(2);
        al.add(-3);
        al.add(-4);
        al.add(5);
        // getting Spliterator object on al
        Spliterator<Integer> splitr1 = al.spliterator();

        // estimateSize method
        System.out.println("estimate size : " +
splitr1.estimateSize());
        // getExactSizeIfKnown method
```

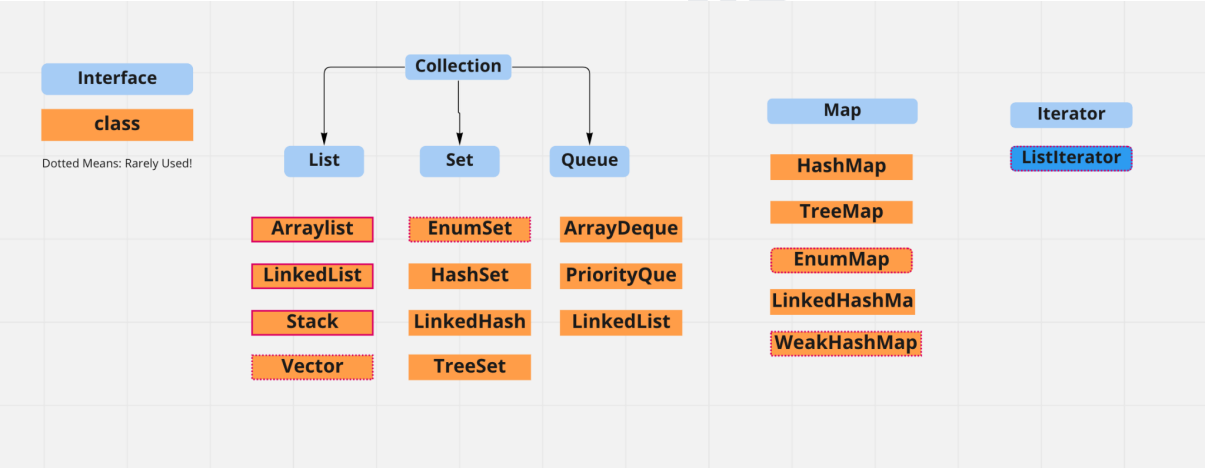
```
        System.out.println("exact size : " +
splitr1.getExactSizeIfKnown());


        // Split

        Spliterator<Integer> splitr1_split = splitr1.trySplit();
// If splitr1 could be split, use splitr2 first.
        if(splitr1_split != null) {
            System.out.println("Output from splitr2: ");
            splitr1_split.forEachRemaining((n) ->
System.out.println(n));
        }

        // Now, use the splitr
        System.out.println("\nOutput from splitr1: ");
        splitr1.forEachRemaining((n) -> System.out.println(n));



        ArrayList<Integer> tempList = new ArrayList<>();
        while(splitr1.tryAdvance((n) -> tempList.add(Math.abs(n))));
        Spliterator<Integer> splitr2 = tempList.spliterator();
        while(splitr2.tryAdvance((n) -> System.out.println(n)));


    }
}
```

ConcurrentModificationException

- Can’t add to the List while iterating.

Set

- Interface, java.util.Set
- Unique Only
- If o1.equals(o2) Only one will be in the List
- Set interface has the following concrete subclasses:
    - o HashSet
    - o LinkedHashSet
    - o TreeSet .



Check Implementation of Hash Set in Java doc

HashSet

- It is fast for searching and retrieving elements.
- It does not maintain any order for stored elements.> A, B, C

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
boolean	add(E e)	Adds the specified element to this set if it is not already present.
void	clear()	Removes all of the elements from this set.
Object	clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean	contains(Object o)	Returns true if this set contains the specified element.
boolean	isEmpty()	Returns true if this set contains no elements.
Iterator<E>	iterator()	Returns an iterator over the elements in this set.
boolean	remove(Object o)	Removes the specified element from this set if it is present.
int	size()	Returns the number of elements in this set (its cardinality).
Spliterator<E>	spliterator()	Creates a *late-binding* and *fail-fast* Spliterator over the elements in this set.
Object[]	toArray()	Returns an array containing all of the elements in this collection.
<T> T[]	toArray(T[] a)	Returns an array containing all of the elements in this collection; the runtime type of the retur

LinkedHashSet

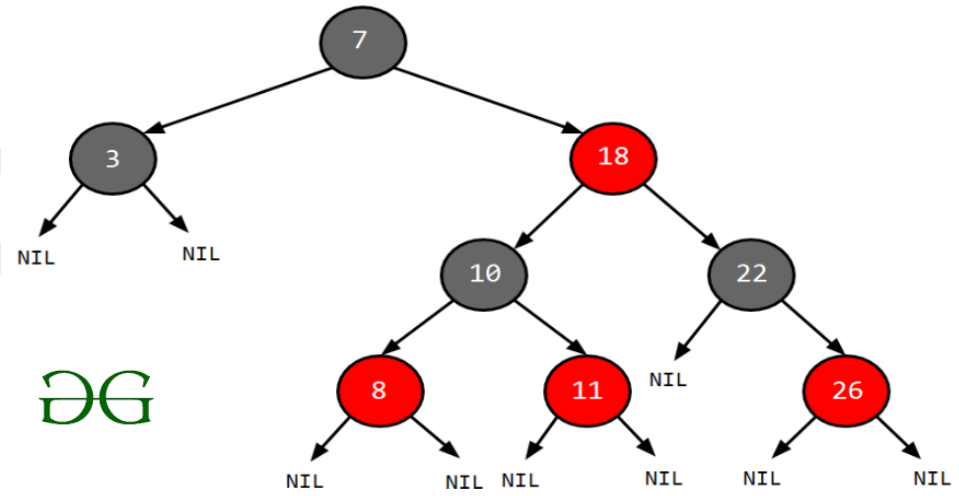
- LinkedHashSet is a subclass of HashSet.
- It stores the data in the order as added by the user. A,B,C -> ABC

TreeSet

- It stores the elements in a sorted order. - 23,45,1 -> A,Z,B -> ABz, s1, s2, s3 -> It won't able -> Collections.srot
- compareTo() method is used to identify the object uniquely and to manage the order of elements in TreeSet.
- The object you are adding in the TreeSet must be the subtype of Comparable interface.
- **If object is not the subtype of Comparable, then it will throw an exception at runtime**

java.lang.ClassCastException

- TreeSet allows storing elements of similar type only.
- Null value cannot be stored in TreeSet.



NavigableSet

- NavigableSet is an interface added in Java 6.
- It is a subtype of SortedSet.

- Its functionality is almost similar to TreeSet except it provides some extra methods for easy navigation.
- Elements from NavigableSet can be accessed in both forward and reverse order.

```
➜  Code javap java.util.Set
Compiled from "Set.java"
public interface java.util.Set<E> extends java.util.Collection<E> {
  public abstract int size();
  public abstract boolean isEmpty();
  public abstract boolean contains(java.lang.Object);
  public abstract java.util.Iterator<E> iterator();
  public abstract java.lang.Object[] toArray();
  public abstract <T> T[] toArray(T[]);
  public abstract boolean add(E);
  public abstract boolean remove(java.lang.Object);
  public abstract boolean containsAll(java.util.Collection<?>);
  public abstract boolean addAll(java.util.Collection<? extends E>);
  public abstract boolean retainAll(java.util.Collection<?>);
  public abstract boolean removeAll(java.util.Collection<?>);
  public abstract void clear();
  public abstract boolean equals(java.lang.Object);
  public abstract int hashCode();
  public default java.util.Spliterator<E> spliterator();
  public static <E> java.util.Set<E> of();
```

TheTestingAcademy | Pramod Dutta

Comparing Sets

	Data Structure	Sorting	Iterator	Nulls?
HashSet	Hash table	No	Fail-fast	Yes
Linked HashSet	Hash table + linked list	Insertion Order	Fail-fast	Yes
EnumSet	Bit vector	Natural Order	Weakly consistent	No
TreeSet	Red-black tree	Sorted	Fail-fast	Depends
CopyOnWrite ArraySet	Array	No	Snapshot	Yes
Concurrent SkipListSet	Skip list	Sorted	Weakly consistent	No

Comparable vs Comparator

Comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface to compare its instances.

Comparator is external to the element type we are comparing. It's a separate class

CompareTo(??)
Compare

Comparable interface	Comparator interface
Comparable interface present in java.lang package.	Comparator interface present in java.util package.
Sort the elements according to natural sorting order.	Sort the elements according to customized sorting order.
It contains only one method i.e. compareTo()	It contains two methods compare() and equals()
CompareTo() method is responsible to sort the elements.	compare() method is responsible to sort the elements

Queue

- Added interface in Java 5.
- PriorityQueue is one of the commonly used subclasses of Queue interface.
- PriorityQueue doesn't allow null values.
- You can add only comparable objects to PriorityQueue

```
➜  Code javap java.util.Queue
Compiled from "Queue.java"
public interface java.util.Queue<E> extends java.util.Collection<E> {
  public abstract boolean add(E);
  public abstract boolean offer(E);
  public abstract E remove();
  public abstract E poll();
  public abstract E element();
  public abstract E peek();
}
```

TheTestingAcademy | Pramod Dutta

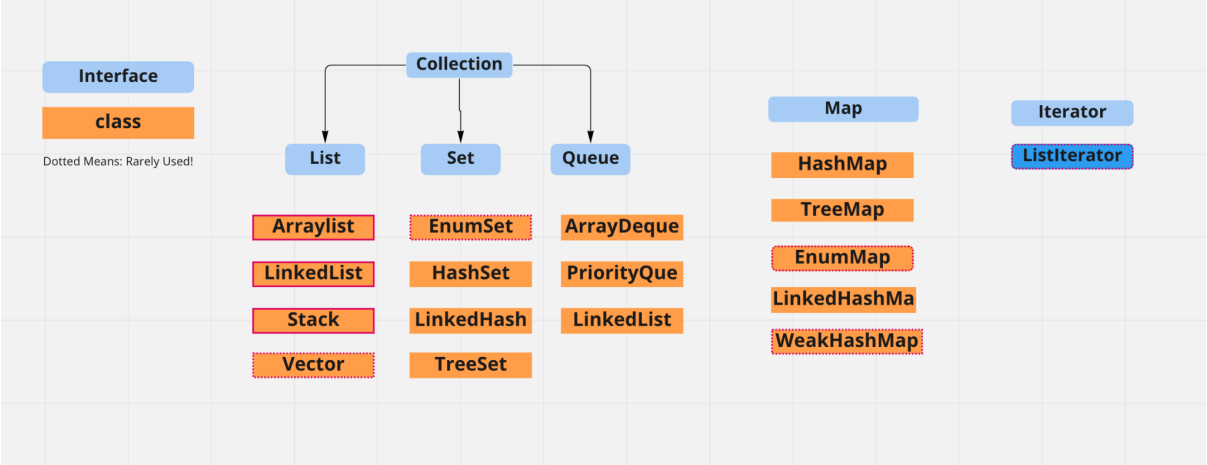
Task	Hashset	LinkedHashSet	TreeSet
Order	No Order	User added	Default sort
Duplicate Vaue	No	No	No
Null Value	Yes	Yes	No
Synchronized	No	No	No
Iterator (used?)	Yes	Yes	Yes
List Iterator	No	No	No
Enumeration	No	No	Yes
Index Representation	No	No	No
Node Representation	No	Yes	No
NavigateableSet API used	No	No	No
Serializable	Yes	Yes	Yes
Cloneable	Yes	Yes	Yes

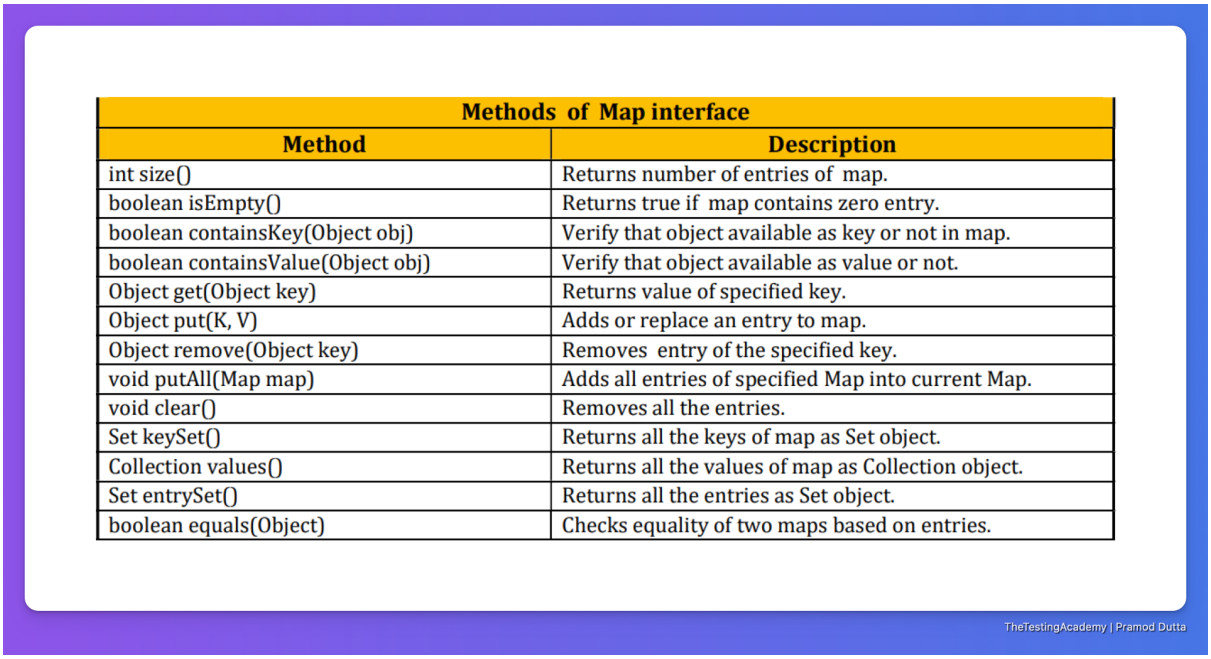
TheTestingAcademy

Map

- Map is an interface available in **java.util** package and introduced in the Collection Framework API.
- a key and a value.
- The **key is always unique** in the Map.
- A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null values like the HashMap and LinkedHashMap, but some do not like the TreeMap.
- The order of a map depends on the specific implementations. For example, TreeMap and LinkedHashMap have predictable orders, while HashMap does not.
- There are two interfaces for implementing Map in java. They are Map and SortedMap, and three classes: HashMap, TreeMap, and LinkedHashMap

- Map hm = new HashMap();
- Map interface has following concrete sub classes:
    - o HashMap
    - o LinkedHashMap
    - o TreeMap
    - o Hashtable





Methods of Map interface	
Method	**Description**
int size()	Returns number of entries of map.
boolean isEmpty()	Returns true if map contains zero entry.
boolean containsKey(Object obj)	Verify that object available as key or not in map.
boolean containsValue(Object obj)	Verify that object available as value or not.
Object get(Object key)	Returns value of specified key.
Object put(K, V)	Adds or replace an entry to map.
Object remove(Object key)	Removes entry of the specified key.
void putAll(Map map)	Adds all entries of specified Map into current Map.
void clear()	Removes all the entries.
Set keySet()	Returns all the keys of map as Set object.
Collection values()	Returns all the values of map as Collection object.
Set entrySet()	Returns all the entries as Set object.
boolean equals(Object)	Checks equality of two maps based on entries.

HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs.

No Duplicate Key on HashMap

LinkedHashMap is just like HashMap with an additional feature of maintaining an order of elements inserted into it.

The TreeMap in Java is used to implement the Map interface and NavigableMap along with the Abstract Class. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

```
class HashMap  implements Map {
  // unsorted, unordered
  // key's hashcode() is used
}
```

```
class LinkedHashMap  implements Map {
  // insertion order is maintained (optionally can maintain ac
  // well)
  // slower insertion and deletion
  // faster iteration
}
```

```
// A,C,B
// A,B,C
class TreeMap  implements Map,NavigableMap {
  // sorted order is maintained
  // implements NavigableMap
}
```

Property	HashMap	LinkedHashMap	TreeMap
Time Complexity(Big O notation) Get, Put, ContainsKey and Remove method	O(1)	O(1)	O(1)
Iteration Order	Random	Sorted according to either Insertion Order of Access Order (as specified during construction)	Sorted according to either natural Order of keys or comparator(as specified during construction)
Null Keys	allowed	allowed	Not allowed if keys uses Natural Ordering or Comparator does not support comparison on null Keys.
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()
Data Structure	List of buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black(a kind of self-balancing binary searc tree) implementation of Binary Tree. This data structure offers O(log n) for insert, Delte and Search operations and O(n) space complexity.
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of amployees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten.	Equals() and hashCode() needs to be overwritten.	Comparotor needs to be supplied for key implementation, otherwise natural order will be used to sort the keys.

HashTable

- It is similar to HashMap, but is synchronized.
- Hashtable stores key/value pairs in hash table.
- In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.
- HashMap doesn’t provide any Enumeration, while Hashtable provides not fail-fast Enumeration

Sort HashMap by Keys

- By using TreeMap
- By using LinkedHashMap
- Sort HashMap by Values
- Sort HashMap by Values using Comparator Interface

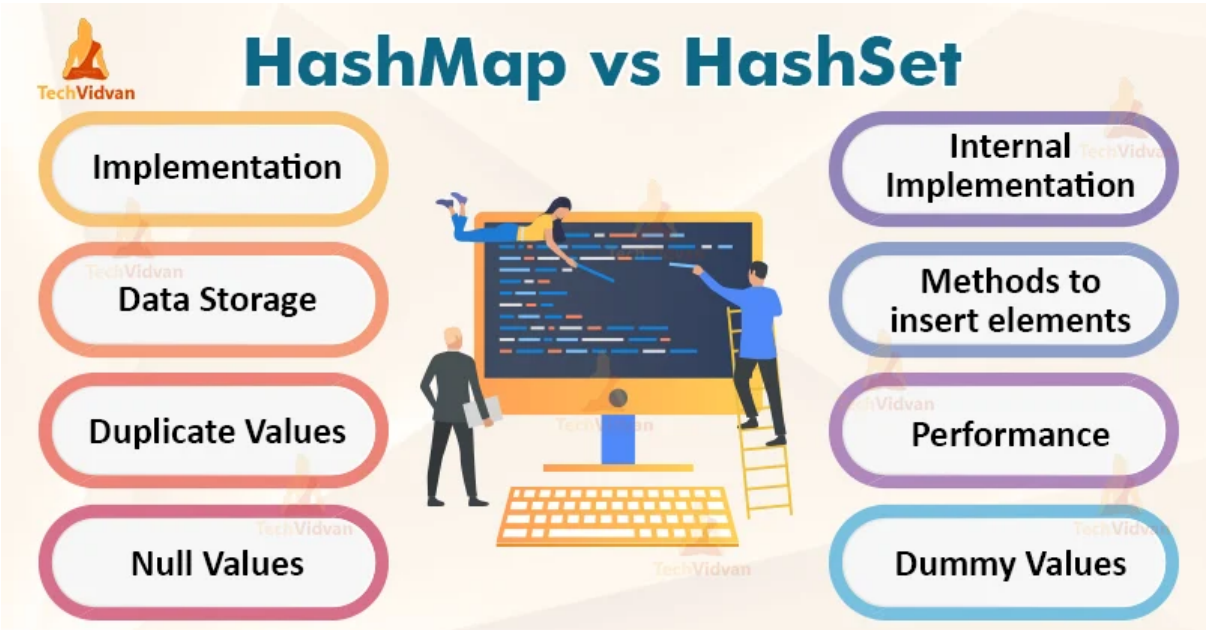
TheTesting

```
class Hashtable  implements Map {
  // Synchronized - Thread Safe - version of HashMap
  // unsorted, unordered
  // key's hashcode() is used
  // HashMap allows a key with null value. Hashtable doesn't.
}
```

		HashMap	LinkedHashMap	TreeMap	Hashhable
1	Order	Unordered	As added by user	Sorted by key	Unordered
2	Duplicate value	Yes	Yes	Yes	Yes
3	Duplicate Key	No	No	No	No
4	Different type of value	Yes	Yes	Yes	Yes
5	Different type of key	Yes	Yes	No	Yes
6	Null value	Yes	Yes	Yes	No
7	Null key	Yes	Yes	No	No
8	Synchronized	No	No	No	Yes
9	Enumeration can be used	No	No	No	Yes

TheTestingAcademy | Pramod Dutta

- Count Occurance of Character in the given String
- **Count Occurance of Word in the given String**



What are the main differences between array and collection?

- Arrays are always of fixed size, Collection, size can be changed dynamically as per need.
-
- Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
-
- Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use

What is the difference between **Set and Map?**

- Set contains values only whereas Map contains key and values both.
- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

What is the difference between HashMap and Hashtable?

No.	HashMap	Hashtable
1)	HashMap is not synchronized.	Hashtable is synchronized.
2)	HashMap can contain one null key and multiple null values.	Hashtable cannot contain any null key or null value.
3)	HashMap is not ?thread-safe,? so it is useful for non-threaded applications.	Hashtable is thread-safe, and it can be shared between various threads.
4)	4) HashMap inherits the AbstractMap class	Hashtable inherits the Dictionary class.

What is the difference between Collection and Collections?

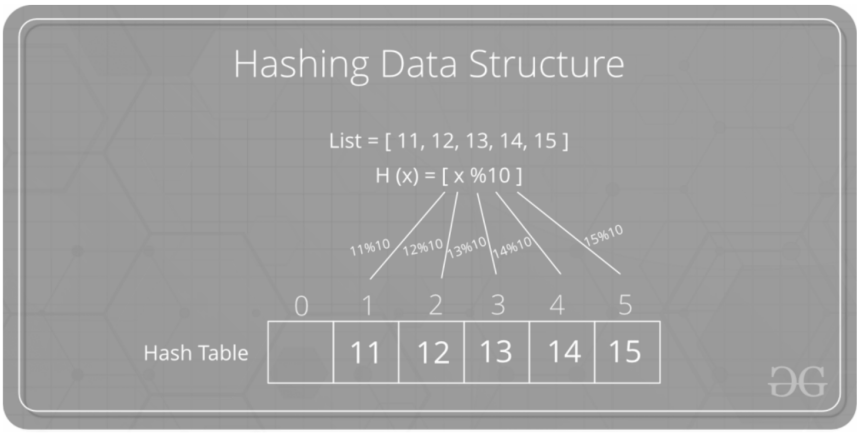
- The Collection is an interface whereas Collections is a class.
- The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
- The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

What does the hashCode() method?

- The hashCode() method returns a hash code value (an integer number).
- The hashCode() method returns the same integer number if two keys (by calling equals() method) are identical.

- However, it is possible that two hash code numbers can have different or the same keys.
- If two objects do not produce an equal result by using the equals() method, then the hashcode() method will provide the different integer result for both the objects.

Let a hash function H(x) maps the value at the index x%10 in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



.

How to synchronize List, Set and Map elements?

1. public static List synchronizedList(List l){}
2. public static Set synchronizedSet(Set s){}
3. public static SortedSet synchronizedSortedSet(SortedSet s){}
4. public static Map synchronizedMap(Map m){}
5. public static SortedMap synchronizedSortedMap(SortedMap m){}

What is the default size of load factor in hashing based collection?

The default size of load factor is 0.75. The default capacity is computed as initial capacity * load factor. For example, 16 * 0.75 = 12. So, 12 is the default capacity of Map.

Collection Framwwork QnA

How do you copy one ArrayList to another?

Using the addAll() method: You can use the addAll() method to add all elements of the original ArrayList to the new ArrayList. Example: ArrayList<Integer> copiedList = new ArrayList<>(); copiedList.addAll(originalList);

Using the subList() method: You can use the subList() method to create a new ArrayList that contains a subset of elements from the original ArrayList. Example: ArrayList<Integer> copiedList = new ArrayList<>(originalList.subList(fromIndex, toIndex));

How do you check if an element exists in an ArrayList?

Generics

- Introduction to Generics, Generics Class
- Implementing Generics for the Custom List
- Examples

"Generics allow the reusability of code, where one single method can be used for different data-types of variables or objects."

idea is to allow different types like Integer, String, … etc and user-defined types to be a parameter to methods, classes, and interfaces.

```
public class GenericsDemo {
    public static < T1, T2, T3 > void temp(T1 x, T2 y, T3 z) {
        System.out.println("This is x =" + x);
        System.out.println("This is y =" + y);
        System.out.println("This is z =" + z);

    }
    public static void main(String args[]) {
        temp( x: 1,  y: 2,  z: 3);
    }
}
```

MaxtTheerGeneric Demo

TheTestingAcademy

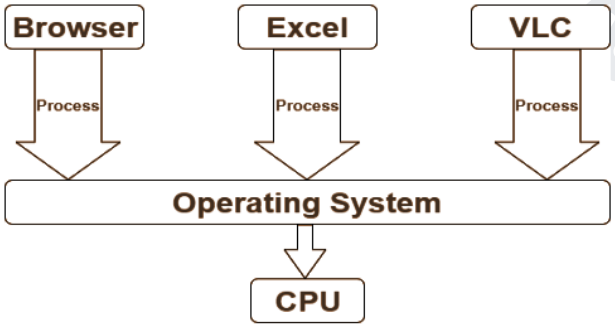
TheTestingAcademy

Multithreading

1. Multi Tasking
    a. **Multi Thread**
    b. Multi Processing

Multitasking

- CPU executes multiple jobs by switching among them.
- typically using a small-time quantum, and these switches occur so frequently that the users can interact with each program while it is running.



Multitasking

Multi Processing:

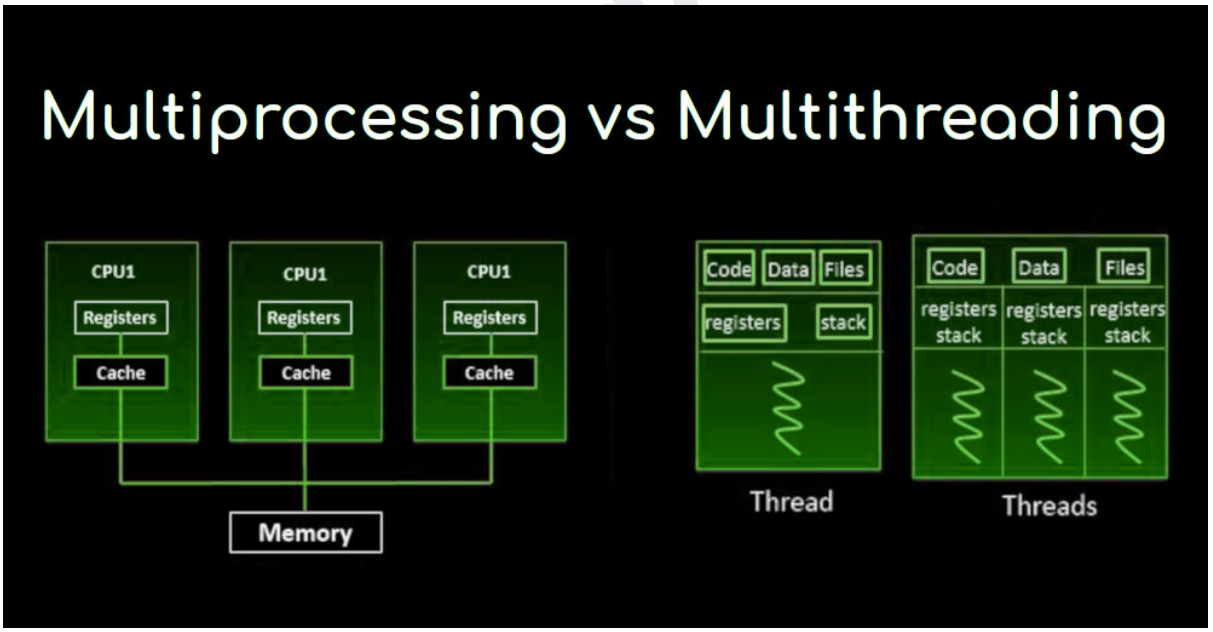
- Process is a program.
- Each process is treated as a separate execution unit and is assigned to a different CPU.
    - Running MS Word C#
    - Running PPT
    - Running Zoom
    - Running WhatsApp
- System Level concept. CPU level, Raspberry Pi 1 Single CPU
- They will have **different memories (written in different languages).**
- They can share data, but it is a very complex process.
- Context switch or control is difficult. (P1 will ask P2, P3 for permission)

TheTestAcademy

Multi Threading:

- Thread is part of Program
- **Thread is a sub process.**
- E.g. Zoom,
    - SP1, SP2, SP3 three sub processes each are Thread
    - SP1 - Audio transfer
    - SP2 - Video Recordings
    - SP3 - Chat
- Application level
- They can share the memories
- They can do it easily in threads.
- Switching to control in Threads is easy. (T1 -> T2)

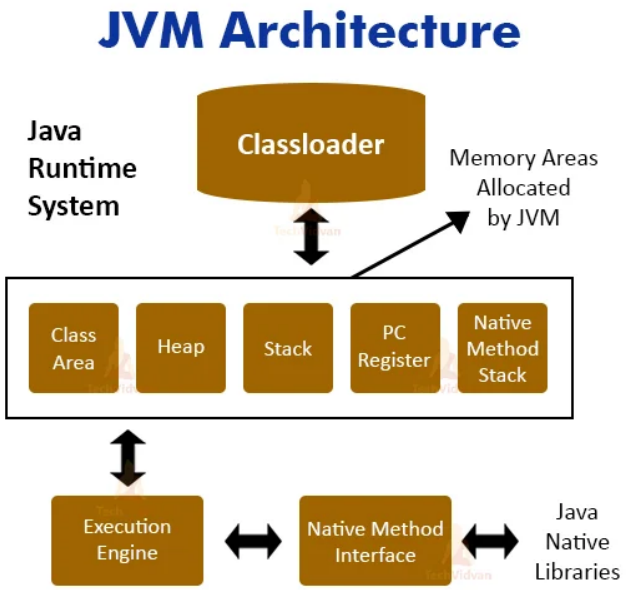
- Bank Application allows multiple users to login, bank balance, withdraw amount.
-



Multitasking vs Multithreading vs Multiprocessing

```
public class Task02 {
```

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

        Thread t = Thread.currentThread();
        System.out.println(t);
        for (int i = 0; i < 10; i++) {
            System.out.println(i+" - "+t.getName());
            System.out.println(i+" - "+t.getPriority());
            Thread.sleep(5000);
        }
    }
}
```



Flow of Java program

1. **Java Task02 123**
2. The JVM asks the OS for the memory.
3. If the OS allocates the memory -> JVM adds Stack, Heaps.
4. JVM creates 2 thread groups, Main and System (2-3 threads). (GC, finalizer)
5. JVM adds the main thread to the main group.
6. JVM started the main thread. The main thread will do the program task.
7. Check whether **Task02 .class** is available or not.
8. ByteCode verification will be done.
9. Collect parameters (123).

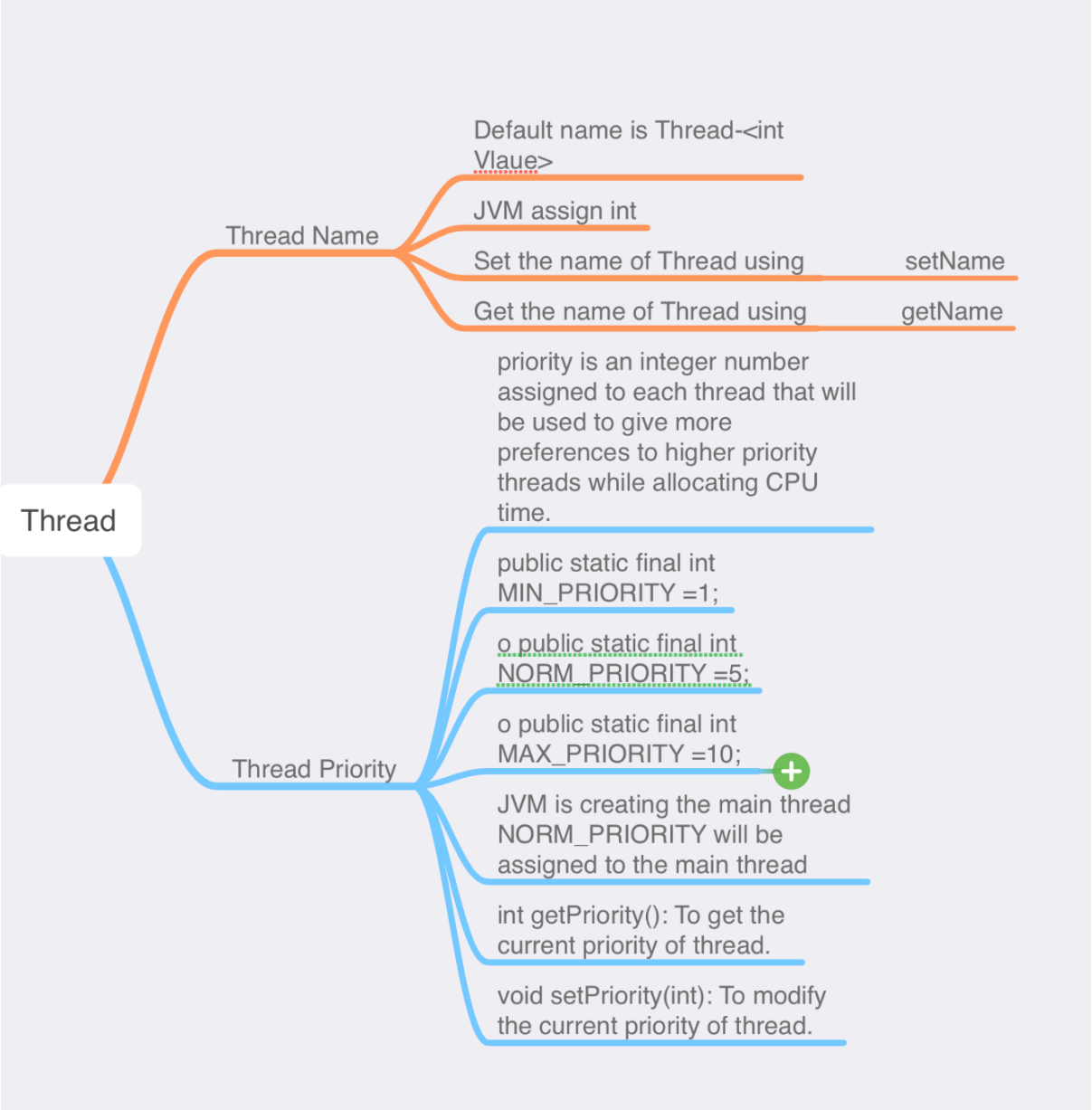
10. Verify that the main method is present.
11. Create a String array and store command-line args.
12. Invokes ClassName.main() (Task02.main()) by JVM
13. The main thread will execute your program.
14.
15. JVM will sJVM will clear garbagehut down.

Main Thread - Main Method

There are two ways you can create the Threads in Java.

1. Extend Thread
2. Runnable

```
package thetestingacademy.multithreading;

public class Task03 {
    public static void main(String[] args) {

        Worker worker = new Worker();
        worker.start();
        // Where is the Start ?
        Worker worker2 = new Worker();
        worker2.start();

    }
}

class Worker extends Thread{

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                System.out.println(i  + " --> ");
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```



Why don't we call run() method directly, why call start() method?

- We can call run() method if we want but then it would behave just like a normal method and we would not be able to take the advantage of multithreading.
- When the run method gets called though start() method then a new separate thread is being allocated to the execution of run method, so if more than one thread calls start() method that means their run method is being executed by separate threads (these threads run simultaneously).

- public Thread():
  Creates a new Thread object. When started, then run() will be invoked from the current object.

- public Thread(String);

Creates a new Thread object by specifying the name.

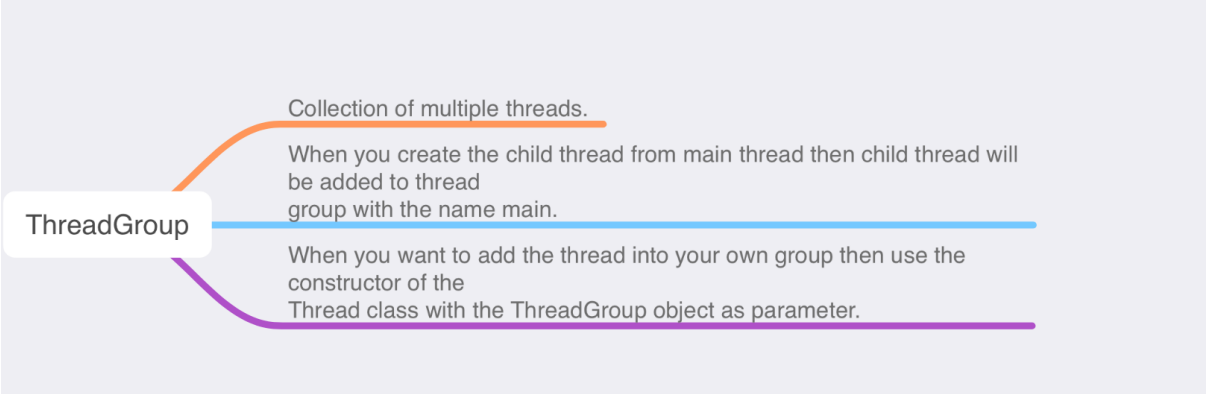
- public Thread(ThreadGroup, String);
  Creates a new Thread object by specifying the ThreadGroup name.

- public Thread(Runnable);
  Creates a new Thread object. When started then run() will be invoked from specified Runnable object.

- public Thread (Runnable, String);
  Creates a new Thread object by specifying the Runnable object and thread name.

- public Thread(ThreadGroup, Runnable);
  Creates a new Thread object by specifying the threadgroup Runnable object.
-

- public Thread(ThreadGroup, Runnable, String);
  Creates a new Thread object by specifying the ThreadGroup, Runnable object and thread name.



Methods from ThreadGroup Class

- public final String getName() **Returns the name of this thread group.**
- public final ThreadGroup getParent() Returns the parent of this thread group
- public final boolean parentOf(ThreadGroup)
  Tests if this thread group is either the thread group argument or parent of that thread group
- public final int getMaxPriority() Returns the maximum priority of this thread group.
- public final void setMaxPriority(int) Sets the maximum priority of the group.
- public final boolean isDaemon() Tests if this thread group is a daemon thread group.
- public final void setDaemon(boolean) Changes the daemon status of this thread group.
- public synchronized boolean isDestroyed()

Tests if this thread group has been destroyed.

- public int activeCount() Returns an estimate of the number of active threads in this thread group and its subgroups.
- public final void stop() Stops this thread group.

By extending Thread class	By implementing Runnable interface
◆ Define your own thread class by extending java.lang.Thread class. ◆ Override the run() method with following signature. public void run() ◆ Write the logic required for your thread inside the run() method. ◆ Create the object of Thread sub class written by you ◆ Call the start() method using your own thread object.	◆ Define your own thread class by implementing java.lang.Runnable interface. ◆ Override the run() method with following signature. public void run() ◆ Write the logic required for your thread inside the run() method. ◆ Create the object of java.lang.Thread by passing Ruuable object as parameter to constructor. ◆ Call the start() method using thread object.

Constants from Thread Class

- static final int MIN_PRIORITY Constant represents **minimum priority of a thread**
- static final int NORM_PRIORITY Constant represents **normal priority of a thread**
- static final int MAX_PRIORITY Constant represents **maximum priority of a thread**

Methods from Thread Class

Methods from Thread Class	
static native Thread currentThread()	Returns a reference to the currently executing thread object.
synchronized void start()	Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
void run()	Inherited from Runnable Interface
long getId()	Returns the identifier of this Thread.
final native boolean isAlive()	Tests if this thread is alive.
final void stop()	Stops this thread execution.
final void setPriority(int)	Changes the priority of this thread.
final int getPriority()	Returns this thread's priority.
final synchronized void setName(String)	Changes the name of this thread.
final String getName()	Returns this thread's name.
final ThreadGroup getThreadGroup()	Returns the thread group to which this thread belongs.
static int activeCount()	Returns an estimate of the number of active threads in the current thread's thread group and its subgroups.
boolean isInterrupted()	Tests whether the current thread has been interrupted.
void interrupt()	Interrupts this thread.
final void setDaemon(boolean)	Marks this thread as either a daemon thread or a user thread.
final boolean isDaemon()	Tests if this thread is a daemon thread.

Thread.State getState()	Returns the state of this thread.
static native void yield()	A hint to the scheduler that the current thread is willing to leave its current use of a processor.
static native void sleep(long)	Causes the currently executing thread to sleep (temporarily pause the execution) for the specified number of milliseconds.
static void sleep(long, int)	Causes the currently executing thread to sleep (temporarily pause the execution) for the specified number of milliseconds plus the specified number of nanoseconds
final synchronized void join(long)	Waits at most millis milliseconds for this thread to die.
final synchronized void join(long, int)	Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.
final void join()	Waits for this thread to die.

Concurrency vs Parallelism

Concurrency refers to the ability of a system to handle multiple tasks or threads at the same time, allowing them to be executed independently and asynchronously.

Concurrency enables a system to perform multiple tasks simultaneously, but it does not necessarily mean that the tasks are executed simultaneously.

For example, **a web server can handle multiple client requests concurrently**, but each request is handled by a single thread and the server may not be able to process all the requests at the same time.

Parallelism refers to the ability of a system to execute multiple tasks or threads simultaneously, using multiple cores or processors.

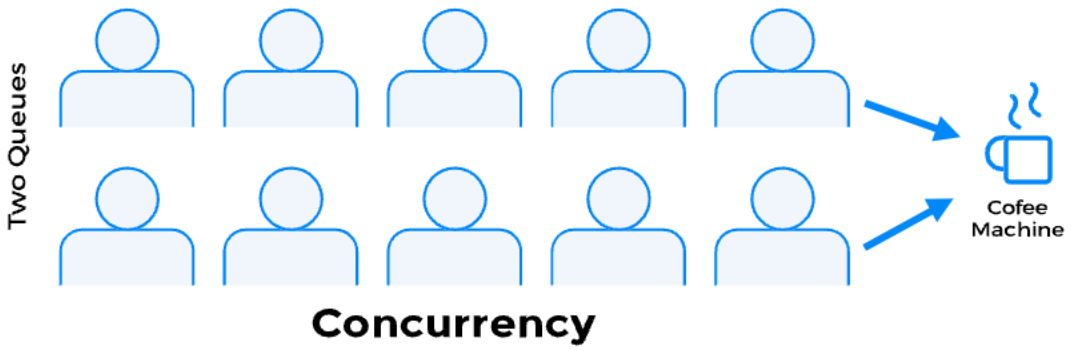
Parallelism enables a system to perform multiple tasks at the same time, rather than one at a time.

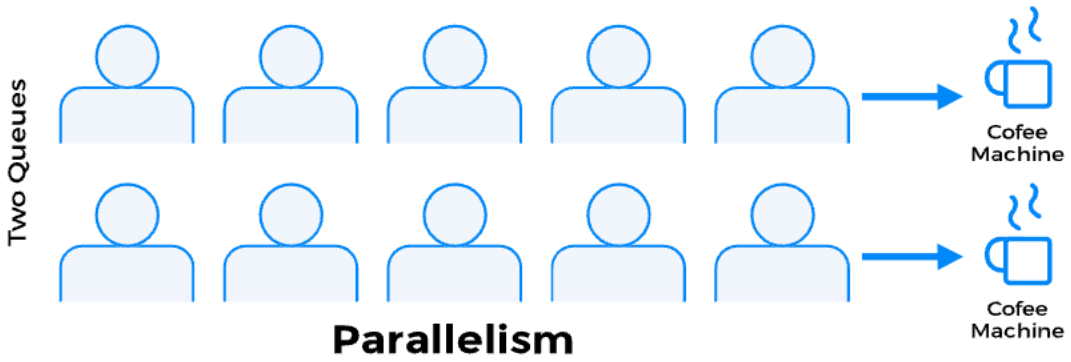
For example, a parallel program can use multiple cores or processors to perform a large computation in less time.

In summary, **concurrency enables a program to handle multiple tasks at the same time**, but it does not necessarily mean that the tasks are executed simultaneously.

Parallelism enables a program to execute multiple tasks simultaneously, using multiple cores or processors.

TheTestingAcademy



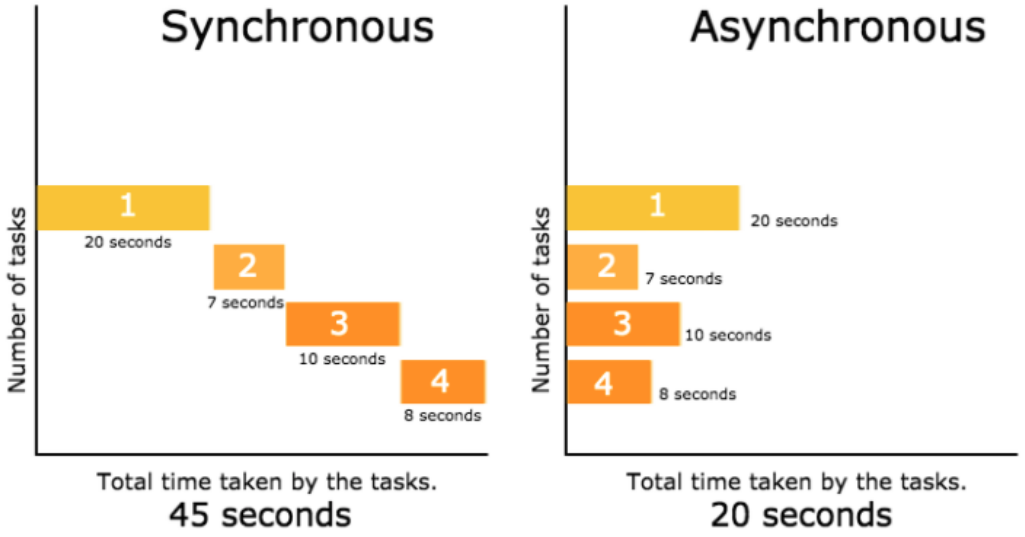


Synchronous vs Asynchronous

Synchronous operations block the execution of the program or system until the operation is completed.

This means that the program or system will not execute any other code or perform any other tasks until the current task is completed.

This can lead to a situation where the program or system is "stuck" waiting for an operation to complete, and this can make the program or system unresponsive or slow.

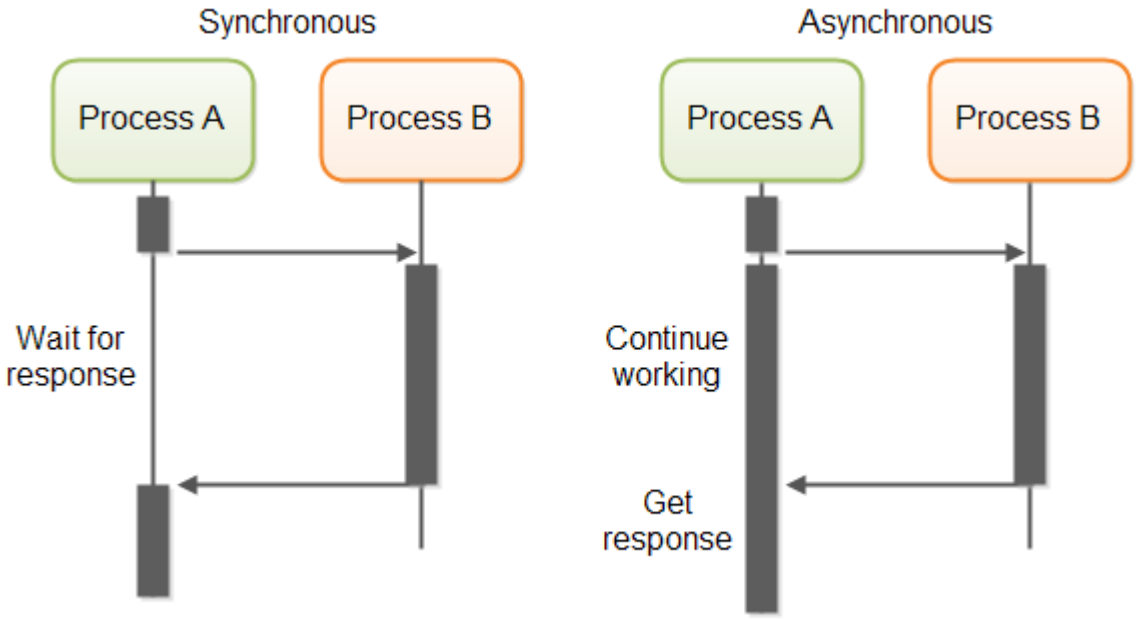


Asynchronous operations, on the other hand, do not block the execution of the program or system.

The program or system can continue to execute other code or perform other tasks while an asynchronous operation is in progress.

This can make the program or system more responsive and efficient, because it can perform multiple tasks at the same time.

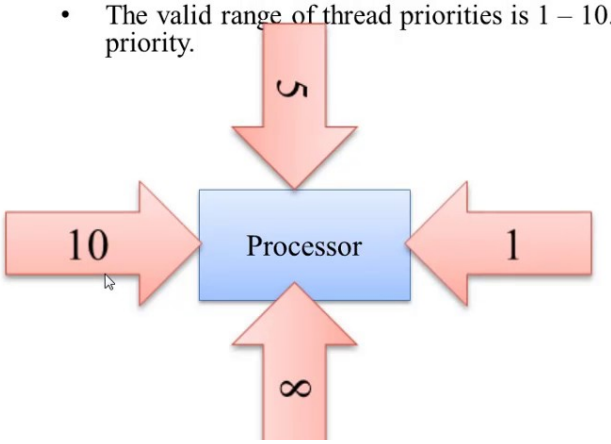
In summary, synchronous operations block the execution of a program or system until the operation is completed, while asynchronous operations do not block the execution and allow the program or system to continue running while the operation is in progress.



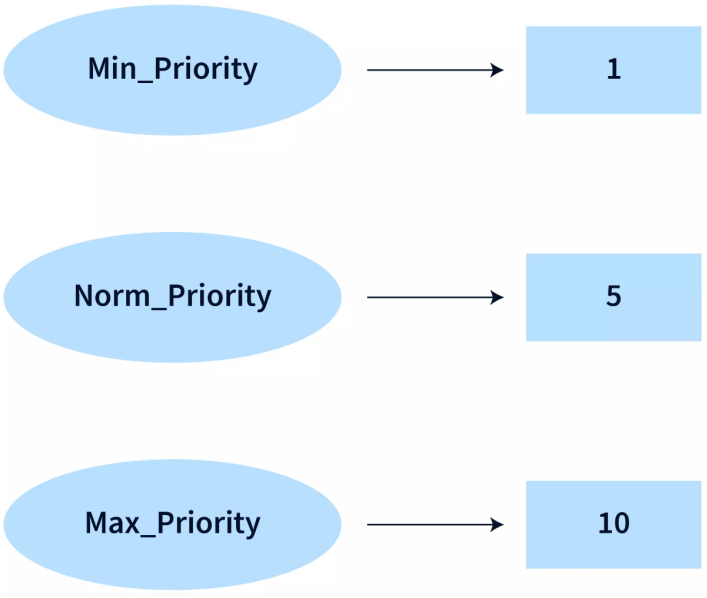
Thread Priority

Thread priority determines how a thread should be treated with respect to others.

- Every thread in java has some priority it may be default priority generated by JVM or customized priority provided by programmer.
- The valid range of thread priorities is 1 – 10. where 1 is min priority and 10 is max priority.



- Thread scheduler will use priorities while allocating processor.
- The thread which is having highest priority will get the chance first.



Java Thread Priority

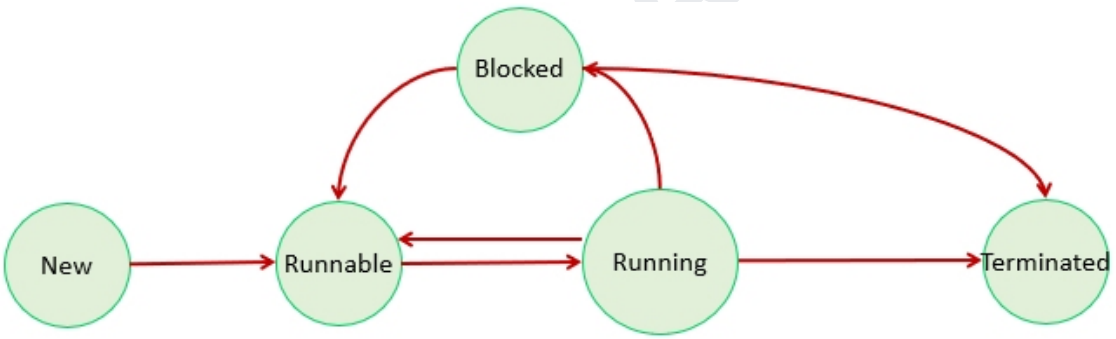
You can create your own Thread Group also

Constants from Thread.
State enum
NEW RUNNABLE BLOCKED
WAITING TIMED_WAITING TERMINATED

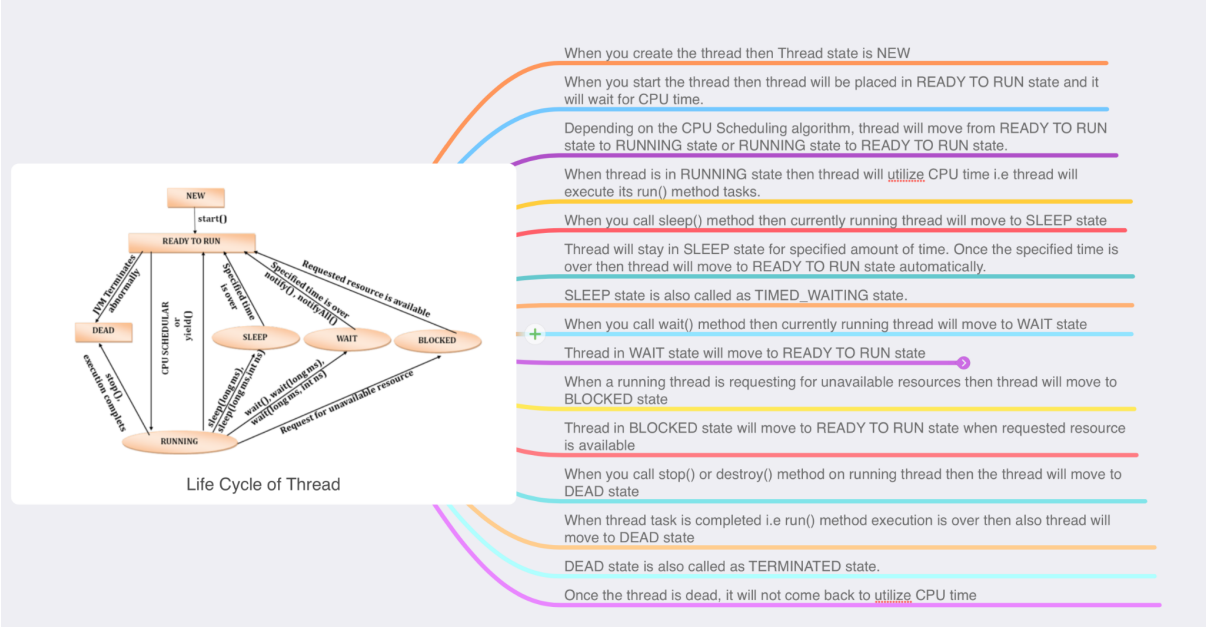
Thread Life CYCLE

1. Create a Thread.
2. Start the Thread (th.start()) , T1, T2.
3. Initial state will be ready TO Run (waiting for CPU).
4. Thread scheduler selects the t1

- New - Instance of thread created which is not yet started by invoking start() it will be.
- Runnable - After Invocation of start() and before it is selected to be run by the scheduler.
- Running - After the thread scheduler has selected it.
- Non-runnable - Thread alive, not eligible to run.
- Terminated - run() method has exited.

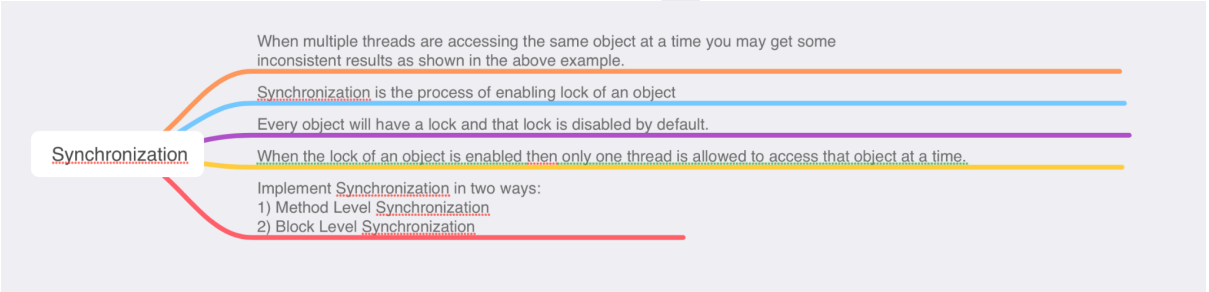


The Academy



Synchronization

- Lock on an Object.



- Withdraw example for the Synchronization

Method Level Synchronization

- Applying synchronized keyword to the method is called Method Level Synchronization.
- You can apply synchronized keyword for both instance methods and static methods.
- When instance method is synchronized then the object which is used to invoke the synchronized method will be locked by the current thread.
- When static method is synchronized then the default object of the invoking class will be locked by the current thread.

```
public synchronized void show(){ ... }
```

Block Level Synchronization

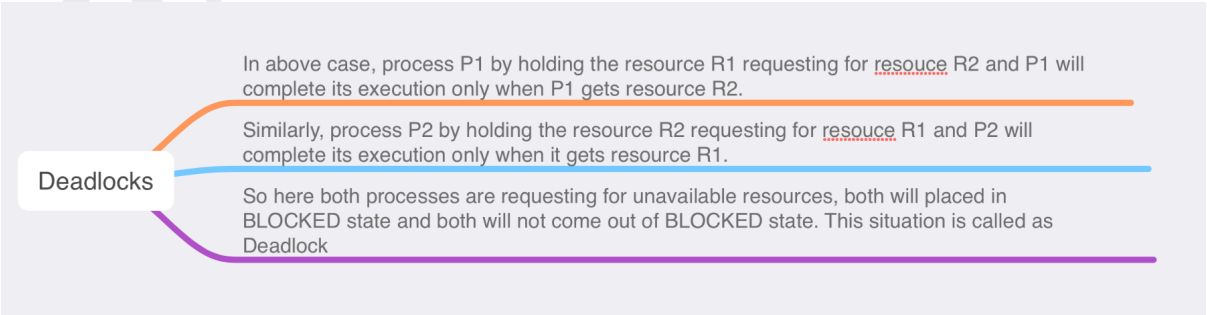
- Applying synchronized keywords for local blocks is called as Block Level Synchronization.
- You must specify some object as a parameter to synchronized block.
- When a local block is synchronized, then the object which you are passing as a parameter to the synchronized block will be locked by the current thread.

```
class Pramod{

ArrayList al = new ArrayList();
void show(){ ...
synchronized(al){ ... } ...
}
}
```

Important points for the Synchronization

- Case 1
- Case 2
- Case 3
- Case 4
- Case 5
- Case 6

Deadlocks



Solution in ATBNotes.zip (or Github URL)

```
package thetestingacademy.multithreading.deadlock;

public class ThreadDeadLock {

    public static void main(String[] args) {
        final String resource1 = "ATB";
        final String resource2 = "MTB";
        // t1 tries to lock resource1 then resource2
        Thread t1 = new Thread() {
            public void run() {
                synchronized (resource1) {
                    System.out.println("Thread 1: locked resource 1");

                    try {
                        Thread.sleep(100);
                    } catch (Exception e) {
                    }

                    synchronized (resource2) {
                        System.out.println("Thread 1: locked resource
2");
                    }
                }
            }
        };

        // t2 tries to lock resource2 then resource1
        Thread t2 = new Thread() {
            public void run() {
                synchronized (resource2) {
                    System.out.println("Thread 2: locked resource 2");

                    try {
                        Thread.sleep(100);
                    } catch (Exception e) {
                    }

                    synchronized (resource1) {
                        System.out.println("Thread 2: locked resource
1");
                    }
                }
            }
        };
```

```
        t1.start();
        t2.start();
    }
}
```

Wait.

- wait() method is an instance method available in java.lang.Object class.
- When you call wait() method on any locked object then lock of that object will be released
- immediately and the current thread will be moved to WAIT state.
- Following are the wait() methods defined in Object class:
    - o public final void wait()
    - o public final native void wait(long ms)
    - o public final void wait(long ms, int ns)
- If you call wait() method on unlocked object then java.lang.IllegalMonitorStateException
- will be thrown at runtime

```
package thetestingacademy.multithreading.synchronization.WaitDemo;

public class WaitDemo {

    public static void main(String[] args) {

        WaitForMe waitForMe = new WaitForMe();

        CustomThread customThread = new CustomThread(waitForMe, "A");
        CustomThread customThread2 = new CustomThread(waitForMe, "B");

        customThread.start();
        customThread2.start();
    }
}


class WaitForMe {

    //    synchronized void show(){
    synchronized void show() {
        Thread thread = Thread.currentThread();
        for (int i = 0; i < 5; i++) {
            System.out.println(" Name -> " + thread.getName() + " -> " +
i);
            try {
                this.wait(1000);
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    // Blocking a Block or Third Party Object
//    void show() {
//        Thread th = Thread.currentThread();
//        ArrayList al = new ArrayList();
//        synchronized (al) {
//            for (int i = 1; i <= 5; i++) {
//                System.out.println(th.getName() + " - show -" + i);
//                try {
////this.wait(1000);
//                    al.wait(1000);
//                } catch (Exception ex) {
//                    System.out.println(ex);
//                }
//            }
//        }
//    }
}


class CustomThread extends Thread {
    WaitForMe wait;

    public CustomThread(WaitForMe wait, String name) {
        super(name);
        this.wait = wait;
    }

    @Override
    public void run() {
        wait.show();
    }
}
```

Notify & NotifyAll()

- These methods are instance methods available in java.lang.Object class.
- When you call notify() method on any locked object then following task will happen:
    - Only one thread which has been waiting for long time will be picked.

    - Checks whether the resources released by the thread are available or not.
    - If resources are available, then the thread acquires the resources and goes to READY TO RUN state.
    - If resources are not available then the thread goes to BLOCKED state.

- When you call notifyAll() method on any locked object then following task will happen:
    - All or some required number of threads as per the requirement which are in WAIT state will be picked.
    - Checks whether the resources released are available or not.
    - If resources are available then the threads acquire the resources and goes to READY TO RUN state.
    - If resources are not available then the threads go to BLOCKED state.

Code sample at ATBNotes.zip

Join()

- This method will be used to join one thread at the end of another thread.
- In Main thread -> t1.join()
    - Main thread will be joined at the end of t1.
- In t1 thread -> t2.join()
    - t1 thread will be joined at the end of t2.

Thread Pool

- A thread pool in Java is a collection of worker threads that are used to execute multiple tasks concurrently.
- The main advantage of using a thread pool is that it allows the program to reuse threads, which can improve performance and reduce the overhead of creating and destroying threads.
- In a thread pool, a fixed number of worker threads are created and are managed by a thread pool executor. When a task is submitted to the thread pool, it is placed in a task queue and is executed by an available worker thread. Once a task is completed, the worker thread is returned to the thread pool and is available for executing another task.
- Java provides the Executor framework, which simplifies the task of creating and managing thread pools. The Executor interface has a single execute() method, which is used to submit tasks to the thread pool

```
Executor executor = Executors.newFixedThreadPool(5);
```

TheTestingAcademy

```
executor.execute(new RunnableTask());
```

Thread Local

ThreadLocal is a class in the Java standard library that allows you to create variables that are specific to a single thread. These variables are called "thread-local" variables, because they are local to the thread that accesses them.

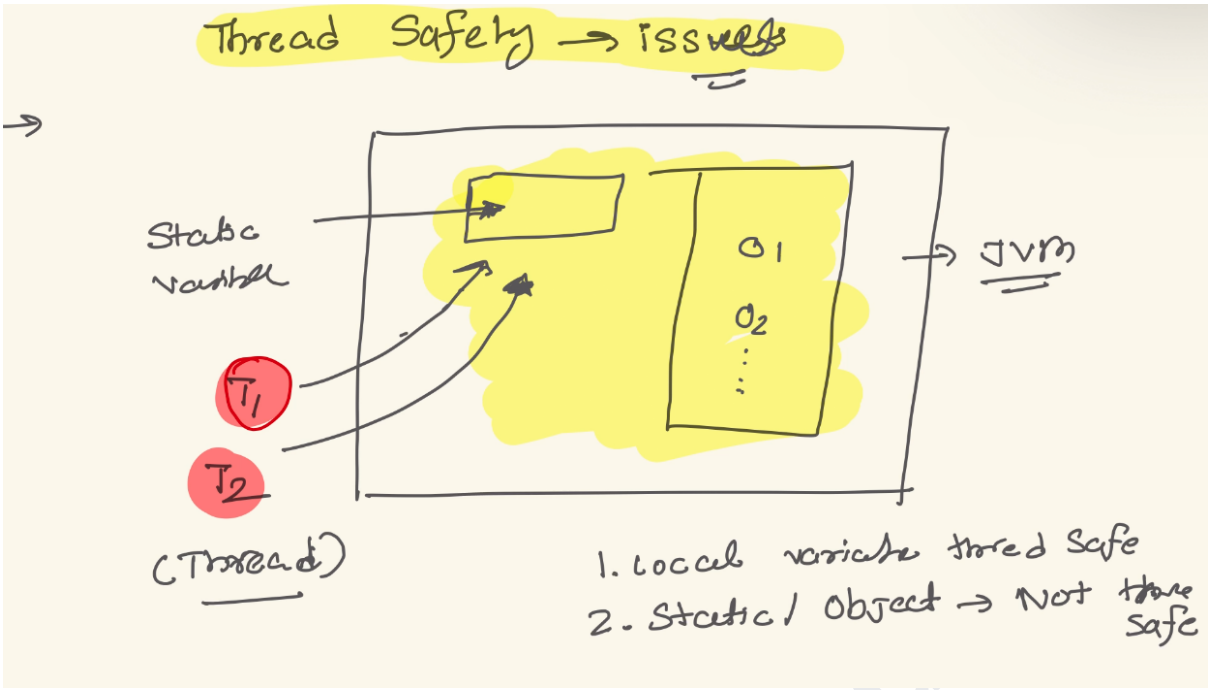
Instance variables are more prone to the Thread Safty, Local variables are always safe.

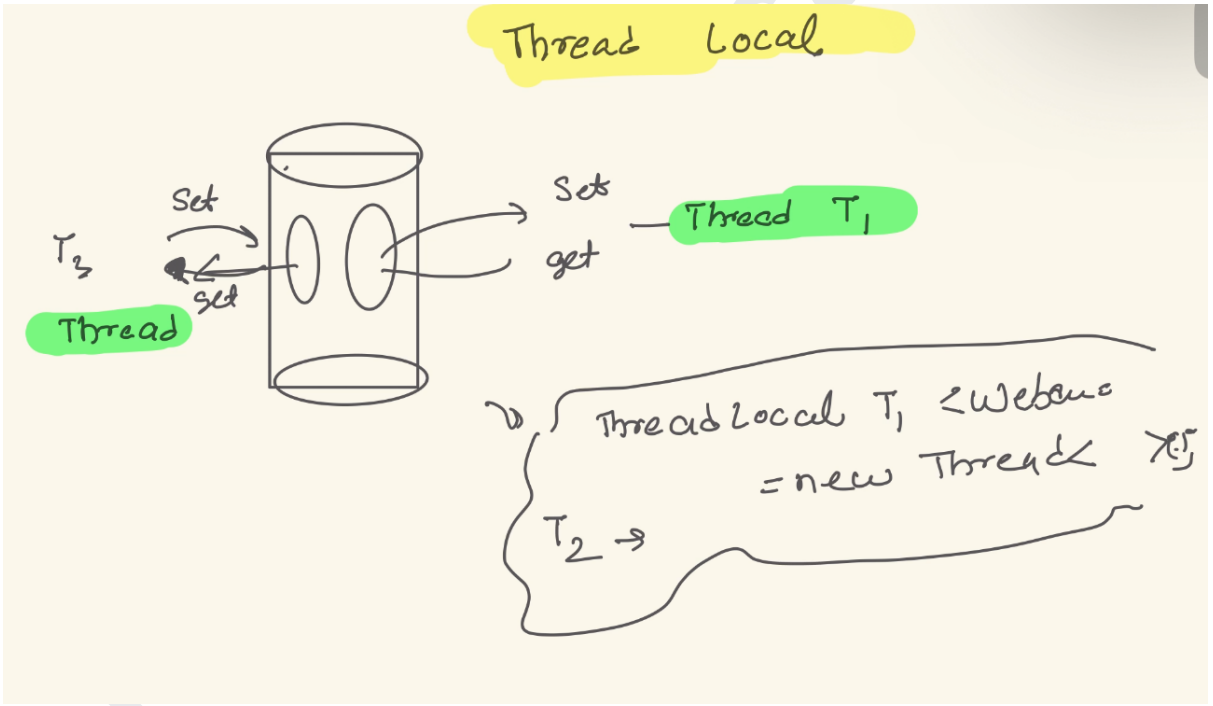
```
class MyThreadLocal {
    private static ThreadLocal<Integer> myThreadLocal = new
ThreadLocal<>();
    public static void set(Integer value) {
        myThreadLocal.set(value);
    }
    public static Integer get() {
        return myThreadLocal.get();
    }
}
```

```
MyThreadLocal.set(10);
System.out.println(MyThreadLocal.get()); // 10
```

It's important to note that each thread will have its own copy of the thread-local variable, so the value of the thread-local variable set by one thread will not affect the value of the thread-local variable in another thread.

For example, ThreadLocal can be used to store a user's session information in a web application, where each user's session information is specific to the thread that is handling the user's request.





What is the finalizer Thread?

- A finalizer thread is a special type of thread in Java that is used to run the finalize() method of an object before it is garbage collected.
- The finalize() method is a protected method that is defined in the Object class and can be overridden by subclasses to perform cleanup tasks, such as releasing resources or closing open file handles.

```java
import java.io.File;

public class MyResource {
    private File file;
    // constructor and other methods

    protected void finalize() throws Throwable {
        try {
            file.delete();
        } finally {
            super.finalize();
        }
    }
}
```

In this example, the finalize() method is used to delete a file when the MyResource object is eligible for garbage collection.

However, it's important to note that the finalize() method is not guaranteed to be run before an object is garbage collected, and it's not guaranteed to be run at all.

So, it's not recommended to rely on finalize() method to release resources, instead use try-catch-finally or try-with-resources statements to release resources and also use explicit methods like close() to release resources.

it's not recommended to use finalize() method for critical tasks, because it's not guaranteed to be run and it might cause performance issues.

Can I overload the run() method in thread class?

Overloading of run() method is possible

Can I define the run() method as synchronized?

Yes, we can synchronize a run() method in Java, but it is not required because this method has been executed by a single thread only.

Can I define a thread class without overriding the run () method?

If we don't override run() method, compiler will not flash any error and it will execute run() method of Thread class that has empty implemented, So, there will be no output for this

thread.

How do I pause thread execution for specified amount of time?

sleep() method can be used to pause the execution of the current thread for a specified time in milliseconds.

What is the difference between sleep() and wait() method?

Wait() method releases lock during Synchronization. Sleep() method does not release the lock on object during Synchronization.

What is daemon thread?

Daemon thread in Java is a low-priority thread that runs in the background to perform tasks such as garbage collection.
public final void **setDaemon**(boolean on)

What is race condition in Java?
A condition in which the critical section (a part of the program where shared memory is accessed) is concurrently executed by two or more threads.

TheTestingAcademy