Full Stack Web Development Training

- Front Technology: **Angular**
- Backend Technology: Core Java, Spring Framework(Core, DAO, MVC, Data JPA), Spring Boot
- Software Required:

1. JDK
2. Notepad++/Editplus/Sublime text
3. Eclipse/ STS (Spring Tool Suite)
4. Nodejs
5. VSTS

==========Time Duration to Complete the course==========

Core Java : 20 days

Spring(Core, MVC, Data JPA) : 5 days

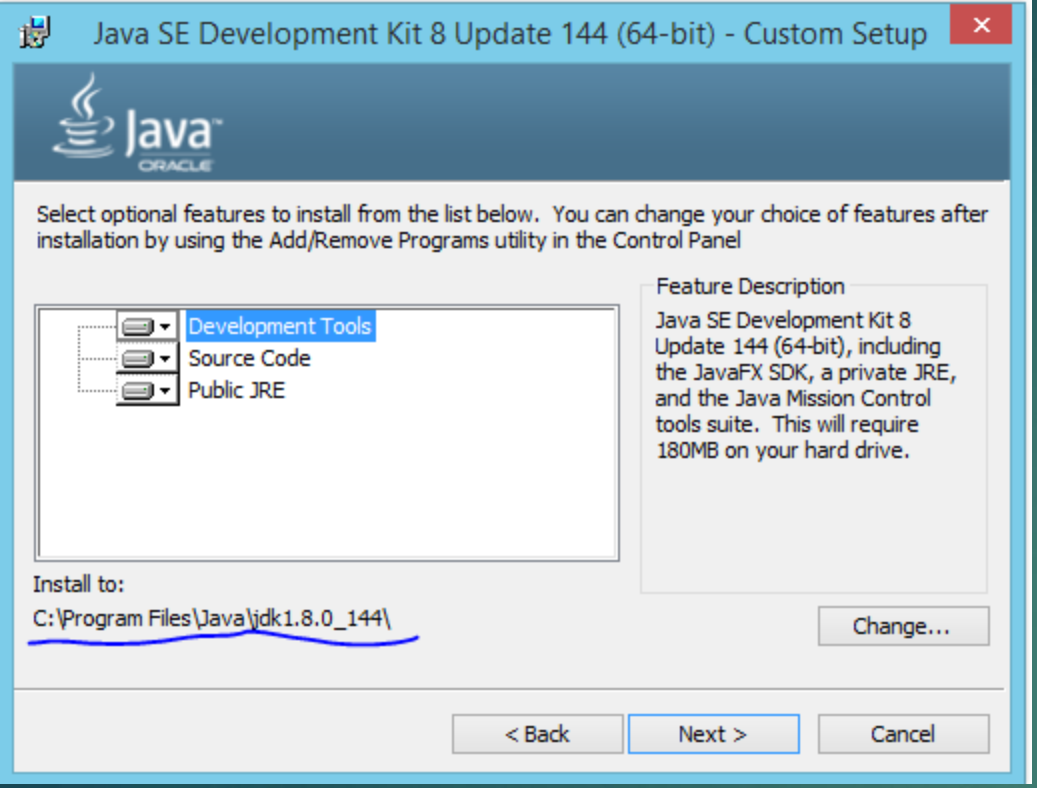
Spring Boot: 5 days

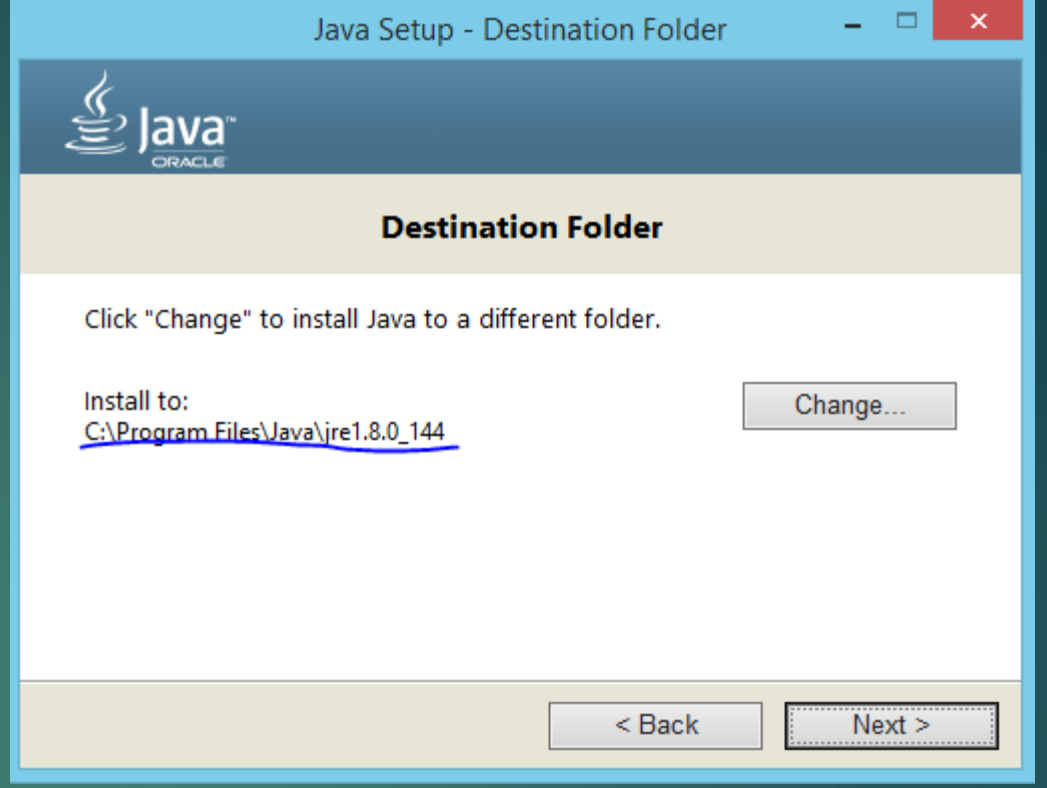
Angular: 15 days

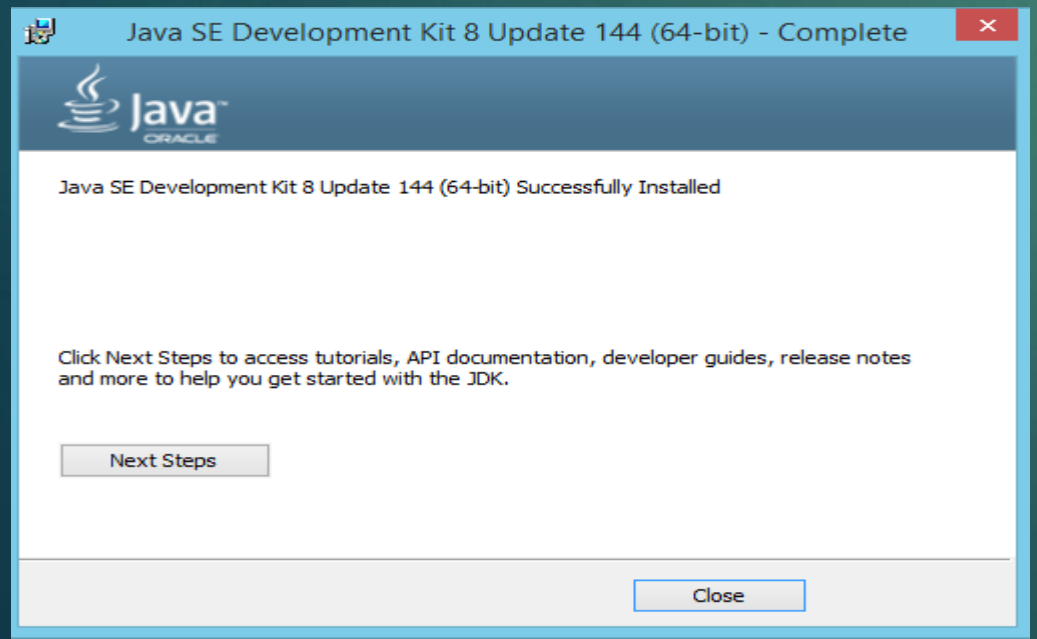
END OF THE COURSE: Create One Project for end-to-end development

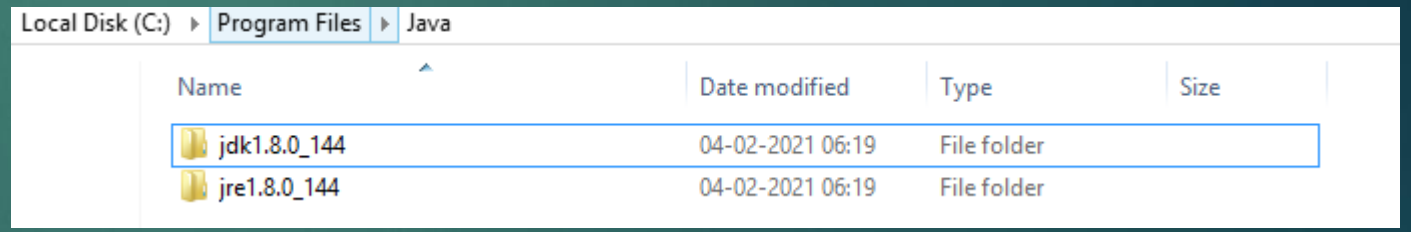
==========GIT URL's==========

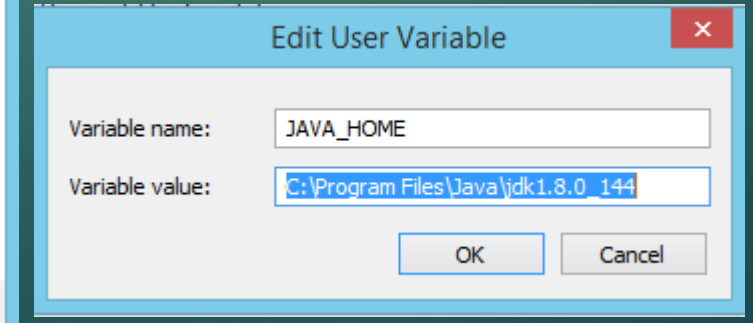
https://github.com/indrajeetydv/Full-Stack-Web-Development-

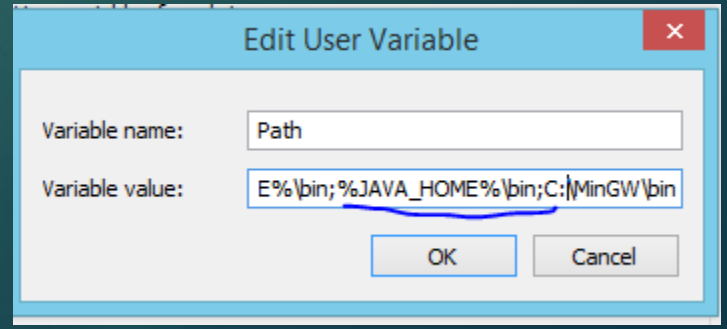

Java SE Development Kit 8 Update 144 (64-bit) - Custom Setup
Java
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Source Code
Public JRE
Feature Description
Java SE Development Kit 8 Update 144 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.
Install to:
C:\Program Files\Java\jdk1.8.0_144\
Change...
< Back
Next >
Cancel


Java Setup - Destination Folder
Java
Destination Folder
Click "Change" to install Java to a different folder.
Install to:
C:\Program Files\Java\jre1.8.0_144
Change...
< Back
Next >


Java SE Development Kit 8 Update 144 (64-bit) - Complete
Java
Java SE Development Kit 8 Update 144 (64-bit) Successfully Installed
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close


Local Disk (C:)
Program Files
Java
Name
Date modified
Type
Size
jdk1.8.0_144
04-02-2021 06:19
File folder
jre1.8.0_144
04-02-2021 06:19
File folder


Edit User Variable
Variable name:
JAVA_HOME
Variable value:
C:\Program Files\Java\jdk1.8.0_144
OK
Cancel


Edit User Variable
Variable name:
Path
Variable value:
E%\bin;%JAVA_HOME%\bin;C:\MinGW\bin
OK
Cancel

Why Java?

- C,C++ programming languages supports only stand-alone application, it can only be executed in current system, cannot be executed from remote system via network call.
- Java Programming language mainly designed to develop internet application by providing platform independency.

Definition of Java:

- Java is a very simple, high-level, secured, multithreaded, platform independent, object-oriented programming language. It was developed by James Gosling in SUN Microsystem in 1990's for developing internet applications. Its first version release in January 23, 1996.

Java Features:

1. Simple
2. Secure
3. Robust
4. Portable
5. Architecture neutral
6. Object Oriented
7. Multithreaded
8. High Performance
9. Distributed
10. Dynamic

Terminology Used in Programming language

- Source Code: Developer written program, it is written according to the programming language syntax.
- Complied Code: Compiler generated program that is converted from source code.
  - Compiler: it is a translation program that converts source code into machine language at once.
  - Interpreter: it is also translation program that converts source code into machine language but line by line.
- Executable code: OS understandable readily executable programme (.exe files)
- Compilation: It is a process of translating source code into compiled code.
- Execution: It is a process of running compiled code to get output.

Java Technology

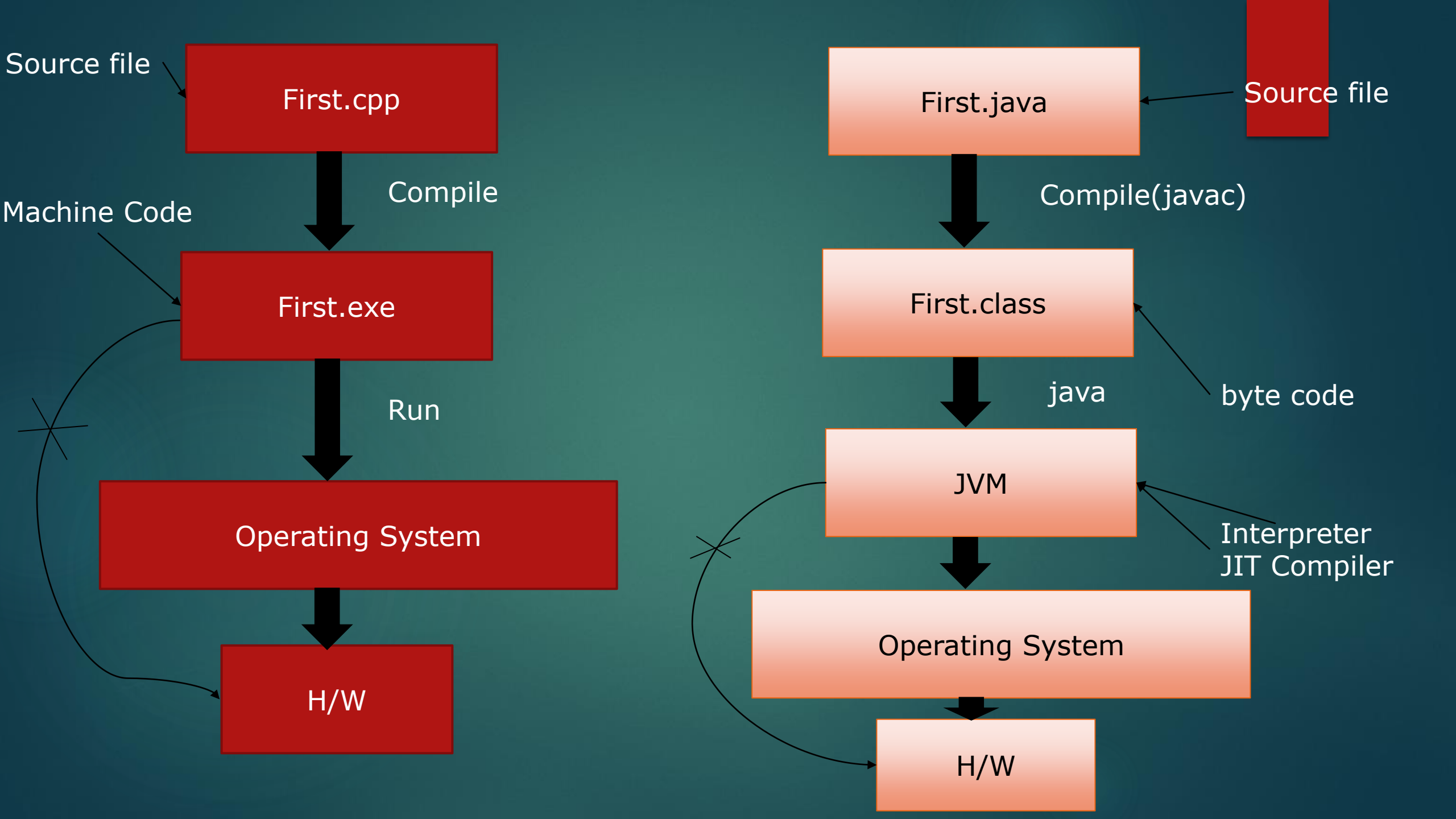
- Unlike other high level programming language, Java Technology is both platform and programming language.
- Platform is a hardware or software environment in which programs are executed.
- Java has its own software based platform called JVM - Java Virtual Machine – to execute Java Programs. Like C or C++ programs, Java programs are not directly executed by OS.

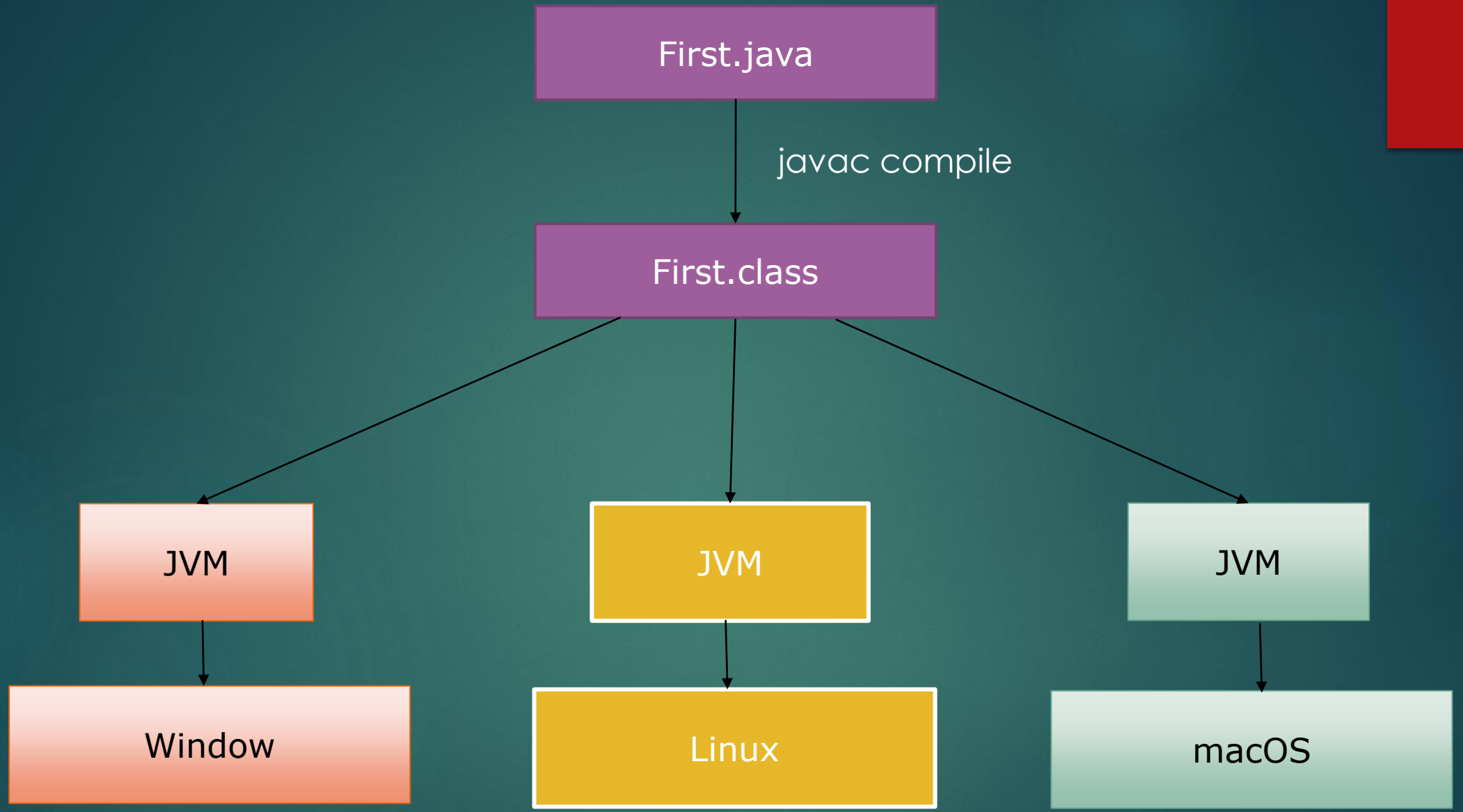
What is platform?

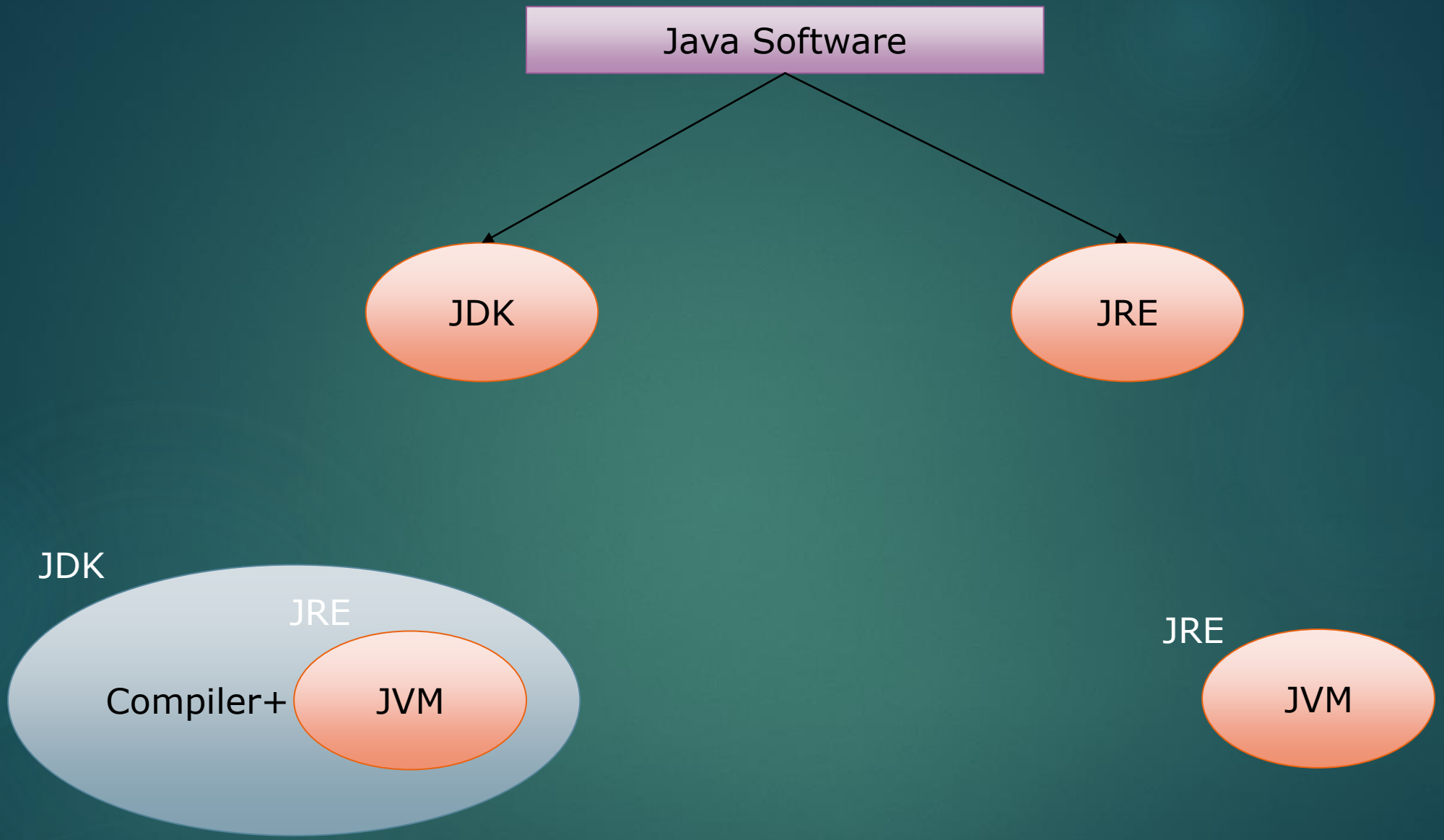
- Platform is a hardware or software environment in which programs are executed.
- For instance, computer platform is (Operating System + hardware Devices)

Difference between next() and nextLine()

- To Read one word we will use next()
- To Read complete sentence we will use nextLine()
- The only difference between the methods nextLine() and next() is that nextLine() reads the entire line including white spaces, whereas next() reads only upto the first white space and then stops.


Source file
First.cpp
Compile
Machine Code
First.exe
Run
Operating System
H/W
First.java
Source file
Compile(javac)
First.class
java
byte code
JVM
Interpreter
JIT Compiler
Operating System
H/W


First.java
javac compile
First.class
JVM
JVM
JVM
Window
Linux
macOS


Java Software
JDK
JRE
JDK
JRE
Compiler+
JVM
JRE
JVM

JDK,JRE and JVM

JDK: Java Development Kit

JRE: Java Runtime Environment

- JVM is a subset of JRE, and JRE is subset of JDK. When we install JDK, JRE is also installed automatically. JRE software available as a separate pack, so we can install JRE alone.
- JDK has both compiler and JVM. So using JDK we can develop, compile and execute new java application and also we can modify, compile and execute already developed application.
- JRE has only JVM. Hence using JRE we can only execute already developed allocations.

JIT compiler:

- JIT compiler stands for Just-in-Time compiler, it is the part of JVM which increases the speed of execution of a java program.

IS JVM Platform independent?

- JVM is responsible for interpreting the ByteCode and finally converting into the machine native code and since the machine code depends on the actual OS and real processor, it should be clear that JVM is platform-dependent.
- Since Java's ByteCode doesn't contain any system calls to OS (That would have been platform-dependent stuff), JVM still needs to be able to interact with operating system for the thing like reading and writing files, Making network connection, displaying the output of the program onto the screen etc.- making it platform dependent. Specific versions of the JVM are needed for each underlying platform to take account of different native instruction sets and machine capabilities (So we can't run Linux JVM on windows and vice versa). The JVM is packages as a part of Java runtime environment.

Identifier, Variables- Naming Rules and Keywords

Identifiers: Identifiers are symbolic names, used for the identification. Identifiers in Java can be a class name, variable name, constant name, array name, methods name, package names. There are some reserved words in Java, which we can't use as identifiers.

1. Case Sensitive: if they have lower case and upper case means they are different. Two variables with same name are not allowed. Even single character is different then that variable is also different even name is same.
2. Contains Alphabet, Numbers, _ and $
3. Starts with Alphabet, _ and $ means shouldn't start with a number
4. Should not be a keyword
5. Should not be a class name
6. No limit on the length of the name
7. Follow Camel case

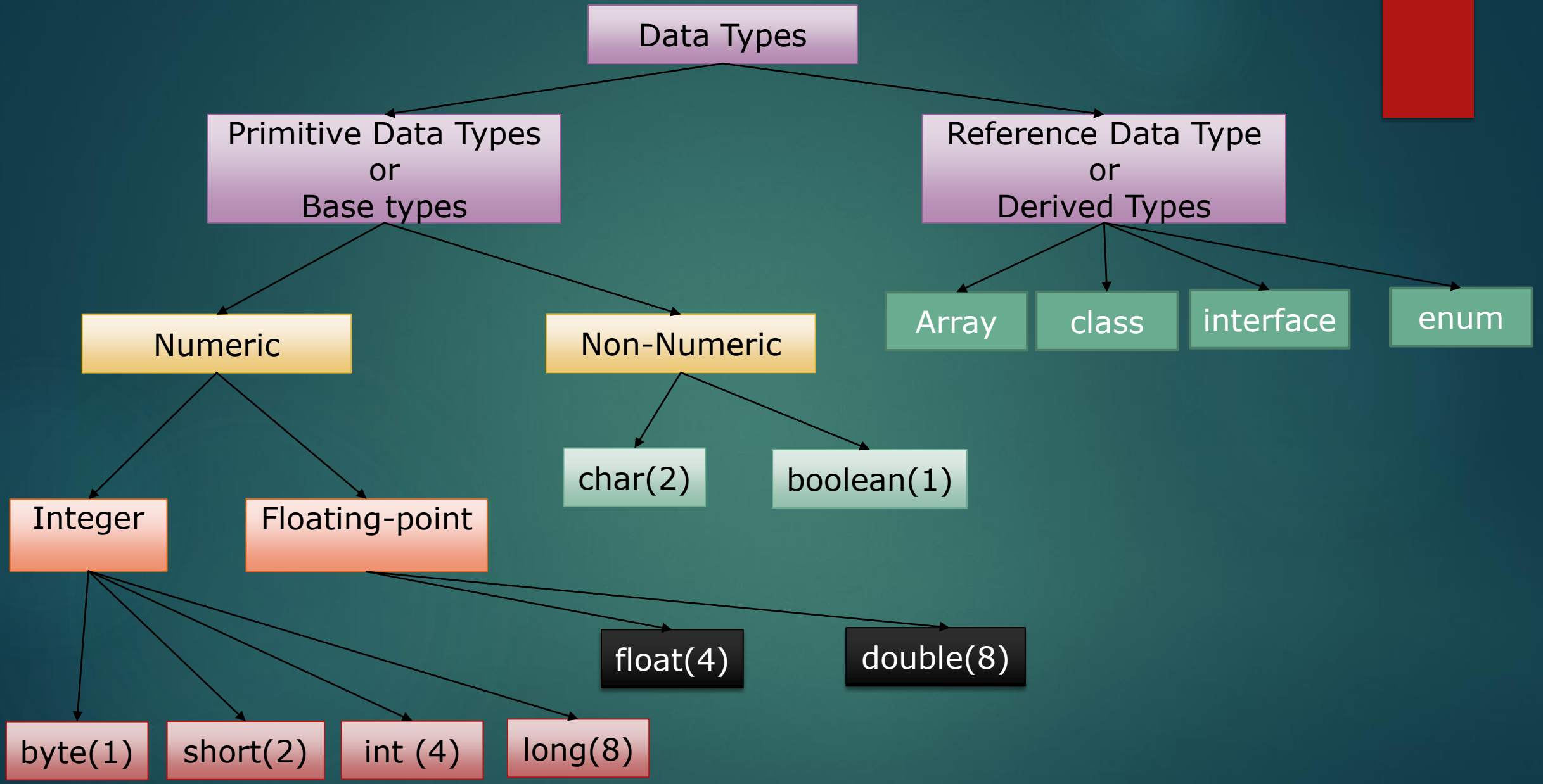
Keywords: Keywords are predefined identifiers available directly throughout the JVM. They have a special meaning inside Java source code.

Rule:

1. Keywords cannot be used as user defined identifiers by a programmer either for variable or method or class name, because keywords are reserved for their intended use.
2. All characters in keyword must be used in lower case because identifier is a case sensitive.

Reserved Keywords

Java Files: ❖class ❖interface ❖enum **Data Types:** ❖ byte ❖ short ❖ int ❖ long ❖ float ❖ double ❖ boolean ❖ void	**Control Statement:** **1. Conditional** ❖ if ❖ else ❖ switch ❖ case ❖ Default **2. Loop** ❖ while ❖ do ❖ for **3. Transfer** ❖ break ❖ Continue ❖ return	**Modifiers:** ❖ static ❖ final ❖ abstract ❖ native ❖ transient ❖ volatile ❖ synchronized ❖ strictfy	**Object Representation:** ❖ this ❖ super ❖ instanceof	**Exception handling:** ❖ try ❖ catch ❖ finally ❖ throws ❖ throw ❖ assert **Unused keyword:** ❖ goto ❖ Const **Default literals:** ❖ null ❖ True ❖ false
Memory Location: ❖ static ❖ new	**Accessibility Modifier:** ❖ private ❖ protected ❖ public	**Inheritance relationship:** ❖ extends ❖ Implements	**Package:** ❖ package ❖ import	


Data Types
Primitive Data Types
or
Base types
Reference Data Type
or
Derived Types
Numeric
Non-Numeric
Array
class
interface
enum
Integer
Floating-point
char(2)
boolean(1)
byte(1)
short(2)
int (4)
long(8)
float(4)
double(8)

Data Type's Size, Range and Default Values

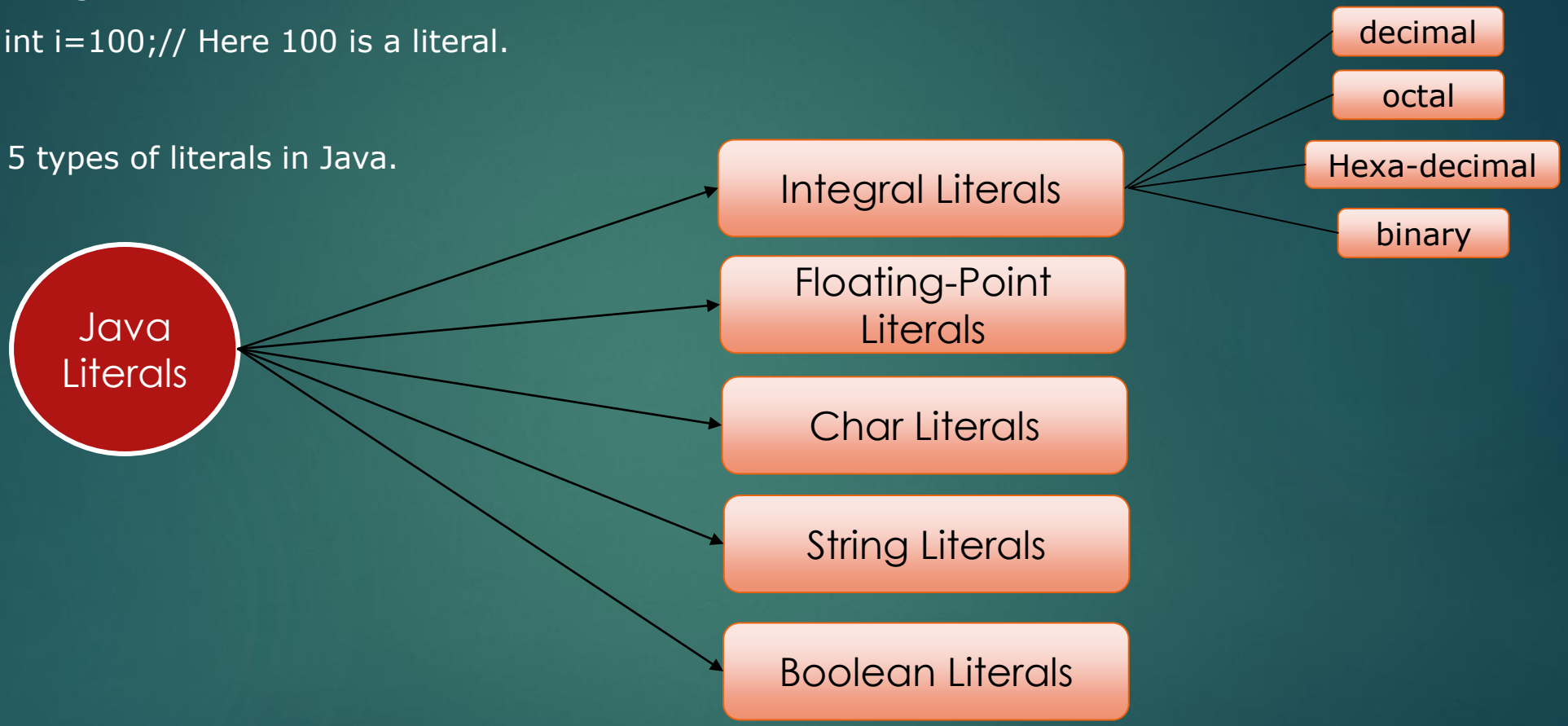
Data Type Name	Size (byte's)	Range		Default Value
byte	1	2^7 bits	-128 to 127	0
short	2	2^15 bits	-32,768 to 32,767	0
int	4	2^31 bits	-2,147,483,648 to 2,147,483,647	0
long	8	2^63 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4	2^31 bits	±1.4e-45 to ± 3.4e+38	0.0f
double	8	2^65 bits	±4.9e324 to ±1.7e308	0.0d
char	2	2^15 bits	0 to 65,535	One white space or \u0000
boolean	1	2^7 bits	false or true	false

Literals

- Literals are number, text, or anything that represent a value. In other words, Literals in Java are the constant values assigned to the variable. It is also called a constant.

Example: int i=100;// Here 100 is a literal.

There are 5 types of literals in Java.



Integral Literals:

- All integer type literals are called integral literals.
- We can specify the integer literals in 4 different ways.

1. Decimal Literals (Base 10): digits from 0-9 are allowed.
2. Octal Literals (Base 8): digits from 0-7 are allowed and should always have prefix 0(zero).
3. Hexa-Decimal Literals(Base 16): digits from 0-9 and also characters from [a-f] are allow. We can use both uppercase and lowercase characters. Literals value should be prefixed with 0x or 0X.

4. Binary Literals(Base 2): From Java 1.7 onwards we can specify literals values even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.

By default, every literal is of int type, we can specify explicitly as long type by suffixed with l or L. There is no way to specify byte and short literals explicitly but indirectly we can specify. Whenever we are assigning integral literal to the byte variable and if the value within the range of byte then the compiler treats it automatically as byte literals.

Floating Point Literals:

- By default every floating point literal is of double type and hence we cant assign directly to float variable. But we can specify floating point literal as float type by suffixed with f or F. We can also specify explicitly floating point literal as double type by suffixed with d or D. Of course this convention is not required.

Character Literals:

- The single character placed inside single quote is considered as character literal. All character literals are of type char.

Sting Literals:

- Characters placed inside double quotes is considered as string literal. All string literals are of type java.lang.String.

PRIMITIVE TYPE CONVERSION:

- THE PROCESS OF CHANGING ONE TYPE OF VALUE TO ANOTHER TYPE OF VALUE IS CALLED TYPE CONVERSION.
- WE DEVELOP TYPE CONVERSION BY ASSIGNING A VALUE OF ONE VARIABLE TO A VARIABLE OF ANOTHER TYPE.

INT A=10;

FLOAT F=A;

- THERE ARE TWO TYPES OF CONVERSION

1. IMPLICIT CONVERSION /AUTOMATIC TYPE CONVERSION /WIDENING
2. EXPLICIT TYPE CONVERSION/CASTING/NARROWING

AUTOMATIC OR IMPLICIT CONVERSION:

- IF STR<=DTR THEN THAT CONVERSION IS CALLED IMPLICIT CONVERSION. IN THIS CONVERSION THER IS NO LOSS OF DATA HENCE COMPILED WILL COMPILE THE CLASS SUCCESSFULLY.
- WE DON'T LOSE THE INFORMATION

EXAMPLE:

INT A=10; //SIZE: 4 BYTES

FLOAT F=A; //SIZE: 8 BYTES : WIDDENING

EXPLICIT CONVERSION OR NARROWING:

- PERFORMING THE TYPE CONVERSION FROM HIGHEST RANGE DATA TYPE VARIABLE TO LOWEST RANGE DATA TYPE VARIABLE USING CAST OPERATOR IS CALLED EXPLICIT CONVERSION.
- CAST OPERATOR IS A DATA TYPE PLACED IN PARENTHESIS AFTER "=" OPERATOR AND BEFORE SOURCE VARIABLE.

SYNTAX:

```
<Destination data type> <variable name>=(data type) <source type>;
```

BY USING CAST OPERATOR

1. WE ARE CONVINCING TO COMPILER THAT THE VALUE STORED IN SOURCE TYPE VARIABLE IS WITHIN THE RANGE OF CAST OPERATOR TYPE.
2. WE ARE ALLOWING THE JVM TO REDUCE THE SOURCE VALUE TO CAST OPERATOR TYPE IF ITS RANGE IS GREATER THAN CAST OPERATOR TYPE.

EXAMPLE:

```
int a =10
byte b=a; //CTE: imcompatiable types:Possible Lossy conversion
          //even i can store 10 into b variable but int data type range is greater than
byte datatype range
```

```
int A=130;
byte b=(byte) A;  // Here we are telling to compiler that the value stored in A is within
the range of int, so
                  //compiler allows this conversion as it assumes ther is no loss of data
output: -126
```

NOTE: NO CE, NO RE, ASSIGNMENT IS PERFORMED BY REDUCING ITS VALUE TO THE CAST OPERATOR RANGE BY USING 2'S COMPLEMENT AND STORES THAT RESULT IN THE DESTINATION VARIABLE.

SHORT CUT FORMULA:

[MINRESULT+(RESULT-MAXRESULT-1)]

Rules in Primitive type conversion and cast operator

1. Source and destination **data type** must be compatible; otherwise it leads to **CTE: "incompatible types"**. Except boolean all other primitive types are compatible. It means boolean value or variable cannot to assigned to any other data types. **CTE: inconvertible type**
2. Source type **range** must be less than or equals to destination type range, otherwise it leads to CTE: "**possible loss of precision**".

Rules of Cast Operator:

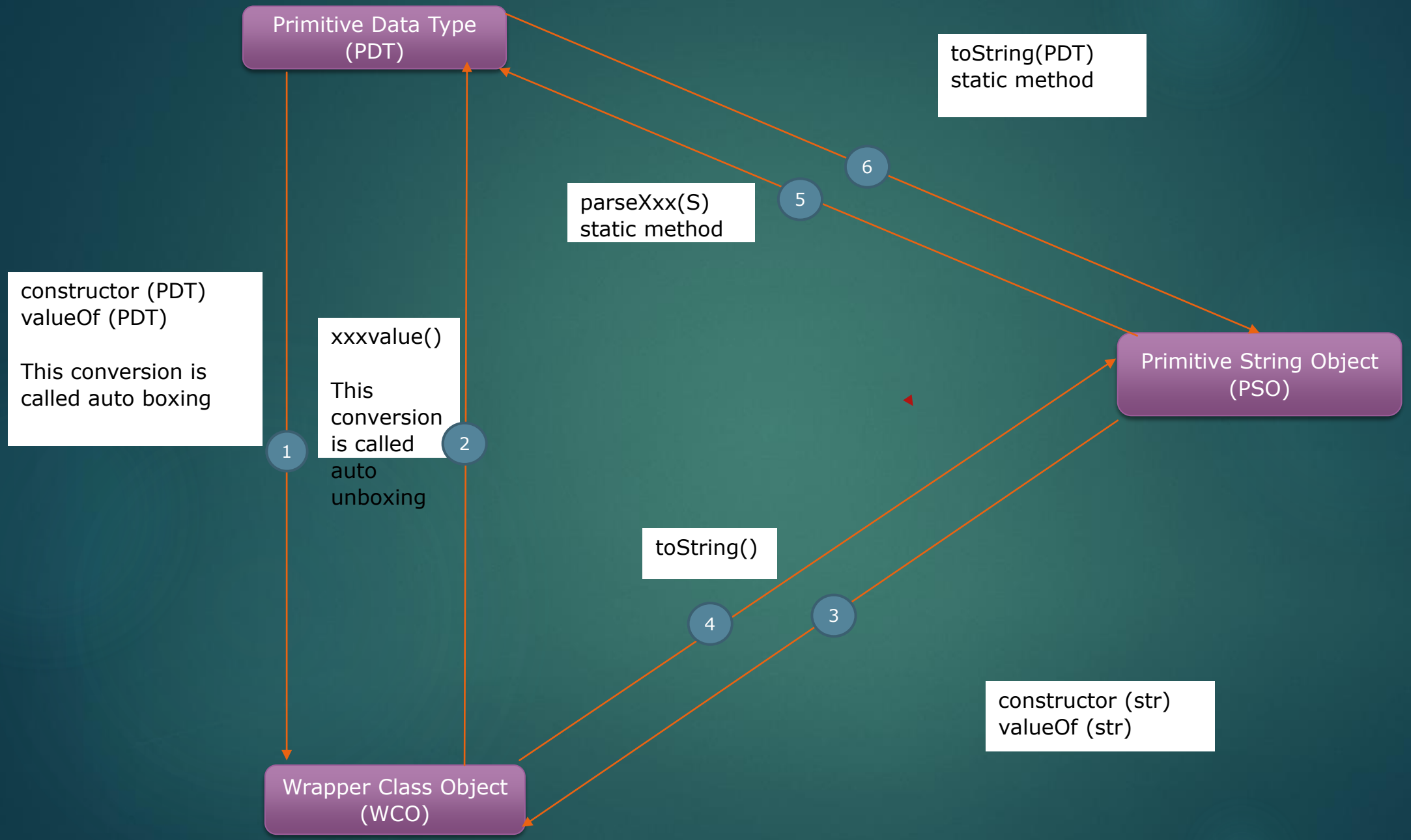
- **CTE: "inconvertiable type":** cast operator data type must be compatible with source type else leads to this error
- **CTE: "incompatiable type:":**it should be compatible with destination type else it leads to this error
- **CTE: "possible loss of precession":** its range must be <= destination type else it leads to this error

```
int i=130;
short s=(byte)i; //-126
```

Wrapper Classes, Auto Boxing and Auto Unboxing

- The classes which are used to represent primitive values as an object are called wrapper classes.
- In java.lang packages we have 8 wrapper classes one per each primitive type to represent them as an object.
- Among them 6 wrapper classes represent number type values these wrapper classes are called number wrapper classes.
- These 6 number wrapper classes are subclasses of a class Number which is an abstract class.
- Number class has 6 xxxValue() method to read primitive value from wrapper class object. Except byteValue() and shortValue() remaining all 4 methods are abstract methods so it is called abstract class.

- Converting primitive type to wrapper class object automatically is called Auto Boxing.
- Converting wrapper class object to primitive type automatically is called Auto Unboxing.


Primitive Data Type (PDT)
Primitive String Object (PSO)
Wrapper Class Object (WCO)
toString(PDT) static method
parseXxx(S) static method
constructor (PDT) valueOf (PDT)
This conversion is called auto boxing
xxxvalue()
This conversion is called auto unboxing
toString()
constructor (str) valueOf (str)
1
2
3
4
5
6

Primitive types	*Wrapper classes*
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Char
boolean	Boolean
void	Void

Number

public byte byteValue()
public short shortValue()

public abstract int intValue()
public abstract long longValue()
public abstract float floatValue()
public abstract double doubleValue()

public char charValue()

public char booleanValue()

Package

- A folder this is linked with java class is called package/ A group of classes is called packages.
- Package are used to organize related or similar class, interfaces and enum into one group. For example: java.sql package has all classes needed for database operation. Java.io package has classes related to input-output operation.
- Package is used to separate new classes from exiting classes. So by using package we can create multiple classes with same name and also we can create user defined classes with same name.
- Packages are also used to avoid naming conflict between the classes. Using package, we can give same name to different classes.
- **Syntax**:

package <package name>;

- package statement should be first statement in a java file.
- **Compilation:**

javac –d <path in which package should be copied> source filename

- **Execution:**

java <packagename>.<classname which have main method>

- Three ways to update the classpath

1. Using java command option "-cp" or "-classpath"
2. Using "Set Classpath" command
3. Using "Environment Variable window"

- **import keyword:** this is used to access other package members from this package classes.

Syntax:

```
import packagename.*;
import packagename.classname;
```

- First syntax allows compiler and JVM to access all public members (classes, interfaces & enums) of that imported package, whereas second syntax allows compiler and JVM to access only that imported class.
- **Rule:** import statement must be placed before all class definitions and after package statement.
- **static import:** this feature is introduced in Java 5 to import static members of a class.
- By using this feature we can access all

1. non-static members without using class name from other classes with in the package and
2. protected and public members form outside package class members without using class name.

Syntax:

```
import static packagename.classname.*;
import static packagename.classname.staticmembername;
```

- first import allows to call all static members of the class.
- Second import allows to call the imported static members.

Access Modifier

- There are 4 types of access modifiers in Java.

private: these members are only accessible within the class.

default: Default members with no-access modifier are access or visible within a package only.

protected: these members can be accessible within package form all classes but from outside package only in subclass that too only by using subclass object.

public: These members are accessible form the places of project.

Variables and Types of Variables

- A variable is an identifier whose values can be changed during the program execution.
- Java Variable is nothing but memory location used to stored the data temporarily which can be manipulate during the program execution.
- Variables in java is strongly typed hence they all must have a data type followed by an identifiers.

- **Primitive variables:** These variables are created by using primitive data types.
- **Referenced Varibles:** These variables are created by using referenced data types.
- The difference between primitive and referenced variables is “primitive variables stores data directly, where as referenced variables stores reference of the object, not direct values.”
- Reference variable are initialized with object reference the is created and returned by “new” keyword/operator.

DEFINING/INITIALIZING A VARIABLE:

VARIABLE CREATION WITH VALUE IS CALLED DEFINING A VARIABLE.

A VARIABLE CAN BE INITIALIZED AT THE MOMENT IT IS DECLARED, THROUGH THE STATEMENT.

STORING A VALUE IN A VARIABLE AT THE TIME OF ITS CREATION IS CALLED INITIALIZING A VARIABLE.

<ACCESS MODIFIER> <MODIFIER> <DATA TYPE> <VARIABLE NAME>=<VALUE>

ACCESS MODIFIER: OPTIONAL

MODIFIER: OPTIONAL

DATA TYPE: MANDATORY

VARIABLE NAME: MANDATORY

VALUE: MANDATORY

DECLARING A VARIABLE:

VARIABLE CREATION WITHOUT A VALUE IS CALLED DECLARING A VARIABLE.

<ACCESS MODIFIER> <MODIFIER> <DATA TYPE> <VARIABLE NAME>;

ASSIGNING A VARIABLE:

STORING A VALUE IN A VARIABLE AFTER ITS CREATION IS CALLED ASSIGNING A VALUE TO A VARIABLE.

TYPES OF VARIABLE BASED ON CLASS SCOPE:

1. *LOCAL VARIABLES*, PARAMETERS
2. CLASS LEVEL VARIABLES
   A. *STATIC VARIABLES*
   B. *NON-STATIC VARIABLES*

- ***Memory Location of all three variables:***

Local Variables: These variables get the memory location when method is called and their creation statement is executed. They get the memory with respect to method so they are called method variables. Local variables are automatically created when method is executing and are destroyed automatically after method execution is completed, so they are also called auto variables.

Static Variables: These variables get memory location when class is loaded into JVM. They get memory with respect to class name, so they are also called "class variables/fields".

Non-Static Variables: These variables get memory location when object is created using new keyword. They get memory with respect to object, so they are also called "object variables/instance variables/properties/attributes/fields".

Rules of Local variables:

1. Local variables can't be access from another method because its scope is restricted within its method.
2. Local variable should not be accesses without initialization.
3. Local variable must be accessed only after its creation statement.
4. Local variables can't be declared as static it leads as CTE: Illegal star of expression, because local variable should get memory location only if method is called. But if we declare as static, it should get memory at the time of class loading, this is violating contract, so it leads to compile time error.

Rules of Class level variables:

1. We must create class level variables only if we want to access a value throughout the class from all its method.
2. Non-static variables and methods must be accessed with object form static methods else it leads CTE: non-static variable can't be referenced from static context.

Final Variable:

- The class level or local variable the has final keyword in its definitions is called final variable.
- *Rule:* one it is initialized by developer, its value can't be changed. If we try to change its value, it leads to CTE.

Transient Variable:

- The class level variable that has transient keyword in its definition is called transient variable.
- Rule: local variable can't be declared as transient. It leads CTE: illegal start of expression.
- Note: We declare a variable as transient variable to tell to JVM that we do not want to store variable value in a file in object serialization. Since local variable is not part of object, declaring it as transient is illegal.

Volatile variable:

- The class level variable that has volatile keyword in its definition is called transient variable.
- Rule: local variable can't be declared as transient. It leads CTE: illegal start of expression.
- Note: We declare variable as volatile to tell to JVM that we don't want to modify variable value concurrently by multiple threads. If we declare variable as volatile multiple threads are allowed to change its value in sequence one after one.

  Since local variable is not directly accessible by thread, declaring it as volatile is illegal.

Note:

- Static members are meant for storing data and operate that data *common* to all instances of an object.
- Non-static members are meant for storing data and operate that data *separately* and specific to every instance of an object.

Modifiers	Private	protected	public	static	final	abstract	native	volatile	transient	synchronized	strictfp
Local var					✓						
Class level var	✓	✓	✓	✓	✓			✓	✓		
Method	✓	✓	✓	✓	✓	✓	✓			✓	✓
class			✓		✓	✓					✓
Interface			✓			✓					✓

Types of static members and non-static members

Types of static members:

1. Static variables
2. Static methods
3. Static blocks
4. Main method

Types of non-static members:

1. Non-static variables
2. Non-static blocks
3. Non-static methods
4. Constructor

this keyword: this is a non-static final references variable used to store current object reference to separate non-static variables of different objects in non-static context.

Who will placed: compiler will placed at compilation phase in all non-static methods

- **Order of execution of all static members:** First all static variables and static block are executed in the order they are defined from top to bottom, then the main method is executed.
- Static methods are executed in the order they are called, not in the order they are defined.
- **Assigning a static variable with local variables or params:** if a local variable or parameter is same with static variable name then we must use class name in assigning static variable with that local variable or parameter else the modification is stored in that local variable or parameter not in static variables.
- **Static block:** static block is a class-level nameless block that contains only static keyword in its prototype.
- It is used to execute logic only at the time of class loading.

1. Initializing static variables.
2. Registering native libraries
3. To know classes loading order etc…

- **Order of execution of all non-static members:** when an object is created first all non-static variables and non-static block are executed in the order they are defined from top to bottom, then the invoked constructor logic is executed.
- Non-static methods are executed only if they are invoked from any of the static or non-static members.
- **Non-Static block:** it is a class-level block which doesn't have any prototype.
- We should define non-static block to execute some logic only at the time of object creation irrespective of the construction of the constructor used in object creation.
- NSB is executed automatically by JVM for ***every object creation***.

Exception

- The aim of exception handling is to build robust application.
- In any programming language when we write a programme we get 3 types of errors:

1. Compile time Errors
2. Logical Errors
3. Runtime Errors

- Compile time errors:
  - These errors occurs at compilation time due to syntaxes are not followed by programmers.
  - These errors solved by programmer at development level.
- Logical Errors:
  - These errors occurs during execution/runtime due to misinterpretation or wrong representation of logic.
  - These errors always generate inconsistence or wrong result or they must be solve by programmer at development time.
- Runtime Errors:
  - These error occur at runtime/execution time due to invalid input enter by application user at implementation level (an application being used by client organization).
  - Industry is highly recommended to generate user friendly error messages where application user can understand what mistake committed at the time of entering the inputs.

Points to be remembered:

1) When the application user enters invalid input then we get runtime error.
2) Runtime error of any programme use System/Technical error message.
3) **Definition of Exception:** "Runtime errors of java programme are known as exception. Exception always generates System error message."
4) Whenever an exception occurs in the java programme, programme execution is abnormally terminated, CPU control comes out of the programme flow and JVM generates system error message which is not a recommended process. Industry is highly recommended to generate user friendly error message instead of generating system error messages by using exception handling concept.
5) **Definition of Exception:** "The process of converting system error messages into user friendly error messages is known as exception handling."

Types of Exception:

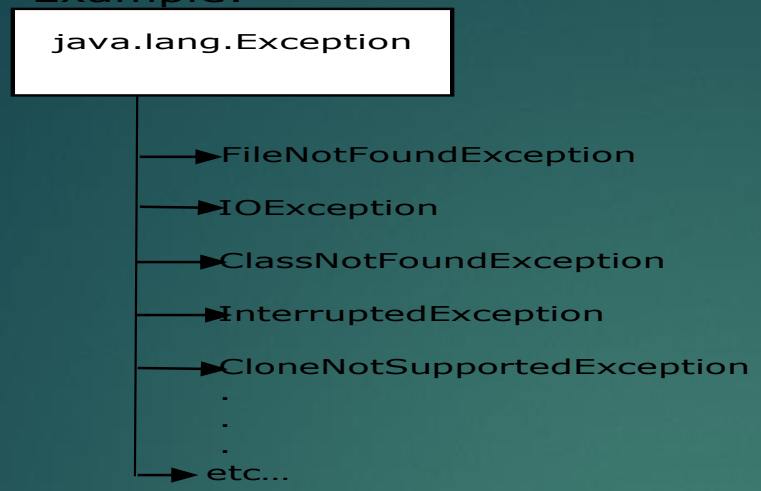
1) Checked Exception
2) Unchecked Exception

Checked Exception: "Checked exception are those which are the direct subclasses of java.lang.Exception."

OR

"The exception which are occurring at runtime will be showing them as error at compile time."

Example:

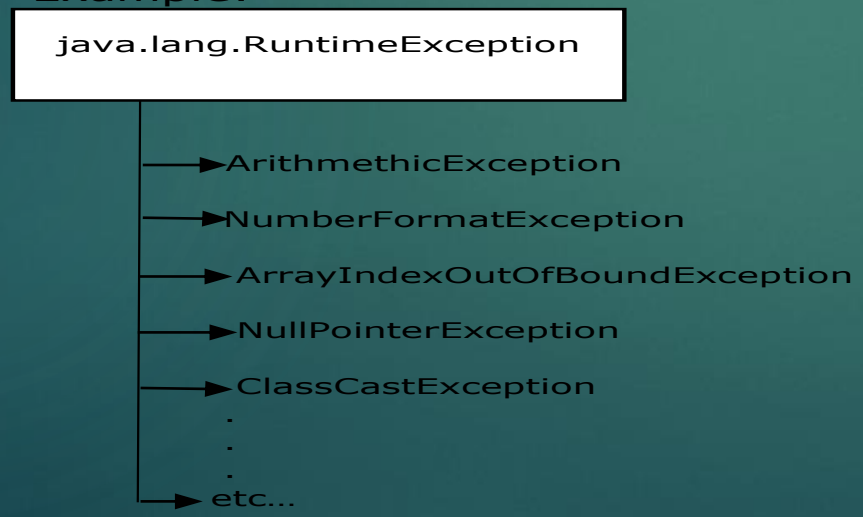


- **Unchecked Exception:** "unchecked exception are those which are the direct subclasses of java.lang.RuntimeException."

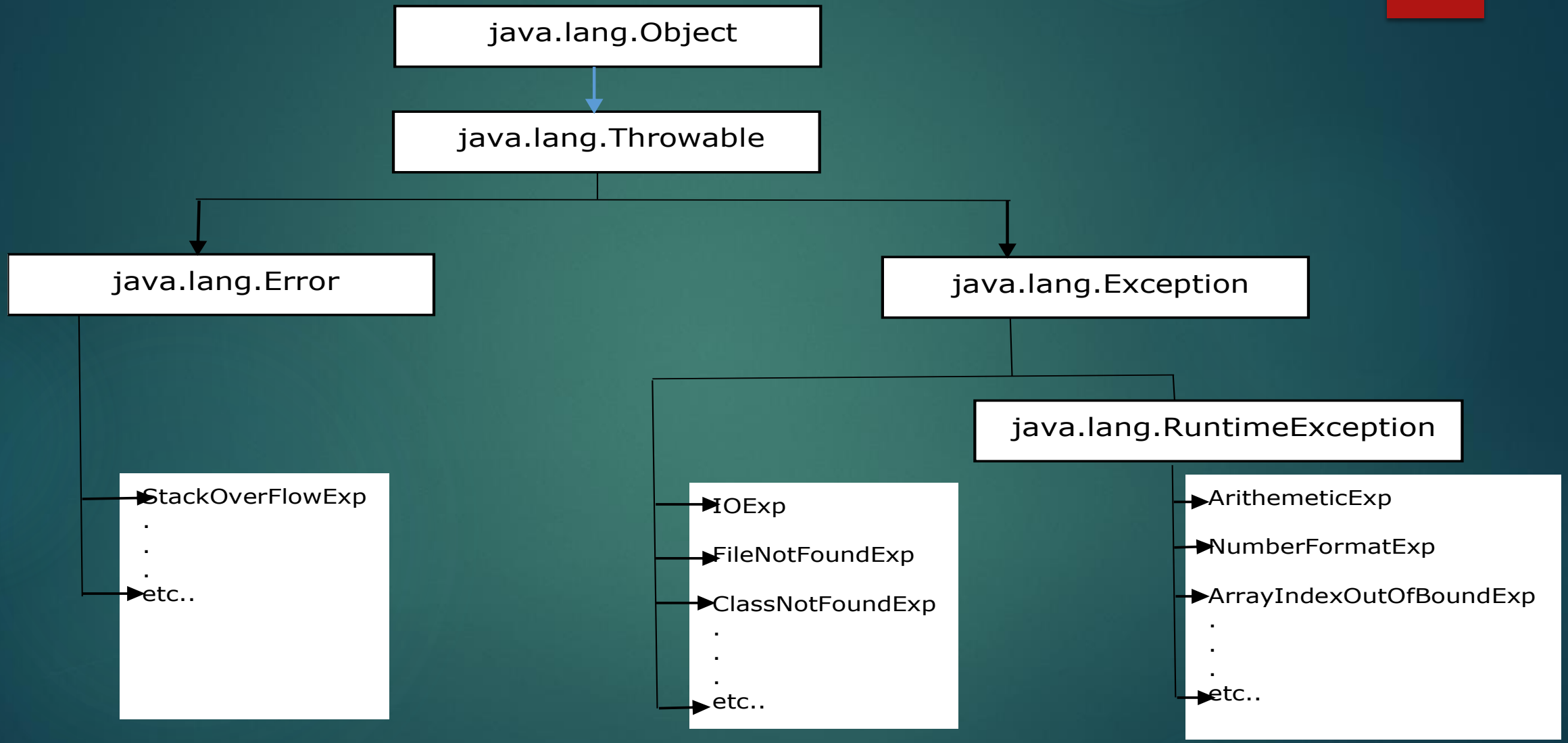
OR

"The exception which are occurring at runtime will be showing them as errors as runtime only."

Example:



- **Note:** Even though java.lang.RuntimeException is the subclass of java.lang.Exception, it should not be treated as checked exception and it is a one of the dedicated super class for all the unchecked exception.



1) java.lang.Object class is one of the implicit predefined super class for all the classes in java and it provides garbage collection facility to its subclasses for collecting unused memory space.
2) java.lang.Throwable is one of the predefined super class for all the exception in Java and the purpose of this class is to check which type of exception occurs in java programme (either ASE or SE).
3) java.lang.Error is always used for dealing with asynchronous exception and it is a super class for all asynchronous exception.
4) java.lang.Exception is one of the super class for all synchronous checked exception.
5) java.lang.RuntimeException is one of the super class for all synchronous unchecked exception.

- **Handling the Exception:**

- Handling the exception is nothing but converting system error message into user friendly error message.
- As a part of handling the exception we have 5 keywords, they are:

1. try
2. catch
3. finally
4. throws
5. Throw

try block:

- This is the block in which we write the block of statements, this is a block of statements which will provide exception at runtime. In other words this block always contains set of problematic statements which will generate exception at runtime. This block is also known as exception monitoring block.
- Whenever an exception occurs in try block, JVM control comes out of the try block and JVM will execute appropriate catch block.
- After executing an appropriate catch block, JVM control never goes to try block to execute the rest of the statements even though we use return statement in catch block.
- Programmatically each and every try block must be followed by catch block otherwise we get compile time error in other words intermediate statements are not permitted between try and catch blocks.
- Each and every try block must contain at least one catch block and industry is highly recommended to write multiple catch blocks for generating multiple user friendly error messages for making java application robust.
- One try block can contain another try block i.e. nested try blocks can be permitted.

catch block:

- This is the block which will contains block of statements which will provide user friendly error messages instead of generating system error messages. In other words, catch block will supress system error messages and generates user friendly error messages.
- Writing catch block is nothing but handling the exception, JVM will execute an appropriate catch block if an exception occurs in try block.
- At any point of time even though their exist many number of catch blocks, JVM can execute only one catch block depends on the type of exception occurs in the try block.
- If at all we write finally block, it must be written immediately after catch block or every catch block immediately followed by finally block or intermediate statements are not permitted between catch and finally block.
- In the catch block as a java programmer we declare an object of exception subclass and internally JVM will reference an object of exception sub class.
- In some of the circumstances one catch block can contain try and catch block.

- **finally block:**
  - It is the block in which we write block of statements which are used for relinquish (release or close) the resource/files/DB which are obtained in try block.
  - Finally block will execute compulsory.
  - Writing finally block is optional.
  - For per java programme, It is highly recommended for java programmer to write only one finally block for releasing the resource which are appending in try block.
  - In some of the circumstances, in the finally block one can write a try and catch block.

Throws keyword:

- *"Purpose of throws keyword is that it will express or describes the exceptions occurring in common method body."*

  OR

- *"Throws is a keyword which gives an indication to the specific method to place the common exception method under try and catch block for generating user friendly error message."*

Throw keyword:

- *It is keyword used for hitting/generating the exception which is occurring as a part of method body.*

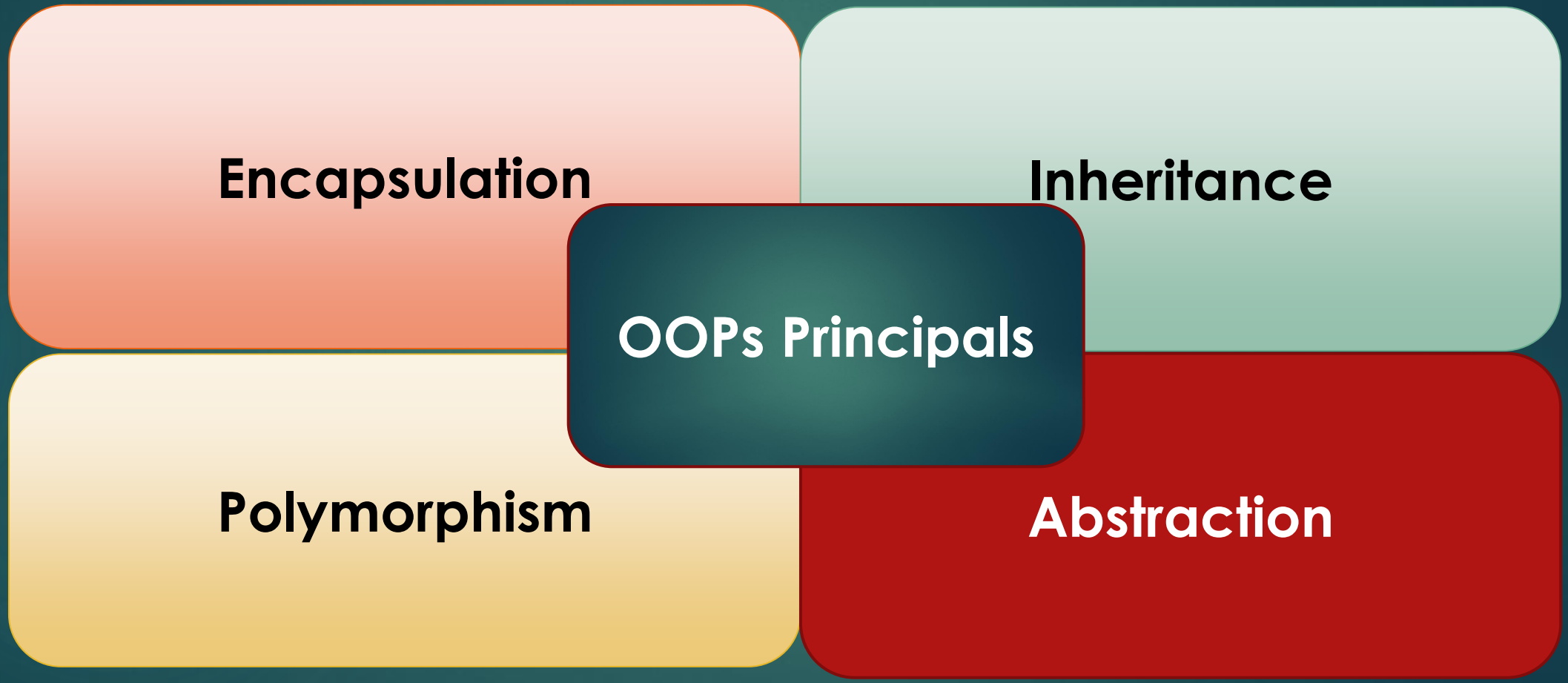
  OR

- *Throw is keyword which is used for carrying the created exception from the method body and handover to the throws keyword.*

Throws	Throw
Throws is keyword which gives an indication for placing the common exception method within try-catch blocks for generating the user friendly error message.	Throw is a keyword used for hitting or generating the exception which is occurring as a part of method body.
The place of using throws keyword is always as a part of method heading.	The place of using throw keyword is always as a part of method body.
When we write throws keyword as a part of method heading it is optional to write throw keyword as a part of method body. This is applicable in case of re-throwing the exception.	Irrespective of type of exception we hot by using throw keyword as a part of method body, it is recommended to the Java programmer to write the throw keyword as a part of method body.

What is OOPs?

- OOPs stands for Object Oriented Programming language. It is a programming paradigm that relies on the concept of classes and object and the following OOPs principal:



Classes and Objects

- The class is one of the basic concepts of the OOPs which is group of similar entities.
- A class is a blue print from which individual objects are created.
- Class contains properties of real objects and the functions that real objects will perform.

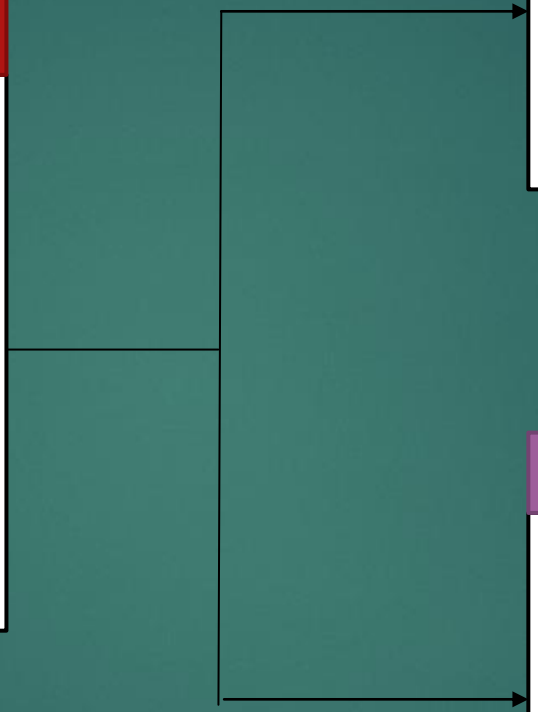
Example: If we have class called “DreamCompany” it could have objects like Amazon,Microsoft ,Facebook, Google etc.. Some might have Amazon as their dream company, and some might have Microsoft. The class property can have companyName, location, package etc… While the methods may be performed with these companies are doDevelopment, doTesting, doAutomation, doSupport etc.

- An object can be defined as an instance of a class.
- Object of the class will have some properties and functions as defined in the class.
- Objects is one of the Java OOPs concepts which contains both the data and the functions which operates on the data.

Class: DreamCompany

Attributes:
- companyName:String
- location: String
- package: Integer

Methods:
- doDevelopment(): void
- doTesting(): void
- doAutomation: void



Object: Amazon

Attributes:
- companyName:"Amazon"
- location: "Bangalore"
- package: 30LPA

Object: Google

Attributes:
- companyName: Google
- location: Hyderabad
- package: 50LPA

Encapsulation

- Encapsulation is process of wrapping code and data together into a single unit (this unit is called Class).
- This is to prevent the access to the data directly, the access to them is provided through the functions of the class. It helps in data hiding.

Example:

1. Consider an example of a capsule. Basically a capsule encapsulates several combinations of medicine which are required. Binding everything that is necessary within a single unit.

2. Suppose you have an account in the bank what if third person is able to see your Bank balance? So, would you like it?

Obviously NO!

So, concept of classes ensures safety of your account balance (declare the balance variable as private), so that nobody can see your account balance. That is data hiding!

Advantages of Encapsulation:

- Data Security: Data and functions operating on data are binded together.
- Data Integrity: it means data is no modified by any object of outside class.

Abstraction

- Abstraction refers to the act of representing essential features without including the background details or explanations.
- It is basically hiding unnecessary details from the user.
- It enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

- Advantages of Abstractions:
- Consider the previous example, if brakes implementations changes form drum brakes to disk brakes (different types of brakes), then the driver need not worry and will handle the brakes the same ways.
- In terms of coding, if you are using a library to perform a specific task (consider simple System.out.println() function, even if Java internally changes the implementation of this function, we will not need to change our code and can continue using the same function (given the function signature is same)

Real Life Example of Abstraction

Encapsulation vs Abstraction

Abstraction	Encapsulation
Abstraction is the methos of hiding the unwanted information.	Encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
Abstraction focus on the observable behavior of an object	Encapsulation focus on the implementation.
Abstaction is when you look a class from outside	Encapsulation is when you look a class from inside.

Inheritance and Polymorphism

- Inheritance is a mechanism in which one class acquires the property of another class.
- Real world Example:
  - Classic example of inheritance is a child acquiring behavior from parents.
- Advantage of inheritance
  - We can reuse the fields and methods of the exiting class. Hence inheritance facilitates Reusability and is an important concept of OOPs.

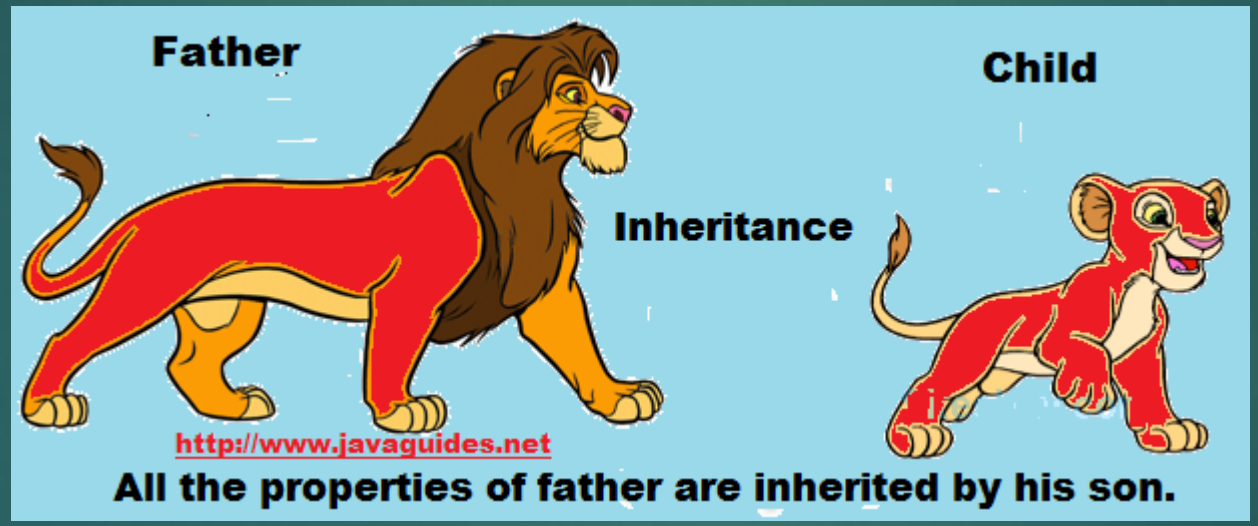
Polymorphism:

- The word polymorphism means having many forms.
- In simple worlds, we can define polymorphism as the ability (of anything) to take more than one form
- Same thing/object will behave differently in different situations.

- Real world example:
- If someone calls you name, you will respond differently to the same name calling depending on who is calling you. For instance, if your friend call you, you will respond “hey buddy”, if mother calls you, you will respond “yes mom” and if your teacher calls you , you will respond “yes sir/mam!”

Classic Inheritance Example

- The simple example of inheritance that we see in our daily life is child and parents. The children look like parents or grandparents. This means they inherit the properties of the parents/grandparents.



Benefit of inheritance?

- Re-usablility of an existing thing/property or any materialistic thing.
  - Using existing stuff in the house or using an old car/building as it is.
- Adding extra functionality to any existing physical entity or modifying it.
  - Example: renovating the old inherited house from parent.

Defining inheritance:

- The term inheritance refers to the fact that one class can inherit part or all of its structure and behavior from another class.
- The class from which features (data + methods) are inherited is known as base/parent/super class.
- And class that is inheriting is known as derived/child/subclass.
- Another way to say that deriving features from base class to derives class is known as inheritance.
- A subclass can add to the structure and behavior that it inherits. It can also replace or modify inherited behavior.

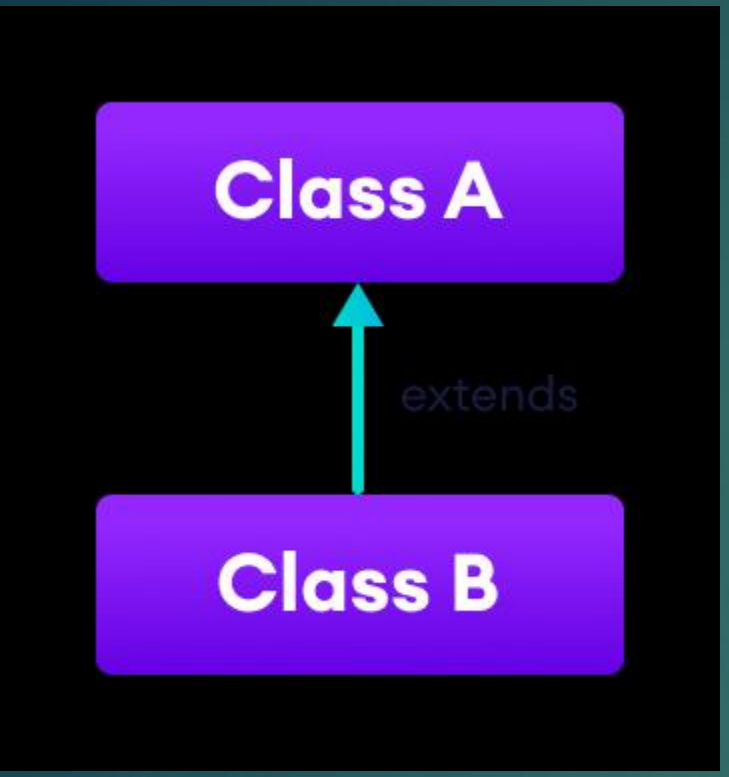
Syntax:

```
class BaseClass{
    //fields + methods
}

class DerivedClass extends BaseClass{
    //fields + methods
}
```

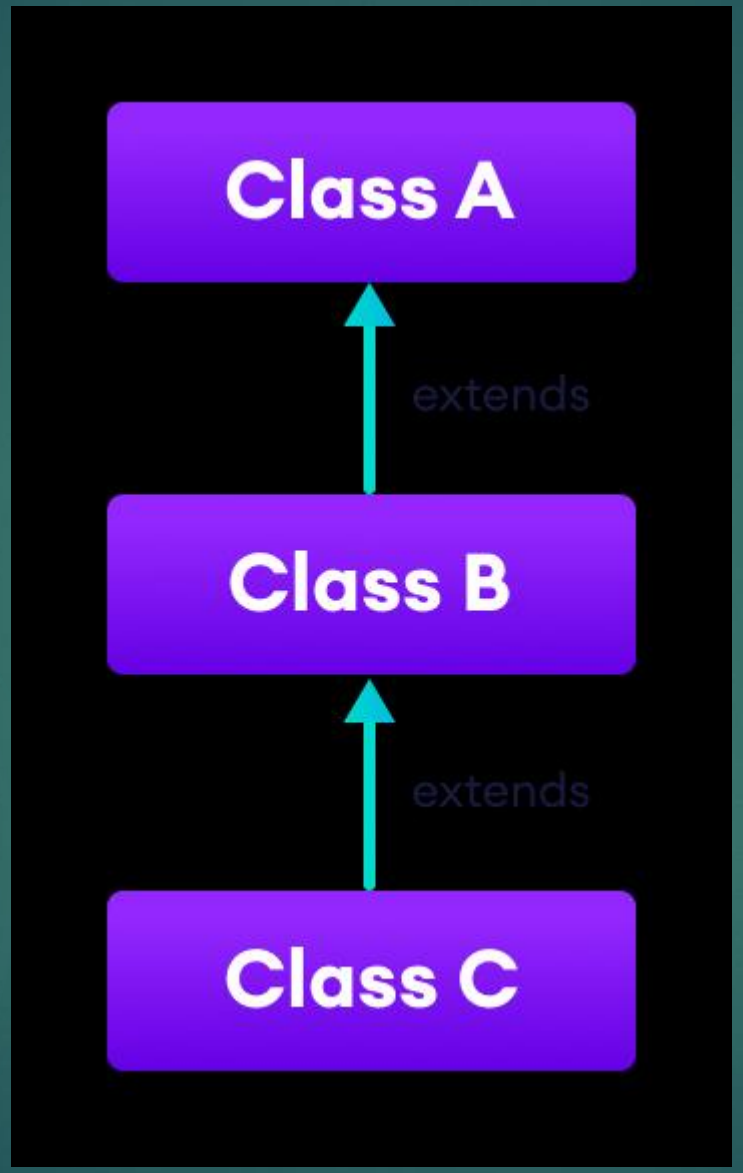
//Here BaseClass is already an existing class. When DerivedClass is created it inherits BaseClass using "extends" keyword.

Advantages of Inheritance

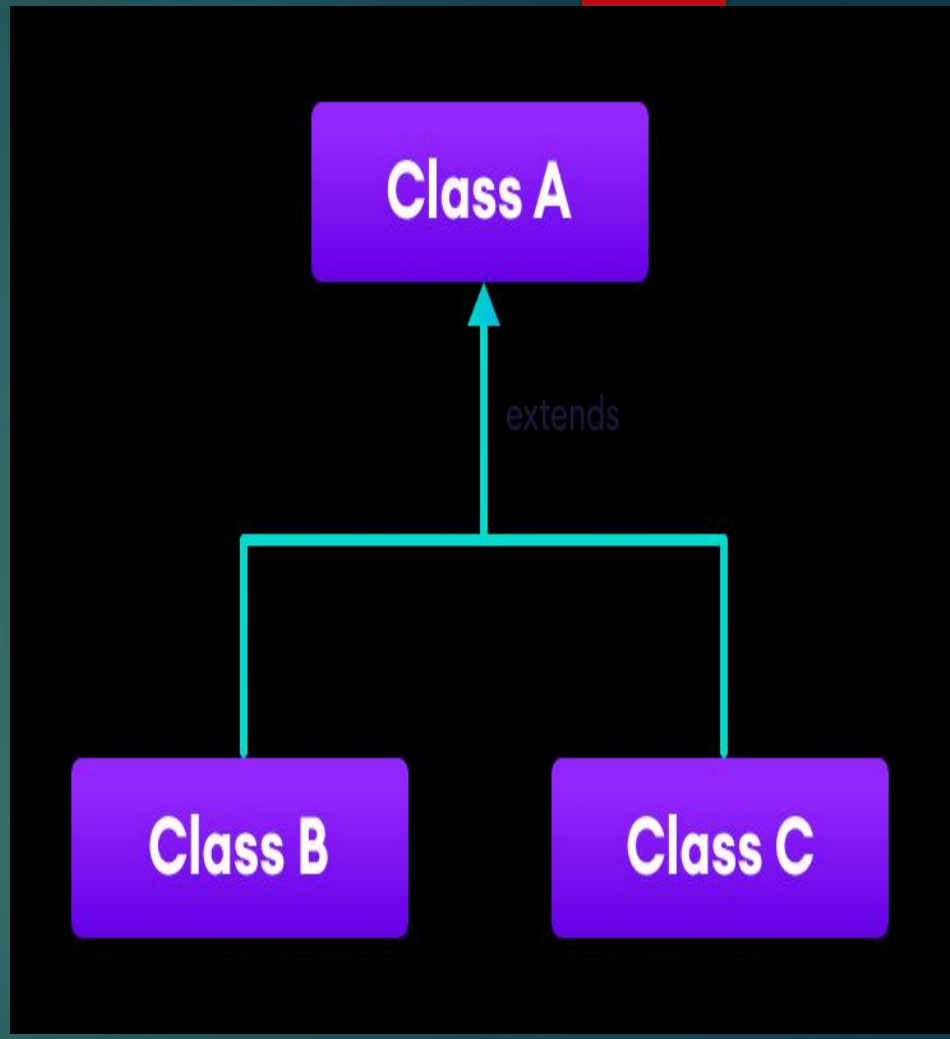
- Inheritance provides the idea of reusability i.e. code (fields and methods) once written can be used again & again in a number of new classes.
- Method overriding and Runtime polymorphism.
- Application development is fast because of reusability of code.
- Classes which is the part of java libraries can be used as base class so development time is also reduced.



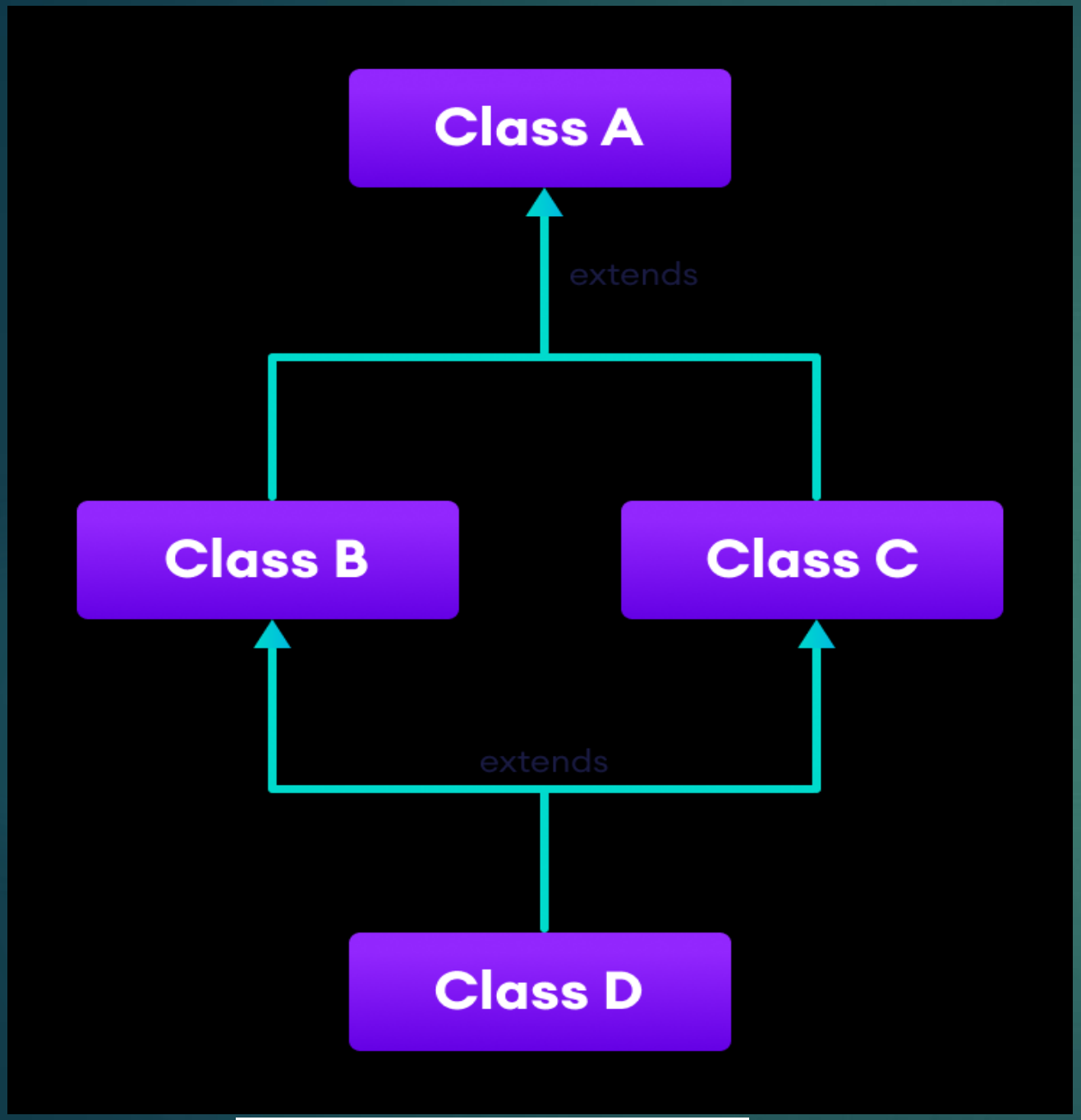
Single Level Inheritance



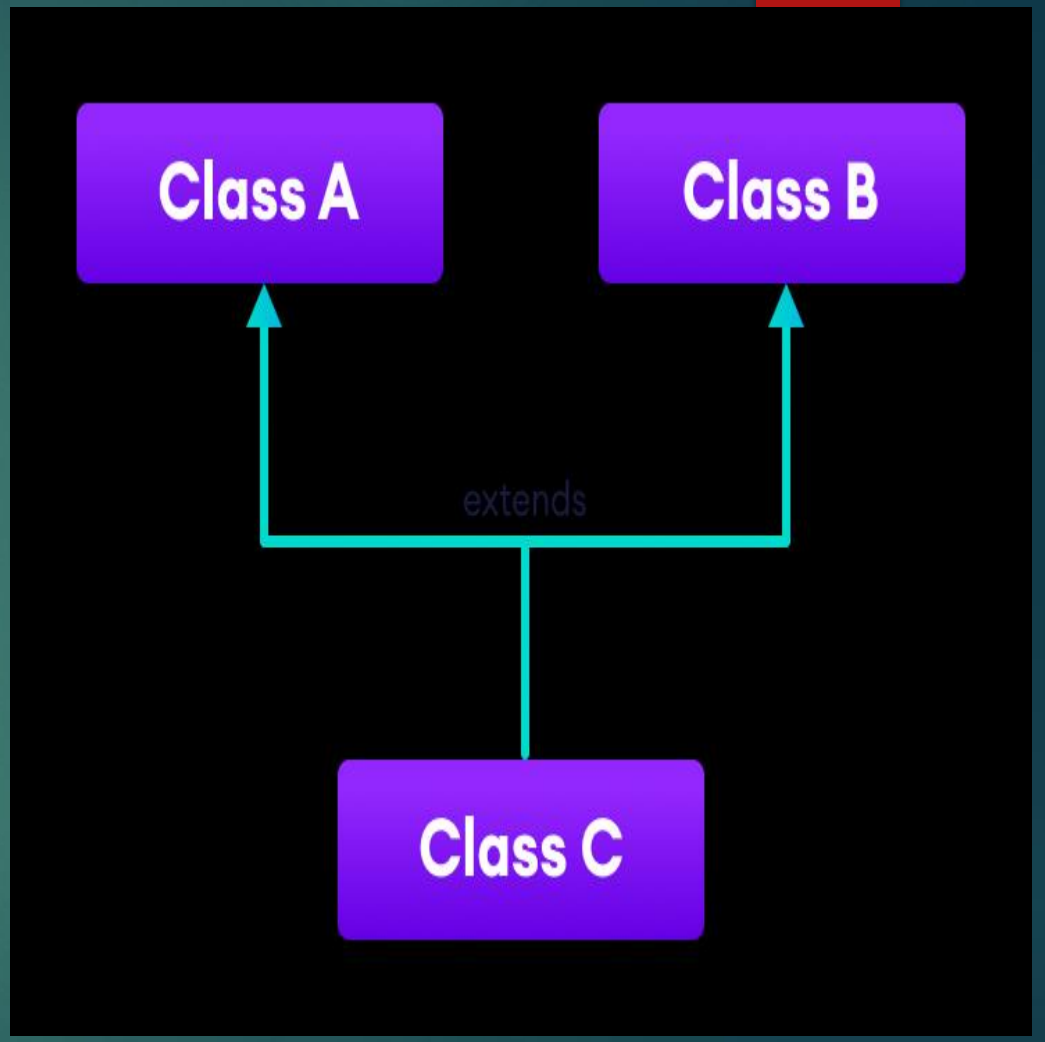
MultiLevel Inheritance



Hierarchical Level Inheritance



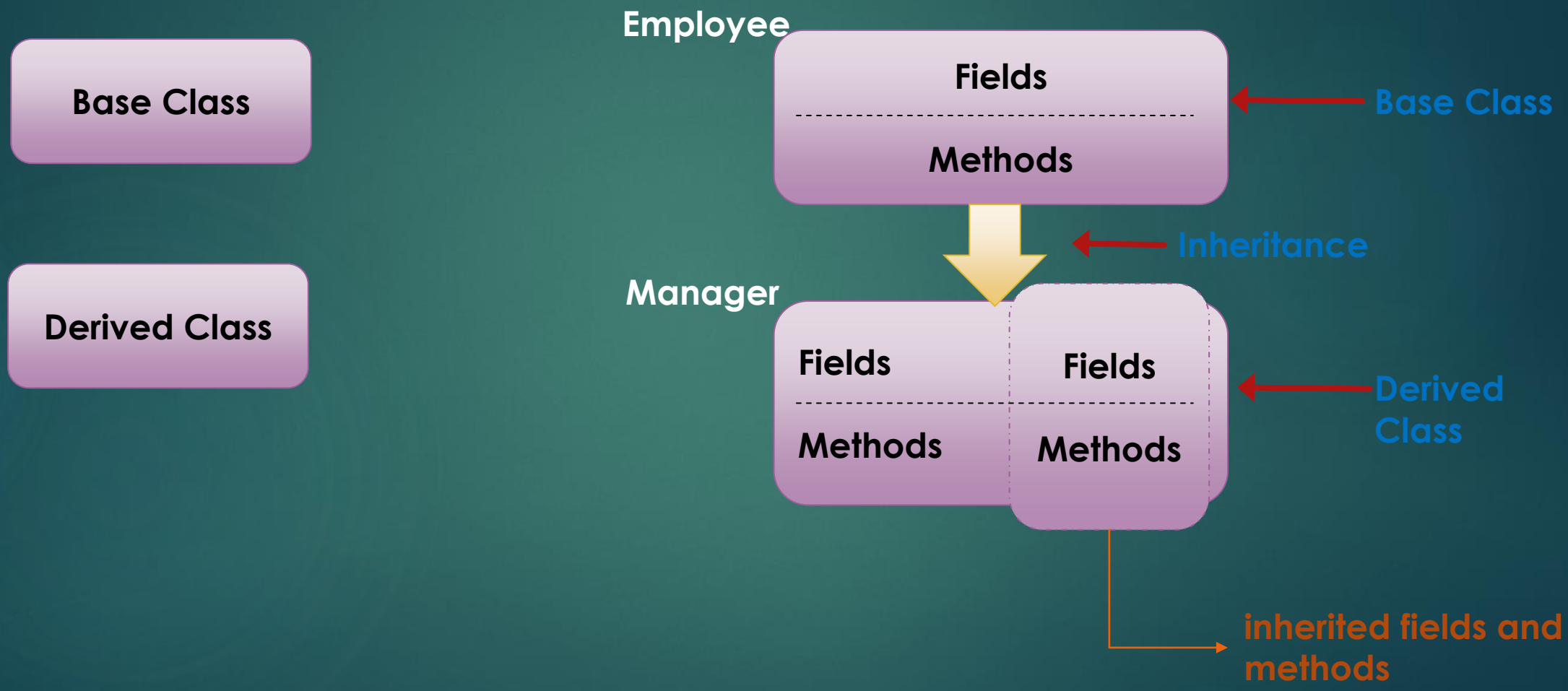
Hybrid Inheritance



Multiple Inheritance

Single Level Inheritance

- In single level inheritance we have just one base class and one derived class. It is represented as:

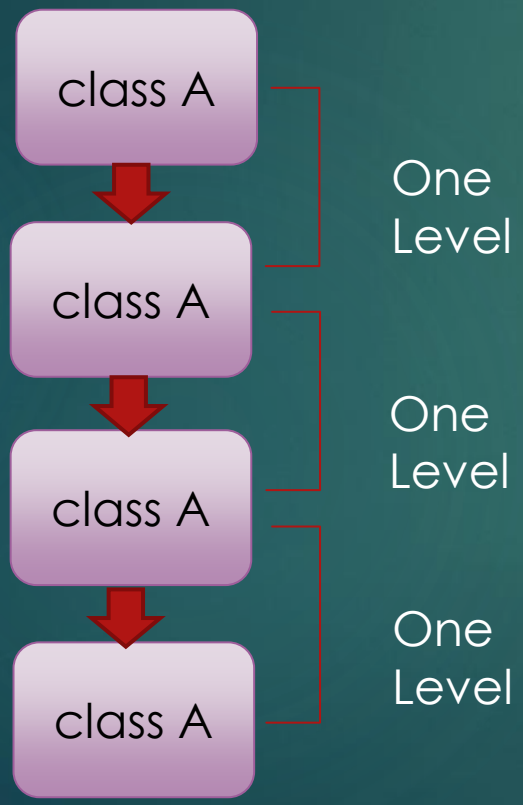


Syntax of single level inheritance

```
class BaseClass{
    //fields + methods
}

class DerivedClass extends BaseClass{
    //fields + methods
}
```

Multilevel Inheritance

- In multilevel inheritance we have one base class and one derived class at one level.
- At the next level the derived class becomes the base class for the next derived class and so on.

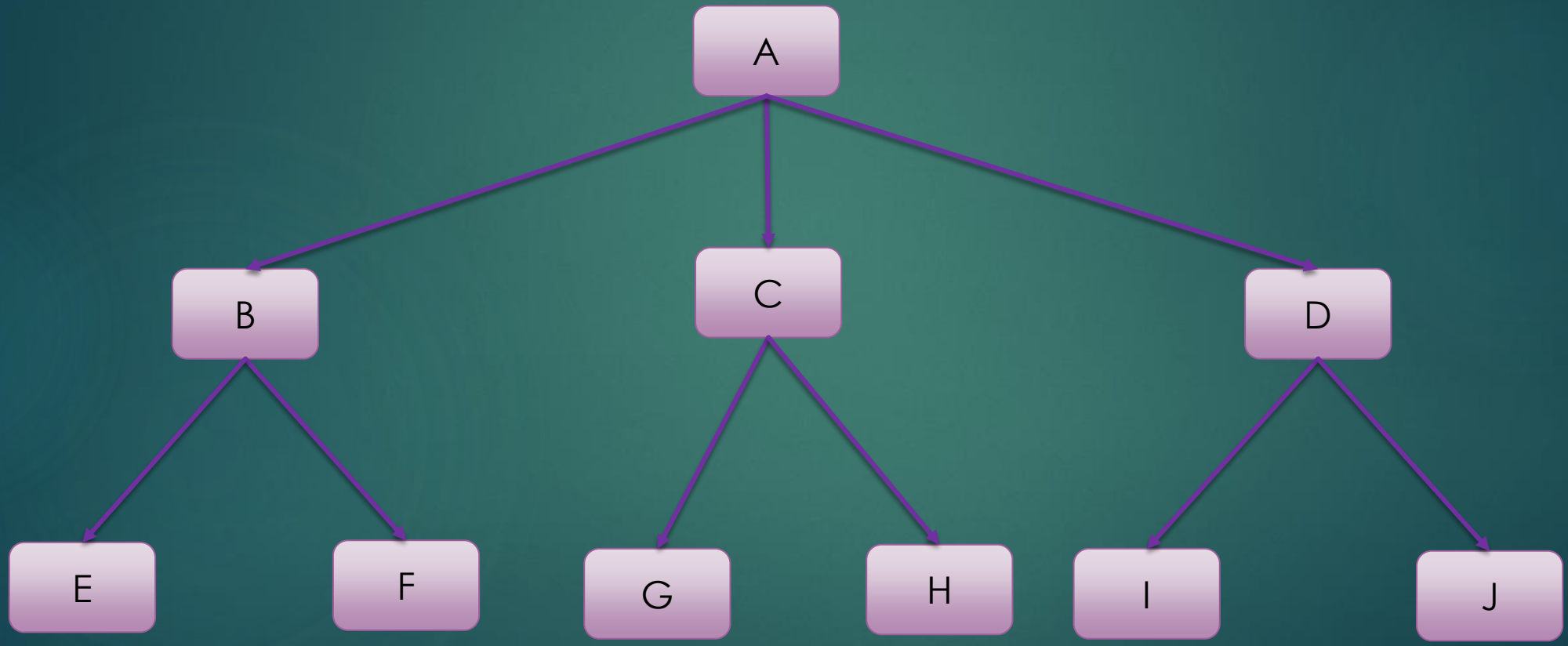


Syntax of Multilevel inheritance

```
class A{
    //fields + methods
}

class B extends A{
    //fields + methods
}
class C extends B{
    //fields + methods
}
class D extends C{
    //fields + methods
}
```

Hierarchical Inheritance

- Multiple classes share the same base class.
- Multiple classes inherit the properties of one common base class.
- The derived classes again may become the base class for other classes.



Syntax of Hierarchical inheritance

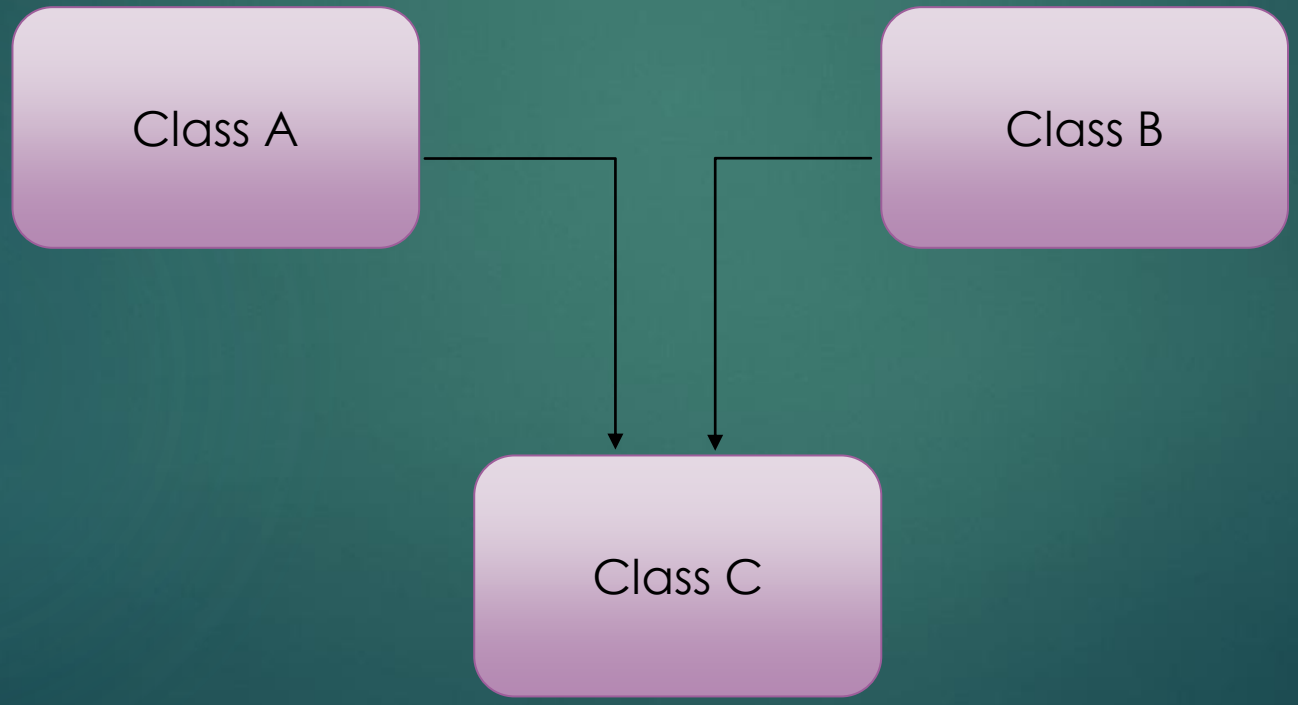
```
class Base{
        //fields + methods
}

class Derived1 extends Base{
        //fields + methods
}
class Derived2 extends Base{
        //fields + methods
}
```

Real life example of Hierarchical inheritance

- Example 1: University with a number of affiliated engineering colleges.
  - All engineering colleges must have UGC affiliations.
- An Animal class can be a root class. Its children can be herbivorous and carnivorous animals.
- Autonomous University with various departments.

Multiple Inheritance

- In a multiple inheritance a child can have more than one parent i.e. a child can inherit properties from more than one class.
- This is as shown (but not true in java).



Syntax of Multiple Inheritance (Not Supported in Java)

```
class A{
        //fields + methods
}

class B {
        //fields + methods
}
class C extends A,B{
        //fields + methods
}
```

Note: Multiple inheritance through classes is not supported in java but through interfaces its possible.

Hybrid Inheritance

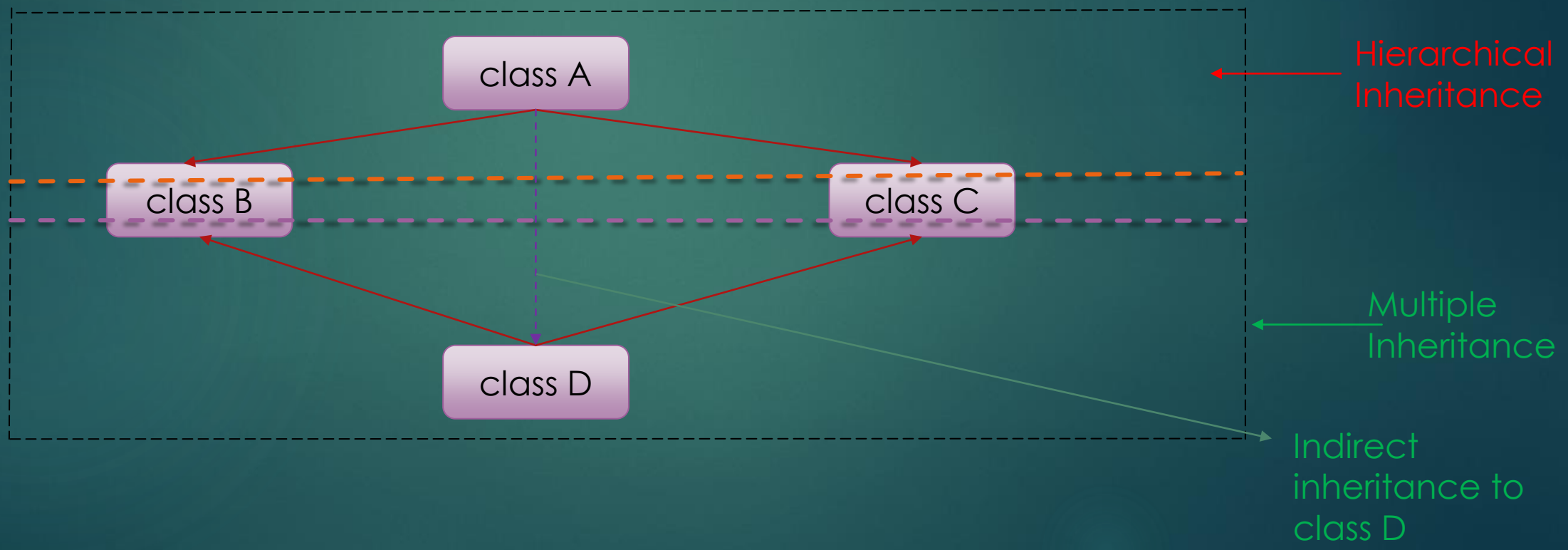
- Mixture of multiple types of inheritance
- Let's see the 3 types of Hybrid inheritance

Type-1:

It is hybrid because it contains hierarchical and multiple inheritance.

It's not possible to have the above type of inheritance as java doesn't support multiple inheritance through classes.

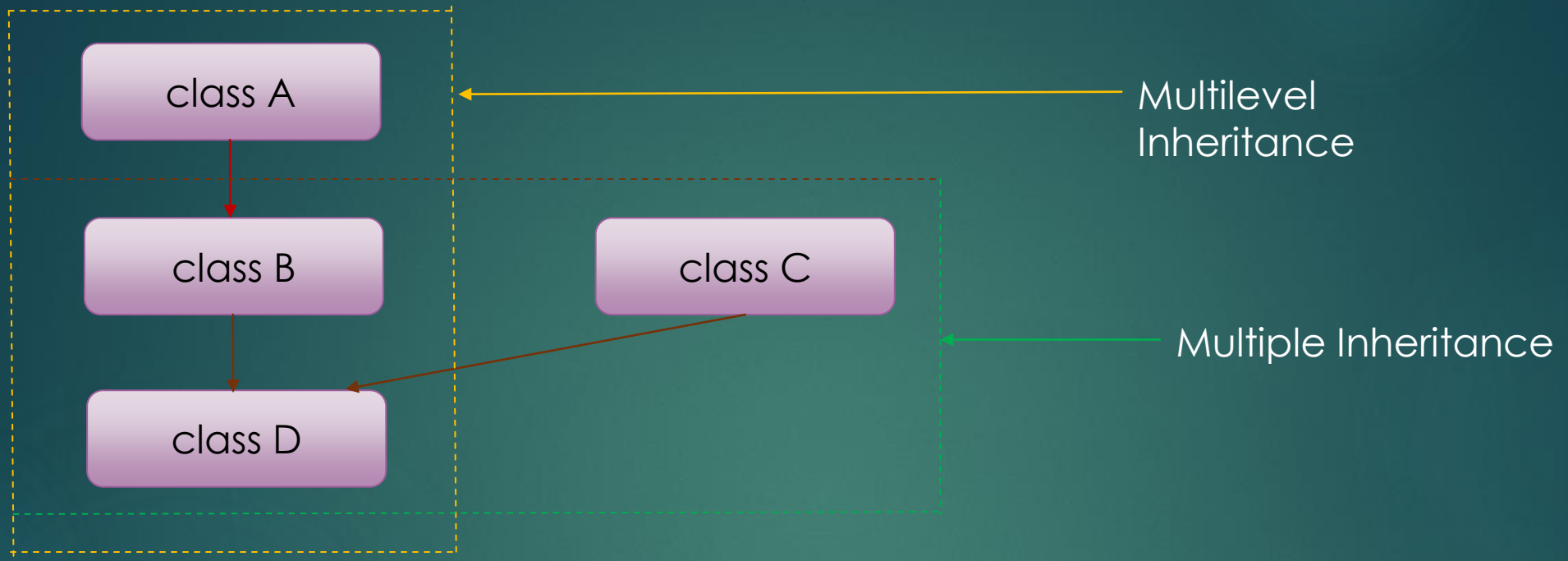
It's possible to have above hybrid inheritance through interfaces.



Diamond Problem

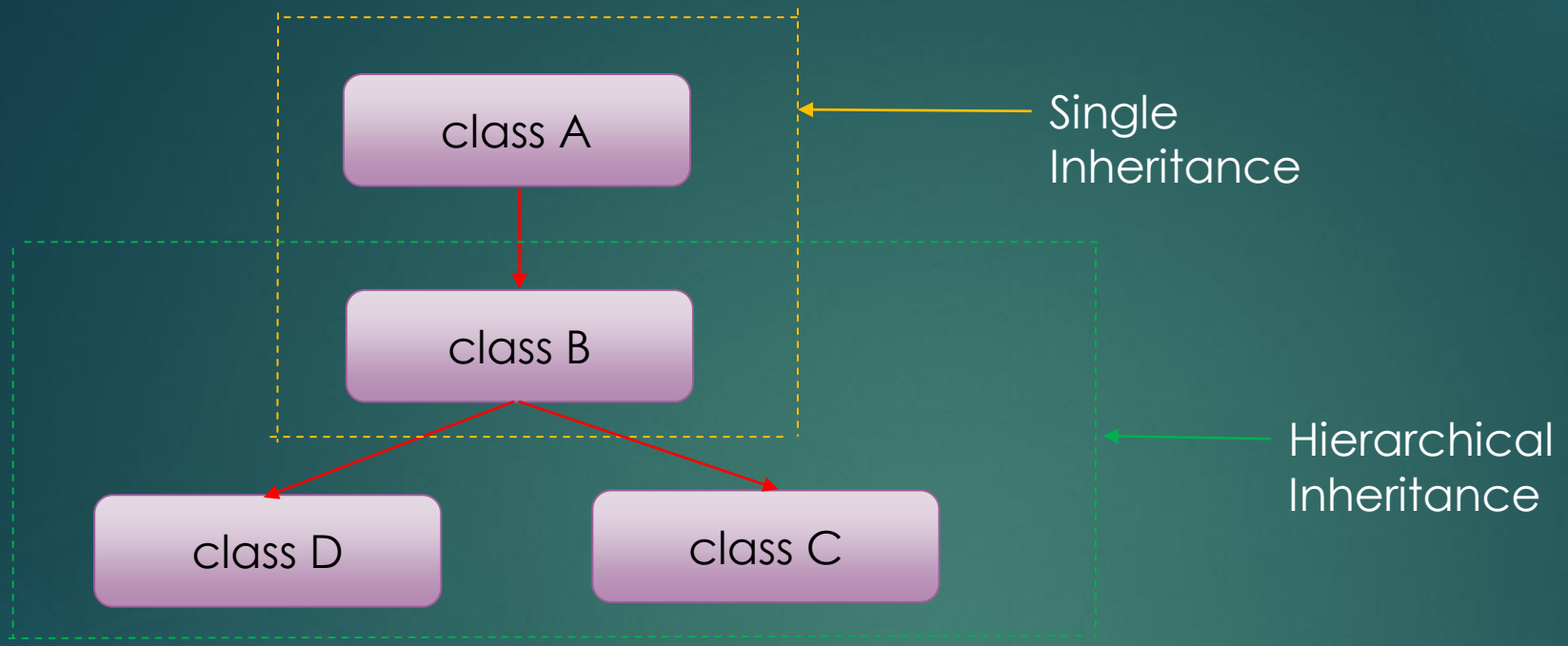
- The above figure looks like the shape of a diamond.
- A method in class A in inherited in class D twice: one from B and second through C.
- This may cause serious issues and create ambiguity as all members of A are inherited twice.
- Because of this, Java doesn’t support multiple inheritance as well.

Type-2:



- It’s hybrid because it combines multilevel and multiple inheritance.
- It’s not possible to have the above type of inheritance as java doesn’t support multiple inheritance through classes.
- It is possible to have above hybrid inheritance through interfaces.

Type-3:



- It’s hybrid because it combines single level and hierarchical inheritance.
- It’s possible to have the above type of inheritance through classes and interfaces both.

OOPs Definition

- **Encapsulation:** The process of creating a class by hiding internal data from the outside world/outside class members; and accessing only through publicly exposed methods is known as encapsulation.
- **Inheritance:** The process of creating a class to reuse exited class members using our class name or object is class inheritance. It can also be defined as it is process of obtaining one object property to another object.
- Types of inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance
5. Multiple Inheritance

- **Polymorphism:** It is process of defining a class with multiple methods with same name with different implementations is called polymorphism.

We can develop polymorphism by using

- Method overloading: can be achieved based on types/list/order of method signature
- Method Overriding

- Types of Polymorphism:

1. Compile time polymorphism
2. Run time polymorphism

1. **Compile time Polymorphism/ Static Binding /Early Binding:** when a method is invoked, if its definition which is bind at compilation time by compiler is only executed by JVM at runtime, then it is called compile-time polymorphism.

Note: Static methods, Overloaded Methods and non-static methods which is not overridden in subclass are come under compile time polymorphism.

2. **Runtime Polymorphism/ Dynamic Binding / Late binding:** when a method is invoked, if its definition which is bind at compilation time by compiler is no executed by JVM at run-time, instead if it is executed from the subclass based on the object stored in the reference variable is called run-time polymorphism.

Note: only non-static methods are come under run time polymorphism. Private non-static methods and default non-static methods from outside package are not overridden. So these methods call comes under compile time polymorphism.

Method Hiding, Overriding and Overloading:

- Redefining super class static method in subclass with same prototype is called "method hiding"
- Redefining super class non-static method in subclass with same prototype is called "method overriding".
- Defining new method with the existed method name but different parameters type/list/order is called "method overloading"

Note: Super class method is called overridden methods and subclass method is called overriding method.

- **Abstraction:** The process of defining a class by providing necessary details to call object operation by hiding or removing its implementation details is called Abstraction.
- We can achieve abstraction in two ways:

1. Abstract class
2. Interface

Thread

- Threads consumes CPU in best possible manner, hence enables multi processing. Multi threading reduces idle time of CPU which improves performance of application.
- Thread are light weight process.
- A thread class belongs to java.lang package.
- We can create multiple threads in java, even if we don’t create any Thread, one Thread at least do exist i.e. main thread.
- Multiple threads run parallely in java.
- Threads have their own stack.
- Advantage of Thread : Suppose one thread needs 10 minutes to get certain task, 10 threads used at a time could complete that task in 1 minute, because threads can run parallely.

Thread State / Thread Life Cycle

1) New : When instance of thread is created using new operator it is in new state, but the start() method has not been invoked on the thread yet, thread is not eligible to run yet.

- Thread object is considered alive but thread is not alive yet.

2) Runnable : When start() method is called on thread it enters runnable state.

- As soon as Thread enters runnable state it is eligible to run, but not running. (Thread scheduler has not scheduled the Thread execution yet, Thread has not entered in run() method yet)
- A thread first enters the runnable state when the start() method is invoked, but a thread can also return to the runnable state after either running or coming back from a blocked, waiting, or sleeping state.
- Thread is considered alive in runnable state.
- Thread is in Runnable pool.

3) Running :

- Thread scheduler selects thread to go from runnable to running state. In running state Thread starts executing by entering run() method.
- Thread scheduler selects thread from the runnable pool on basis of priority, if priority of two threads is same, threads are scheduled in unpredictable manner. **Thread scheduler behaviour is completely unpredictable**.
- When threads are in running state, **yield()** method can make thread to go in Runnable state.

4) Waiting/blocked/sleeping :

- In this state a thread is not eligible to run.
- Thread is still alive, but currently it's not eligible to run. In other words.

How can Thread go from running to waiting state ?

- By calling **wait()** method thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

How can Thread return from waiting to runnable state ?

- Once **notify() or notifyAll()** method is called object monitor/lock becomes available and thread can again return to runnable state.

How can Thread go from running to sleeping state ?

- By calling **sleep()** method thread go from running to sleeping state. In sleeping state it will wait for sleep time to get over.

How can Thread return from sleeping to runnable state ?

- Once specified **sleep time is up** thread can again return to runnable state.
- **Suspend()** method can be used to put thread in waiting state and **resume()** method is the only way which could put thread in runnable state.
- Thread also may go from running to waiting state if it is waiting for some I/O operation to take place. Once input is available thread may return to running state.

5) Terminated (Dead) : A thread is considered dead **when its run() method completes**.

- Once thread is dead it cannot be started again doing so will throw runtimeException i.e. IllegalThreadStateException.
- destroy() method puts thread directly into dead state.

Inter-thread communication

wait():

- When we call wait method on the object then it tell threads to give up the lock and go to sleep state unless and until some other thread enters in same monitor and calls notify or notifyAll methods on it.

notify():

- When we call notify method on the object, it wakes one of thread waiting for that object. So if multiple threads are waiting for an object, it will wake of one of them. Now you must be wondering which one it will wake up. It actually depends on OS implementation.

notifyAll() :

- notifyAll will wake up all threads waiting on that object unlike notify which wakes up only one of them.Which one will wake up first depends on thread priority and OS implementation.

Note about synchronized block :

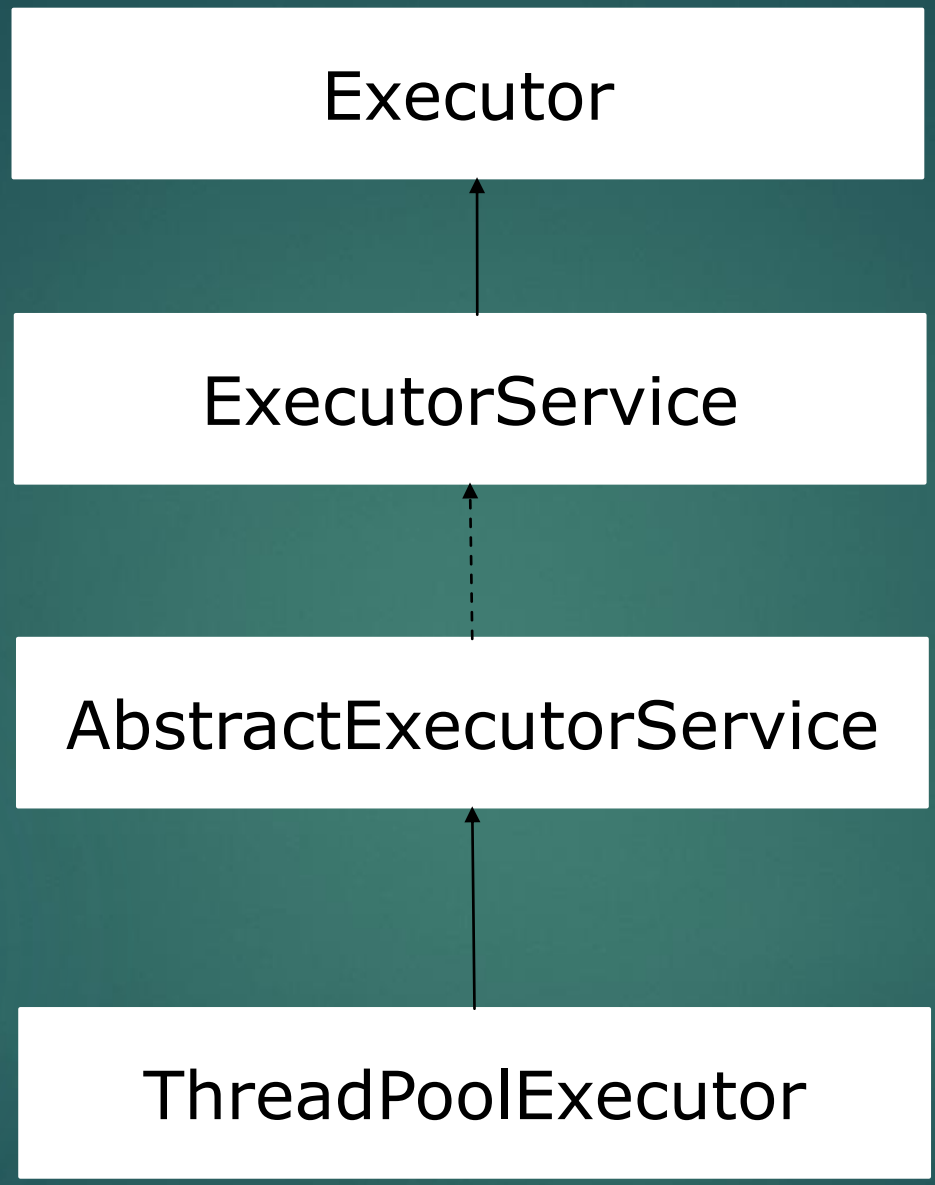
- Only one thread can enter at a time in synchronized block
- A thread required lock on the object to enter in synchronized block.
- If Thread A want to enter in synchronized block then Thread A has to wait for Thread B to release it.

ThreadPool

- Java 5 has introduced new concurrent API called "Executor frameworks" to make programmer life easy. It simplifies design and development of multi-thread applications. It consists of mainly Executor, ExecutorService interface and ThreadPoolExecutor class which implements both interfaces ***i.e. Executor and ExecutorService***. ThreadPoolExecutor class provide the implementation of thread pool.
- A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.

Why do we need Executor framework?

- When we create a simple multithreading application, we create Runnable objects and construct Thread object using Runnable, We need to create, execute and manage thread. It may be difficult for us to do that. Executor Framework does it for you. It is responsible for creating, executing and managing the thread and not only this, it improves the performance of the application too.
- When we follow ***task per thread policy***, we create a new thread for each task then if system is highly overloaded, we will get out of memory error and your system will fail. If you use ThreadPoolExecutor , we won't create thread for new task. You will assign task to a limited number of threads once thread completes one task, it will be given another task.
- Core interface of Executor framework is **Executor**. It has a method called **"execute"**.
- There is another interface called **ExecutorService** which extends **Executor** interface. It can be termed as Executor that provides methods that can control termination and methods that can produce a Future for tracking the progress of one or more asynchronous tasks. It has method such as **submit, shutdown, shutdownNow** etc.
- **ThreadPoolExecutor** is actual implementation of ThreadPool. It extends **AbstractThreadPoolExecutor** which implements ExecutorService interface.
- We can create ThreadPoolExecutor from factory methods of **Executors** class. It is recommended a way to get an instance of ThreadPoolExecutor.
- There are 4 factory methods in Executors class which can be used to get an instance of ThreadPoolExecutor. We are using Executors' newFixedThreadPool to get an instance of ThreadPoolExecutor.



public abstract void execute(Runnable)

public abstract void shutdown()
public abstract List<Runnable> shutdownNow()
public abstract boolean isShutdown()
public abstract boolean isTerminated()
public abstract <T> Future<T> submit(Callable<T>)
public abstract <T> Future<T> submit(Runnable, T)

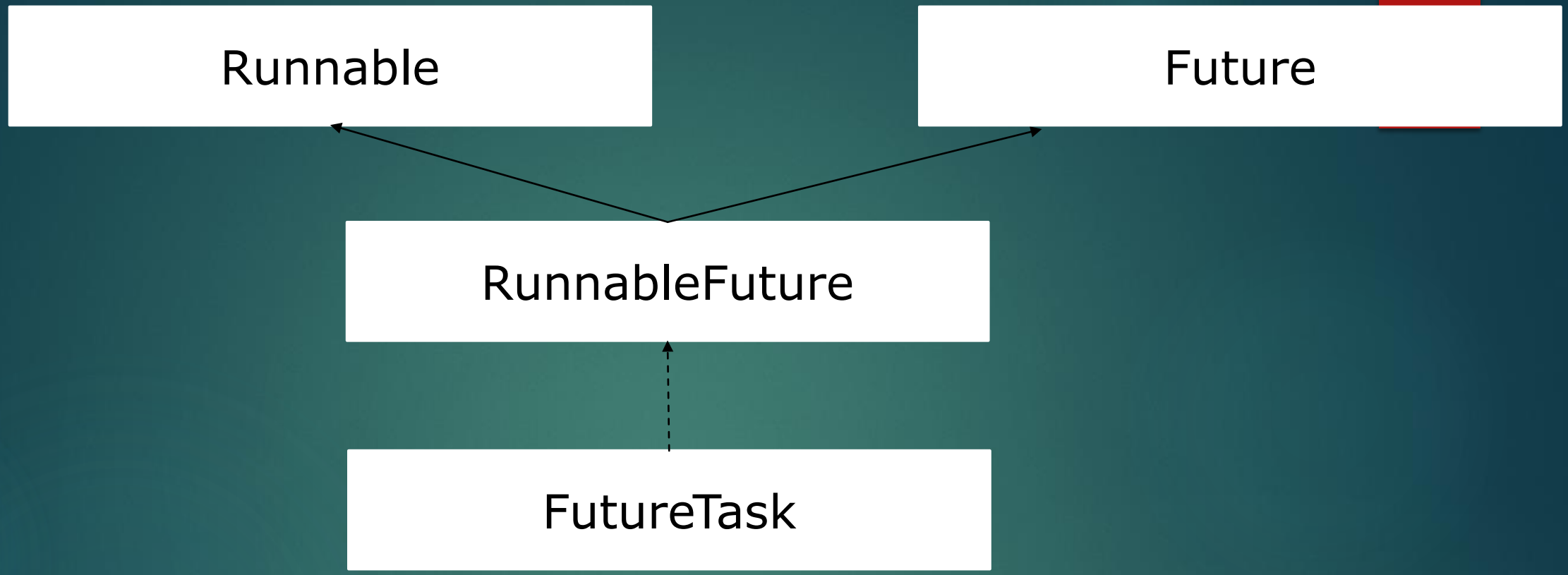
- ***newFixedThreadPool:*** This method returns thread pool executor whose maximum size(let's say n threads) is fixed.If all n threads are busy performing the task and additional tasks are submitted, then they will have to be in the queue until thread is available.
- ***newCachedThreadPool:*** this method returns an unbounded thread pool. It doesn't have maximum size but if it has less number of tasks, then it will tear down unused thread. If thread has been unused for 1 mins (keepAliveTime), then it will tear it down.
- ***newSingleThreadedExecutor:*** this method returns an executor which is guaranteed to use the single thread.
- ***newScheduledThreadPool:*** this method returns a fixed size thread pool that can schedule commands to run after a given delay, or to execute periodically.

Callable and Future:

- Callable interface represents a thread that can return a value. It is very much similar to Runnable interface except that it can return a value. Callable interface can be used to compute status or results that can be returned to invoking thread.
- If it executes successfully, call method will return the result else it must throw an exception.
- ***Future*** is generic interface that represents value which will be returned by callable interface. As callable will return value in some future time.
- There are two methods to get actual value from Future.

  get(): When this method is called, thread will wait for result indefinitely.

  V get(long timeout, TimeUnit unit): When this method is called, thread will wait for result only for specified time.


Runnable
Future
RunnableFuture
FutureTask

Case-Sensitive, Case-Insensitive, Homogenous, Hetrogenous

- In case sensitive matter describe a programming language’s ability to difference between upper case and lower case letters. There are various programming languages : C,C++,C#, Java and many more.
- Case sensitive is the phrase used to describe a programming languages ability to distinguish between upper and lower case versions of a letter in the language's character set. For example, the letter 'a' is considered different than the letter 'A'.
- In case insensitive upper and lowercase letters is same.There are various programming languages for which includes case insensitive like:Ads, ABAP, Fortran, and many more languages.
- In case insensitive lowercase uppercase doesn't matter.
- A container that contains or holds objects of a single type is said to be homogenous. On the other hand, A container that contains objects (derived from a common base class) of a variety of types is termed heterogeneous.
- When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

Generics

- The idea behind generics is to allow any type (Integer, String, etc.. And user defined types) to be passed as a parameter to methods, classes and interfaces.
- Using Generics, we can create classes that work with different data types.
- An entity such as class, interface or method that operates on a parameterized type is called a generic entity.
- Lets see what it was like before Generics!

Before Generics

- Suppose we need to create an arrayList to store Integers/Strings/custom objects

```
public class ArrayList //before Generic classes
{
        private Object[] objArray;
        …..
        public Object get(int i){…..}
        public void add(Object o){….}
}
```

- What is the issue with the above code?
- In the list, any object can be stored, no error checking.
- As Object is the superclass of all other classes

  For Example: in the list of Strings, Integers can be stored.

- The method returns an Object, no type-casting is done.

Advantages of Generics

- Real Life example of Generic Programming: Cup can hold Tea/Milk/Coffee

- Allows writing code that can be reused for objects of many different types.
- Handle multiple types.
- Make the program easier to read and safer.
- Safety type cast and check errors
- Complier ensures the correctness of such types.
- It was introduced in Java 5

Note:

- Generic Methods can be called with arguments of different types.
- Generic methods have a type parameter.

Type Parameters Naming Convention

- Recommended to use Capital letters
- T – type
- E – Element
- K - Key
- N – Number
- V - Value

Bounded Types

- Bounded means "restricted"
- We can restrict types that can be accepted by a method.
- Type Parameter specify a superclass from which all type arguments must be derived.
- This is done using extends keyword as:

  Syntax: <T extends superclassname>

- Thus, superclass defines an inclusive , upper limit.

Imposing multiple restriction on the Bounded types:

- Suppose we want our bounded types to extend a class and implement an interface simultaneously, then bounded type can be specified as:

Syntax: public static <T extends ClassName & Interface> fun(T input)

- Class and interface both can be bound
- One class and one or more interfaces
- Class Type must be specified first
- Use the & operator to connect class and interface
- & creates an intersection type
- T is bounded by ClassName and interface

- Example: Comparable interface

```
public interface java.lang.Comparable<T> {
        public abstract int compareTo(T);
}
```

Return type:

+ve number: if the current object is greater than the object.

-ve number: if the current object is less than the object.

zero: if the current object is equal the object.

Note: Integer, Float and Double classes already implemented Comparable interface

Generic Class

- A class with one or more type variables
- Underlying concept remains same

Explicit Diamond Operator:

- The instance creation portion simply uses <>, which is an empty type argument list.
- It automatically infers the type arguments in the new expression from source type.
- Advantage is just shortening the declaration/initialization expression.

Can we determine the type of instance at runtime?

Yes! , instance.getClass().getName();

Generic Classes with multiple types:

- If we need to create a generic class with two different types
- Just add all types separated by comma

WildCard in Generics

- When the type is unknown or can vary, wildcard is used.
- In Java, ? is a wildcard.
- ? Represents unknown type.
- Type parameter is replaced by ?
- WildCard can be used as type of a parameter, field, return type or local variable

Types of WildCard:

1) Upper bounded wildcards
2) Lower bounded wildcards
3) Unbounded wildcards

UpperBounded WildCards

- This restricts the unknown type to be a specific type or sub-type.

```
public static double listSum(List<Number> L){
        double sum=0.0;
        for(Number x:L)
                sum+=x.doubleValue();
        return sum;
}
```

Issue in above code:

1) List works only for Number Type/
2) It doesn't work for Integer/Double/Float

Reason:

1) Integer is subclass of Number
2) But List<Integer> is not subclass of List<Number>
3) So writing the following code will generate error

```
List<Integer> intList=Arrays.asList<>(4,5,3);
System.out.println("sum= "+listSum(intList));
```

LowerBound

- This restricts the unknown type to be a specific type or super-type of that type.

Unbounded WildCard

- Wildcard without any upper bound or lower bound
- Specified using the wild card character(?)

Collection

- Collection as the name depicts is a group of objects. So, what is the collection framework and what's inside this framework?
- Collection framework is a readymade architecture.
- It is a sophisticated hierarchy of interfaces and classes.
- It is a set of classes and interfaces.
- Each Class has tons of useful functions.
- It represents a unified architecture for storing and manipulating a group of objects.
- All the classes and interfaces are generic.
- Provides high performance through efficient implementation of functions.
- The java.util package contains all the classes and interfaces for the Collection framework.
- Components of collection framework are:
  - Interfaces
  - Classes that implements interfaces
  - Algorithms
- ***Why to use the Collection Framework?***
  - Pre-packaged data structure and algorithms
  - Rich set of API to store, retrieve, manipulate and communicate aggregated data.
  - Used as a container for grouping multiple items into one.

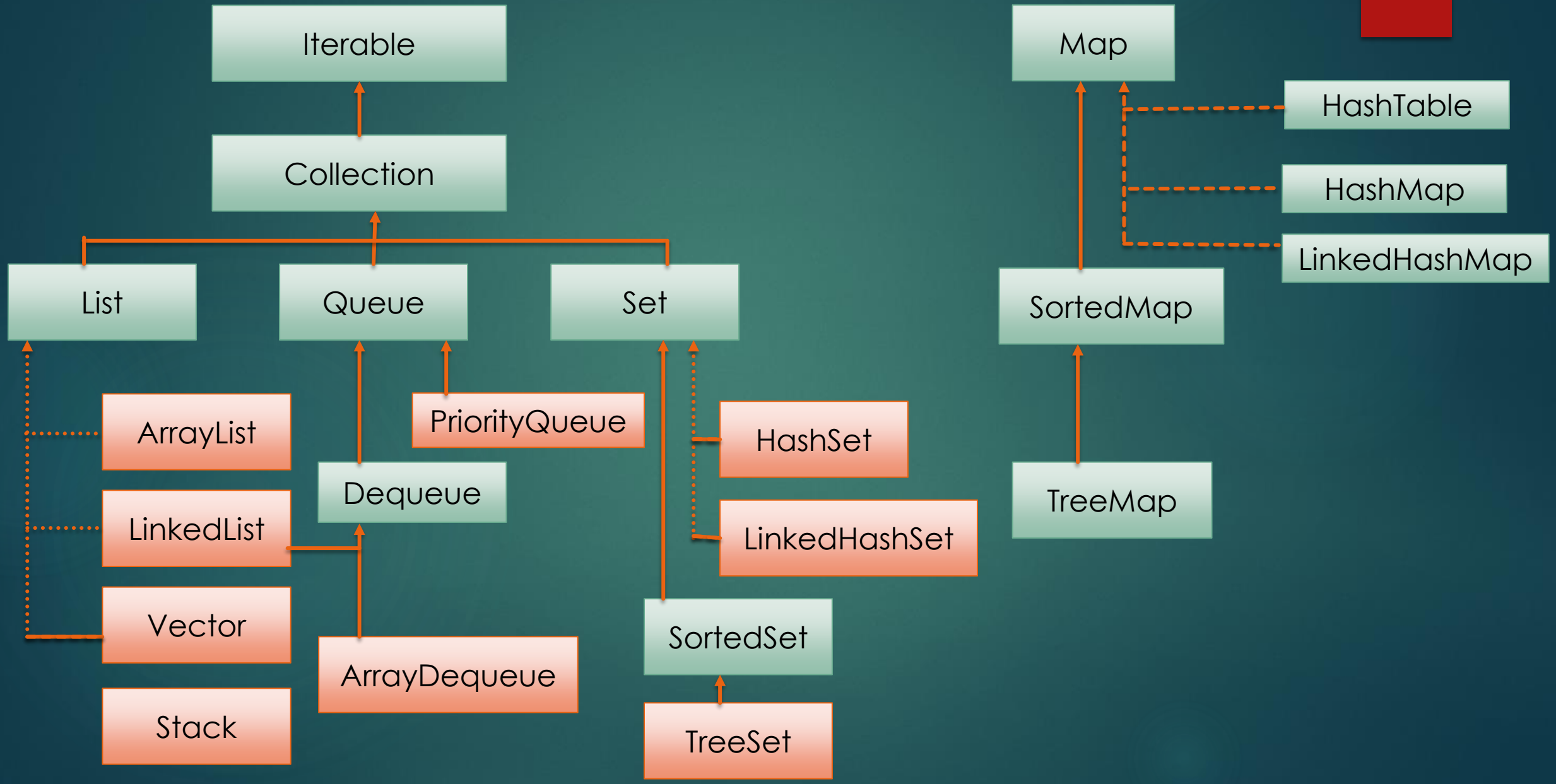
Real Life Examples of Collections

- Representing Collection of Students, Employees, Persons
- A Collections of cards, a mail folders (a collection of letters)
- A telephone directory (a mapping of names to phone numbers)
- Contact List in your mobile phones.
- Files, Directories in Operating System

Benefits of Collection Framework:

- Reduce programming effort
  - Concentrate more on our code
- Increase program speed and quality
  - High Performance and quality implementation of data structure and algorithms
- Reduces effort to learn and to use new APIs
- Fosters software reuse
  - Since we use existing interfaces and classes

Hierarchy of Collection Framework



Iterable Interface

- Iterable interface is the root interface for all the collection classes.
- It enables us to cycle through all the collection elements.
- It provides the facility for iterating the elements in a forward direction only.
- It is just like a for-each loop.
- All collection classes must implement this interface.
- Two commonly used methods of iterator are:
  - Next(): returns the next element in collection
  - hasNext(): check if more elements are there in collection

Collection Interface

- It is the foundation interface of collection framework
- It must be implemented by any class that defines a collection.
- Collection is a generic interface.
- Signature:
  - Interface Collection<E>: E represents any type that collection can hold
- Collection elements can be iterated as iterator is the base interface.
- Iterator is a kind of for-each type loop.

List Interface

- Interface for storing sequence of elements.
- List is a generic interface.
- Signature:
  - Interface List<E>: E represents any type that List can hold.
- Elements can be inserted or accessed by their position in the list.
- Position is zero based.
- List interface is implemented by the cases ArrayList, LinkedList, Vector and Stack.

Set Interface

- The set interface defines a set.
- Underlying class does not allow duplicate elements.
- Set is a generic interface.
  - Interface Set<E>: E represents any type that Set can hold.
- Set is implemented by classes HashSet and LinkedHashSet

SortedSet Interface

- It declares the behaviour of a set sorted in ascending order.
- Parent interface is Set.
- SortedSet is a generic interface.
  - Interface SortedSet<E>: E represents any type that SortedSet can hold.
- SortedSet is implemented by class TreeSet.

Queue Interface

- It declares the behavior of a queue which is often a first-in first-out list.
- Other orders are based on priority or any from our end.
- Queue is a generic interface
  - Interface Queue<E>: E represents any type that Queue can hold.
- Classes PriorityQueue, Deque and ArrayQueue implement the Queue interface.

Dequeue Interface

- It declares the behaviour of a queue which is a double ended queue.
- Insertion or deletion form both ends is allowed.
- Dequeue is a generic interface.
  - Interface Deque<E>: E represents any type that Deque can hold.
- The ArrayDeque class implements the Deque interface.

Collection Classes

- Standard classes that implement core interfaces.
- Few classes provide full implementation.
- Other classes provide partial implementation and become abstract classes.

ArrayList class:

- ArrayList class implements the List interface.
- ArrayList supports dynamic arrays that can grow as needed.
- It removes limitations of standard arrays that cannot grow or shrink dynamically.
- No need to know the array length in advance.
- An ArrayList is a variable-length array of object references.
- Duplicate entries are allowed.
- Insertion order is maintained internally.
- Manipulation (adding/removing an element in between) is time expensive as the underlying data structure is array.
- Supports a number of methods for various types of operations.
- For sorting we have used the sort method of Collections.
- Collections is a Utility class present in java.util package
- It defines several utility methods like sorting and searching which is used to operate on collection.
- It is different form the collection interface.

LinkedList class:

- LinkedList class also implements the list interface.
- It provides a linked list data structure.
- LinkedList class uses a doubly linked list to store the elements.
- Manipulation (inserting/removing elements in between) is fast because no shifting needs to occur.
- LinkedList class can be used as a List, stack or queue.

ArrayList vs LinkedList:

- ArrayList internal storage for elements is dynamic array whereas internal storage for elements in LinkedList is doubly linked list.
- Manipulation with ArrayList is slow as compare to LinkedList.
- An arrayList can only act as a list whereas LinkedList can act as Queue/Deque/Stack.
- ArrayList is better for storing and accessing data but LinkedList is better for manipulating data.

PriorityQueue class:

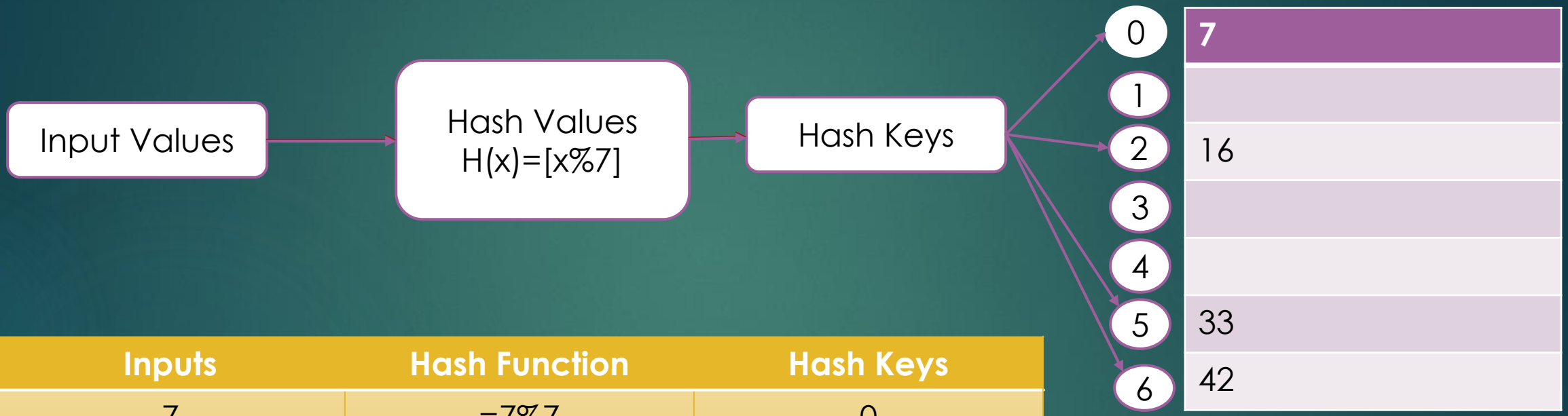
- PriorityQueue class implements the queue interface.
- It doesn’t order the elements in FIFO manner.
- It creates a queue that is prioritized internally.
- The elements of the priority queue are ordered according to the natural ordering(ascending by default) or by a Comparator provided at queue construction time.
- But the iteration order of the PriorityQueue elements is not quaranteed.

Set

- Set is a data structure that keeps only unique elements.

Hashing

- In hashing, the information content of a key us used to determine a unique value, called its hash code.
- Hashing technique is used in searching.



Inputs	Hash Function	Hash Keys
7	=7%7	0
16	=16%7	2
33	=33%7	5
42	=42%7	6

HashSet Class

- HashSet implements the Set interface which implements the Collection interface.
- It creates a collection that uses a hash table for storage.
- A hash table stores information by using a mechanism called hashing.
- HashSet contains unique elements only.
- Elements are inserted on the basis of their hashCode.
- HashSet is the best used for search operations.

LinkedHashSet

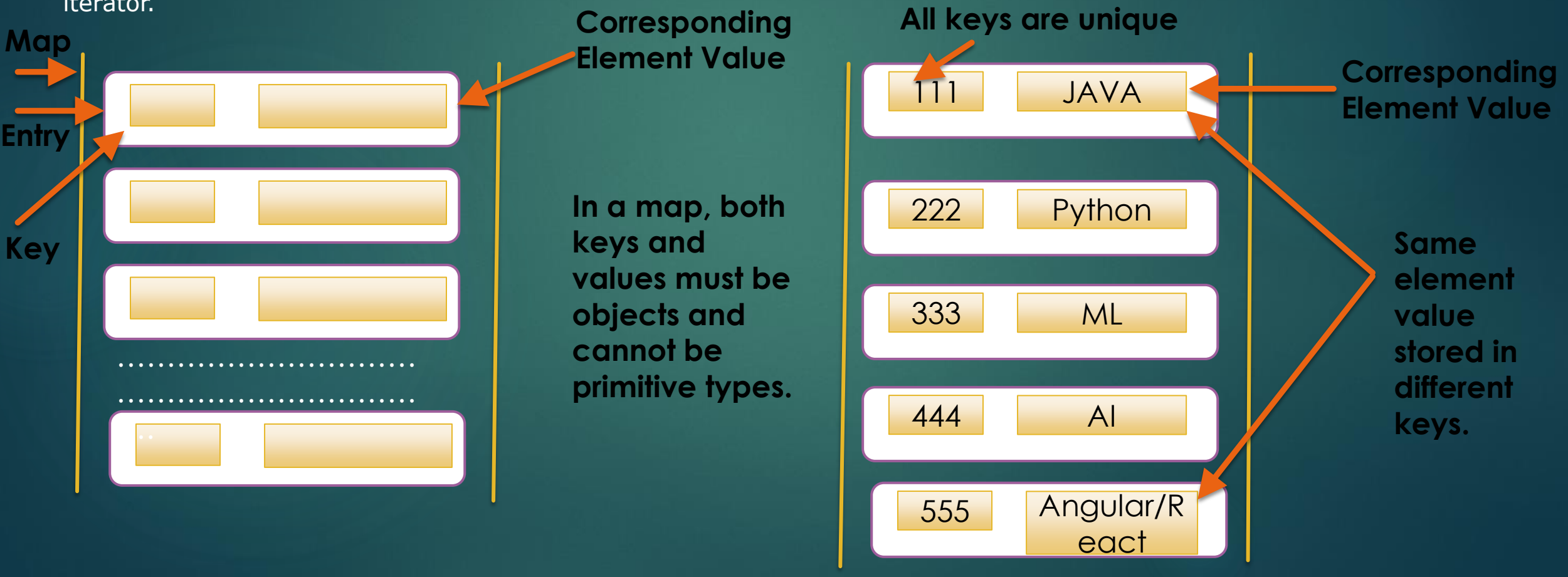
- It maintains a linked list of the entries in insertion order in the Set.
- LinkedHashSet class contains unique elements only like HashSet.
- This allows insertion-order iteration over the set.
- Order is maintained and returned through the iterator.
  - LinkedHashSet=> HashSet+ insertion order

TreeSet

- It created a collection that uses a tree for storage internally.
- Objects are stored in sorted, ascending order.
- Access and retrieval times are quite fast.
- Suitable for finding large amounts of information in a quick manner
- Just like a HashSet, it will also contain unique values.

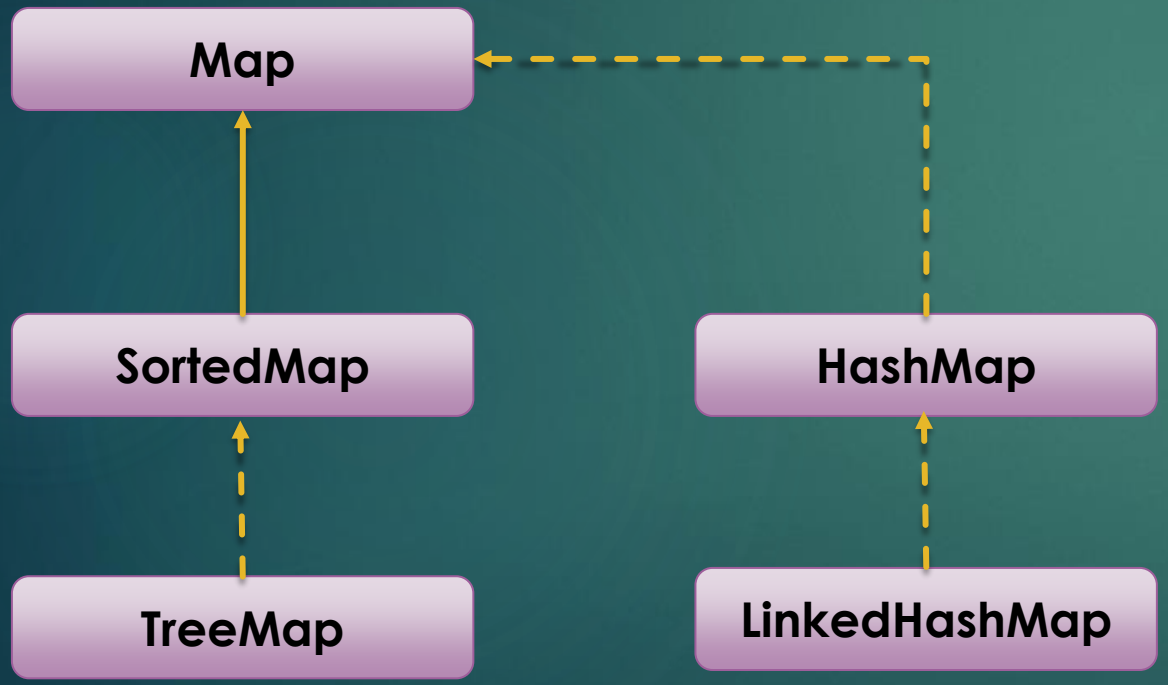
Map

- A map is an object the stores associations between keys and values or key/value pairs.
- Given a key, we can find its values.
- Both keys and values are objects.
- The keys must be unique, but the values may be duplicated.
- Map don't implement the iterable interface (like list and Sets). So we cannot cycle through elements of Map through the iterator.



Map Interface

- The map interface maps unique keys to values.
- Given a key and value, we can store the value in a Map object.
- Stored value can be retrieved using its key.
- Map is generic and is declared as
  - Interface Map<K, V>
  - Here K specifies the type of keys and V specifies the type of values.



SortedMap

- The SortedMap interface extends the Map interface.
- Entries are maintained in ascending order based on the keys.
- Sorted maps allow very efficient manipulations of submaps (in other words, subsets of a map)
- Declaration:
  - interface SortedMap<K,V>

HashMap

- The HashMap class implements the Map interface.
- It internally uses the a hash table to store the Map.
- HashMap is generic class that has this declaration:
  - class HashMap<K,V>: k specifies the type of keys and V specifies the type of values.
- HashMap contains only unique keys.
- HashMap maintains no order

Map.Entry interface

- The Map.Entry interface enables us to work with a map entry.
- Entry is the subinterface of Map.
- Map.Entry is generic and is declared like this:
  - interface Map.Entry<K,V>
- It returns a collection view of the map and map.entrySet represents a set view of the map
- The function getKey gets us key and getValue returns the value for one map entry.

TreeMap

- It creates maps stored in a tree structure.
- It provided an efficient means of storing key/value pairs in sorted order and allows rapid retrival.
- In tree map elements will be stored in ascending key order.
- TreeMap is a generic class that has this declaration:
  - class TreeMap<K,V>
  - K specifies the type of keys and V specifies the type of value.

HashMap vs TreeMap

- HashMap allows exactly one null key but TreeMap doesn’t allow it.
- HashMap maintains no order but TreeMap maintains ascending order.

LinkedHashMap

- LinkedHashMap extends HashMap class
- It maintains a linked list of the entries in the map
- The insertion order is maintained
- LinkedHashMap may have one null key and multiple null values.
- HashMap can also have multiple null values.

Ways to iterate map

- Set is a data structure that keeps only unique elements.

Which approach is better: using forEach or iterator interface

- Iterator approach is better as it is thread-safe.

How does HashMap work in Java?

- HashMap in Java works on hashing principles. It allows ;us to retrieve any object in constant time O(1) provided we know the key
- For every key, hashcode gets generated using the hashcode function
- Every hashCode represents a bucket number (index) in the array where leemnts are stored
- Thys we have an array of buckets.
- While inserting an entry into the mao, it is checked if key is NULL
  - If NULL, it stored in bucket 0
  - If non NULL, it is stored in the bucket decided according to the hashcode
    - (hashcode method is used to decide bucket number, hashCode is calculated on the basis of key)
- For different keys the same hashcode may be generated.
- This situationis know as collision in hashing context.
- To handle a collision, a linked list is created. The new entry is appended to the previous entry in the bucket.
- Each bucket entry can form a linked list.
- All these entry objects in LinkedList will have similar hashCode but equals() method will test for true equality.
- ***If key.equals(K) returns true then both keys are treats as the same key object. This will cause the replacement of the value object else a new entry is added to the linked list.***