

Java

S.D.S.
Page: 9
Date: 11/9

* Difference between JDK, JRE and JVM ? *

1) JVM (Java Virtual Machine)

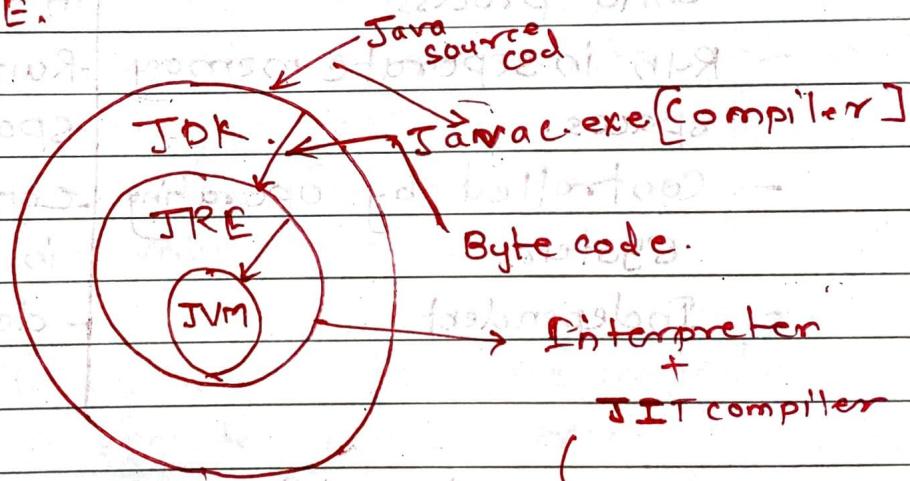
(- It is an abstract machine. It is a specification that provides runtime environment in which Java byte code can be executed.)

2) JRE (Java Runtime Environment)

- It is a runtime environment which implements JVM and provides all class libraries and other files that JVM uses at runtime.

3) JDK (Java Development kit)

- It is the tool necessary to compile, document and package Java programs. The JDK completely includes JRE.



* What is Synchronization ? *

- If multiple threads accessing same object,

Synchronization is a process which keeps all concurrent threads in execution to be in sync.

(One after the other).

- Synchronization avoids memory consistency errors.

* Difference between process and threads.

- Process is running instance of program over the operating system. (running on OS)
- Thread i.e. main thread runs within the process. (running within process)

Process

- An executing instance of program.
- Processes can only control their child process.
- Change in Parent process does not affect the child process.
- Run in separate memory spaces.
- Controlled by operating system.
- Independent

Thread

- A thread is subset of process.
- Threads can control threads of the same process.
- Change in main thread affect the other threads in the same process.
- Run in shared memory spaces.
- Controlled by programmer in a program.
- Dependent.

* Wrapper classes?

- Java primitives converting into the respective reference types and are called wrapper classes.

- Examples:

- | primitive | | → wrapper class. |
|-----------|-------------------|-------------------|
| ① | boolean → Boolean | ⑤ float → Float |
| ② | byte → Byte | ⑥ double → Double |
| ③ | char → Character | ⑦ long → Long |
| ④ | int → Integer | ⑧ short → Short |

#constructing object in Wrapper class. (Boxing)

→ int i = 10; //single value container

Integer iRef = new Integer(i); //Boxing (int → Integer)
 relative points 3
 correct points 3

Extracting the value from object (Unboxing)

→ int j = iRef.intValue(); (Integer → int)

Autoboxing (auto - boxing) points 3 marks no draw

→ Integer kRef = i;

(internally, Integer kRef = new Integer(i);)

Auto-Unboxing (auto - unboxing) points 3 marks no draw

→ int l = kRef;

* final, finally and finalize

final → final is kind of a constant i.e. final

is used to apply restrictions on class, method & variable

→ final class can't be inherited

→ final method can't be overridden

→ final variable value can't be changed.

finally → finally is used to place important code, it will be executed whether exception is handled or not.

e.g. databaseConnection.close();

finalize → It is a method used as destructor for object.

→ finalize is used to perform clean up processing just before object is garbage collected.

→ method finalize called after System.gc();

String → Immutable

stringBuffer and stringBuilder → Mutable.

* Difference between stringBuffer and stringBuilder

String Buffer	String Builder
- Operations are thread-safe	- operations are not thread-safe.
- Multiple threads can work on same String	- multiple threads cannot work on same string i.e. it works on single threaded environment

→ Performance of stringBuilder is faster compared to stringBuffer because of no overhead of synchronized.

* Heaps vs Stack Memory

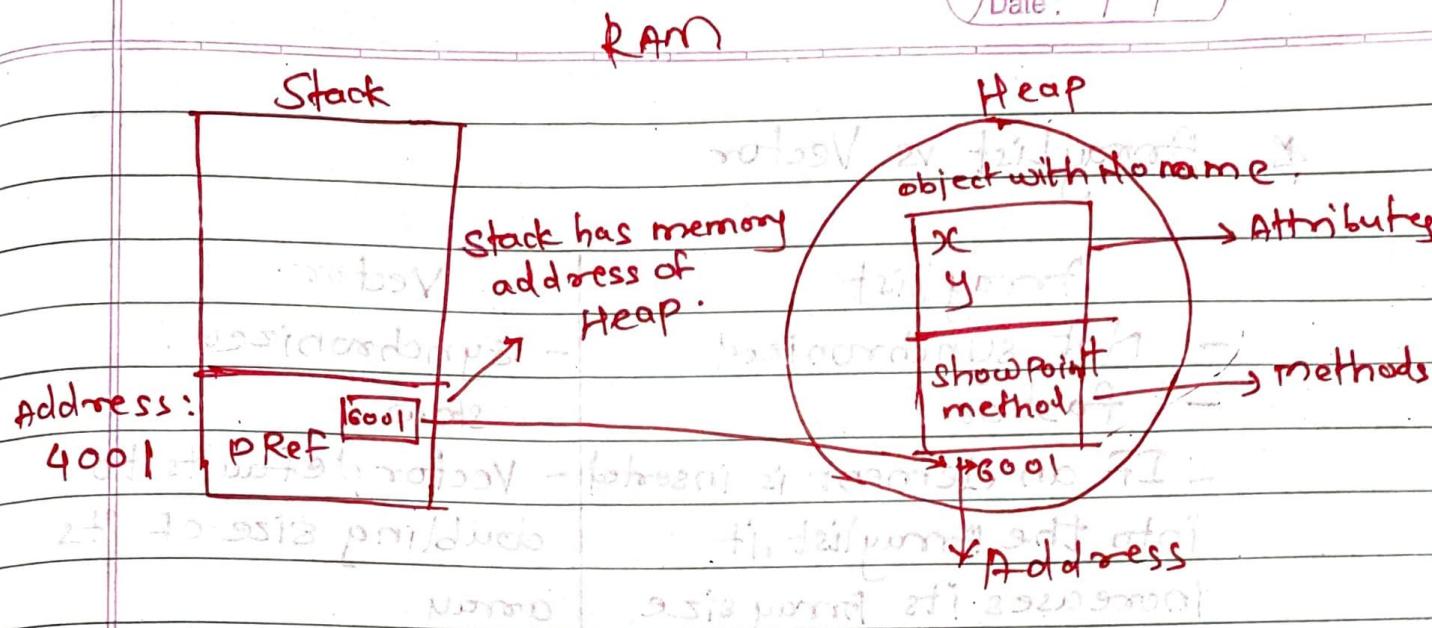
- Example:-

```
class Point {  
    int x; int y;  
    void showPoint() {  
        System.out.println("Point is :" + x + " - " + y);  
    }  
}
```

class Test {

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

```
    Point pRef = new Point();  
    System.out.println("pRef is :" + pRef);  
}
```



- Heap is that memory location where we use `new` operator and `new` operator is used to create the object at runtime.
- So, anything which is constructed at runtime is resides in the Heap memory.

<u>Stack</u>	<u>Heap</u>
- used only by one thread of execution.	- used by all the parts of the application.
- Stack memory can't be accessed by other threads.	- Objects stored in the Heap are globally accessible.
- LIFO manner to free the memory.	- memory management is based on object.
- Lifetime is valid till the execution of thread.	- Lifetime is valid from the start till the end of application execution.
- Stack memory only contains local primitive variables & reference variable to objects in heap space.	- Whenever an object is created, it's always stored in the Heap space.

* ArrayList vs Vector

ArrayList

- Not synchronized.
- faster.
- If an element is inserted into the ArrayList, it increases its Array size by 50%.
- can only use Iterator for traversing an ArrayList.

Vector

- synchronized.
- slow.
- Vector defaults to doubling size of its array.
- can use both Enumeration & Iterator.

* Hashmap vs Hashtable.

HashMap

- ~~synchronized~~ non-synchronized & not-thread-safe.
- Allows one null key & multiple null values.
- faster than Hashtable.
- can make as synchronized by calling Collections.synchronizedMap(HashMap).

Hashtable

- synchronized & thread-safe.
- doesn't allow any null key or value.
- slower than HashMap.
- can't be unsynchronized.

Collections.synchronizedMap

- inherits AbstractMap class.
- traversed by iterator.

Dictionary

- inherits Dictionary class.
- traversed by Enumerator & Iterator.

* Difference equals() & == operator.

→ Example: If you print both, you'll get different output.

```
String str1 = new String("ABC");
```

```
String str2 = new String("ABC");
```

① if (str1 == str2) { → comparing references.

```
s.o.println("str1 == str2");
```

} else { → comparing values.

```
s.o.println("str1 != str2");
```

}

② if (str1.equals(str2)) { → comparing values within the object.

```
s.o.println("str1 equals str2");
```

} else {

```
s.o.println("str1 NOT equals str2");
```

}

→ Output: str1 == str2
str1 != str2

* Abstract classes vs Interfaces.

Abstract class	Interface
----------------	-----------

- provides partial implementation.

- may extend only one other abstract class.

- can have non-abstract methods.

- can have instance variables.

- public, private, protected

- contains constructors

Abstract class	Interface
----------------	-----------

- provides no-implementation but the definition.

- may implement several interfaces.

- All methods are abstract.

- cannot have instance variables.

- must be public.

- cannot contain constructors.

- * **Polymorphism:** → One interface and many implementations.
e.g. assigning different usages to same method.
- * **Runtime polymorphism / dynamic method dispatch:**
→ It is nothing but overriding.
- * **Compile time polymorphism / static dispatch:**
→ Overloading.

Q Difference b/w overloading & overriding.

Overloading	Overriding
- same class, same method name but diff. method signature.	- sub-class having same method name & exactly same method signature.
- "add" or "extends" method behaviour	- "change" existing method behaviour
- compile time polymorphism	- runtime polymorphism.
- inheritance not required	- inheritance required.

- * Can we override a private or static method in java?
→ A private method cannot be overridden since it is not visible from any other class.
- If you create similar method with same return type & same method arguments in child class then it will hide the super class method; (method hiding).
- static method also belongs to class. So, it can't be overridden. To override the static method, method should be static in subclass (method hiding).

- * What is multiple inheritance & does Java support?
 - If a child class inherits the property from classes is known as multiple inheritance.
 - Java does not support multiple inheritance due ambiguity problem.

- * Why Java is not 100% object-oriented?
 - Because we have primitive data types. And to make them object-oriented we need use wrapper classes.

- * Why pointers are not used in Java?
 - Pointers are unsafe & complex.
 - Also, JVM is responsible for memory allocation in Java, thus to avoid direct access to memory by the user, pointers are discouraged in Java.

- * JIT compiler in Java.
 - It is responsible for optimization of Java applications at runtime as interpreters are slow to make code fastly compile into machine code.

- * Why string is immutable?
 - String uses String Constant pool and string pool requires string to be immutable otherwise shared reference can be changed anywhere.
 - Also, string is shared on different area like file system, networking connection, database connection so for security reasons string is immutable. (No one can change reference of string).

* Marker interface.

- Interface having no data members and member functions i.e. empty interface.
e.g. Serializable, cloneable.
- * Why do we need marker interface?
 - Marker interface inform compiler that we have specific to do with the class. It is just a particular kind of information or meta-data to provide to compiler.

* Does "finally" always execute in Java?

- Except in case of following cases:
 - 1) "System.exit()" function.
 - 2) System crash.

* What methods does the object class have?

- ① clone() ⑥ hascode()
- ② equals() ⑦ notify() & notifyAll()
- ③ finalize() ⑧ wait(), notify(), interrupt()
- ④ getClass() ⑨ wait(long timeout)
- ⑤ toString() ⑩ wait(long timeout, int nanos)

* How can you make a class immutable?

- ① Declare the class as final so it can't be extended.
- ② Make all fields private so that direct access is not allowed.
- ③ Don't provide setter methods for variables.
- ④ Make all mutable fields final so that its value can be assigned only once.
- ⑤ Initialize all the fields via a constructor performing a deep copy.

⑥ Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

* What is singleton class in Java?

→ Singleton class is a class whose only one instance can be created at any given time in one JVM.

* How can we make a class Singleton?

→ i) Make constructor private.

ii) Create private static instance of a class.

iii) Create static getInstance() method which will return class instance ~~if and only if~~

[In getInstance() method create a instance only if our private static instance is null, o.w. return the same instance.]

e.g.

```
public class Test { }
```

⑦ → private static Test INSTANCE = null;

⑧ → private Test() { }

```
public static Test getInstance() { }
```

```
    if (INSTANCE == null) { }
```

```
        INSTANCE = new Test();
```

}

```
    return INSTANCE; }
```

}

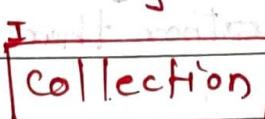
This will help to avoid multiple instantiation of object.

Question: Explain what is final keyword in Java?

Final keyword is used to prevent modification of variable.

↳ `final int a = 10;`

* Collection Hierarchy.



root of Java collection framework &
most of collections inherited from this Interface only. (except Map)

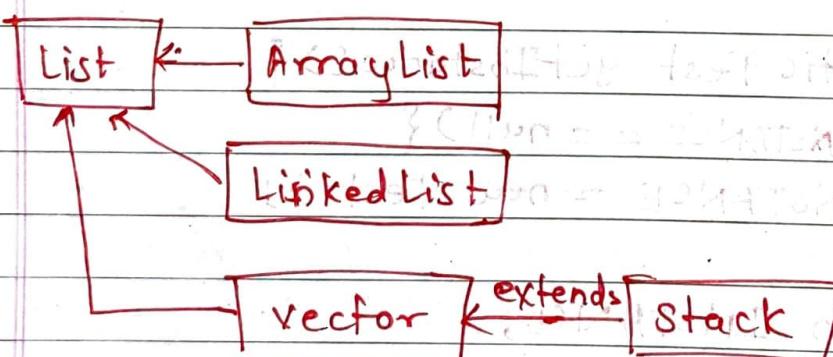
List

- Ordered elements
- follows FIFO approach.
- includes duplicates
- doesn't define any order
- support indexed-based search & random access.
- doesn't support index-based access & elements of List don't support search
- can be easily inserted at the middle.
- duplicates not allowed

Map

- Represents key-value pair.
- does not implement collection interface.
- It contains only unique key.
- can have duplicate values.

* List Interface.



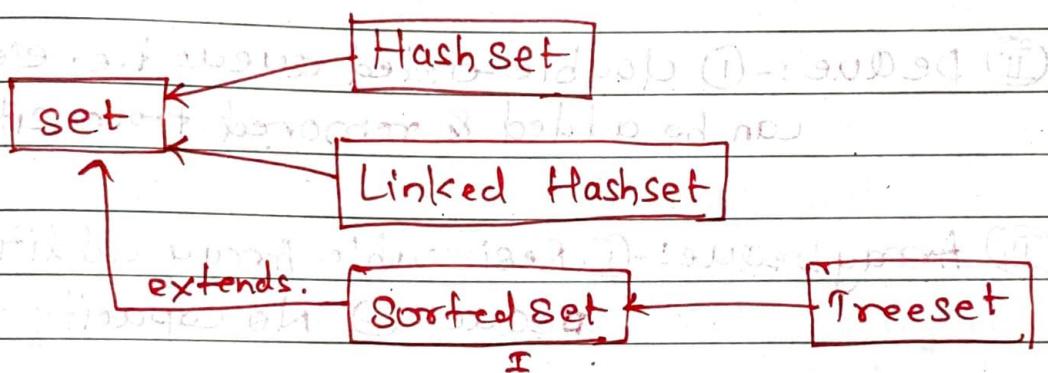
- I**) **ArrayList**:-
- ① Dynamic resizing 50% of original size
 - ② Non synchronized

- II**) **LinkedList**:-
- ① implements List.
 - ② does not support accessing elements randomly.
 - ③ Non-synchronized.

- III Vector :-**
- ① Synchronized & Thread safe.
 - ② maintains the insertion order.
 - ③ increases size by doubling the array size
 - ④ legacy class.

- IV Stack :-**
- ① last-in-first-out
 - ② extends vector

* Set Interface :-



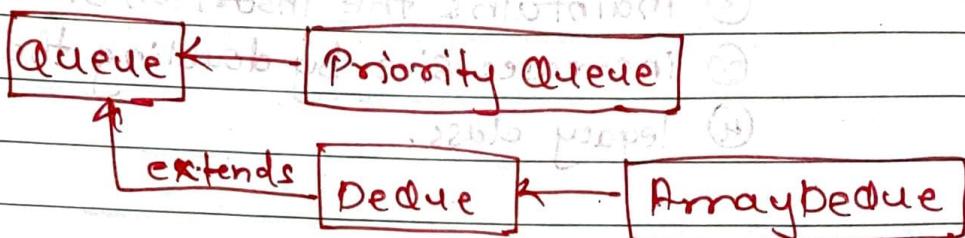
- I HashSet :-**
- ① implicitly implements a hashtable.
 - ② contains only unique elements.
 - ③ one null element can be added.
 - ④ unordered.

- II Linked HashSet :-**
- ① order version of HashSet which maintains a doubly linked list across all elements.
 - ② insertion order is preserved.

- III Sorted Set :-**
- ① All elements of sorted set must be implement the Comparable interface.
 - ② It's a set sorted in ascending order.

- IV TreeSet :-**
- ① uses tree for storage
 - ② objects are stored in sorted & ascending order.

* Queue Interface

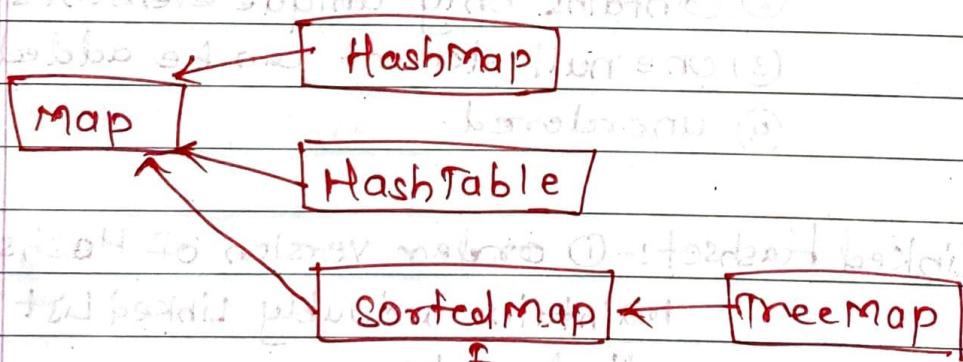


① Priority Queue:- ① high priority element is served before low priority element irrespective of insertion order.

② Deque :- ① double-ended queue i.e. elements can be added & removed from either end.

③ Array Deque:- ① Resizable Array addition to the deque. ② No capacity restriction.

* Map Interface



① HashMap:- ① Non-synchronized.

- ② one null key & multiple null values.
- ③ 16 buckets created by default

② HashTable :- ① synchronized

- ② doesn't allow any null key or null value.

- ③ 11 buckets created by default

III Sorted map :- ① entries are maintained in an ascending key order

IV TreeMap :- ① Implicitly implements the Red-Black Tree implementation. ② Cannot store any null key.

- * Why Map doesn't extends the collection Interface?
 - ① Map follows key-value pair structure.
 - ② add method of collection doesn't support key-value pair like Map.

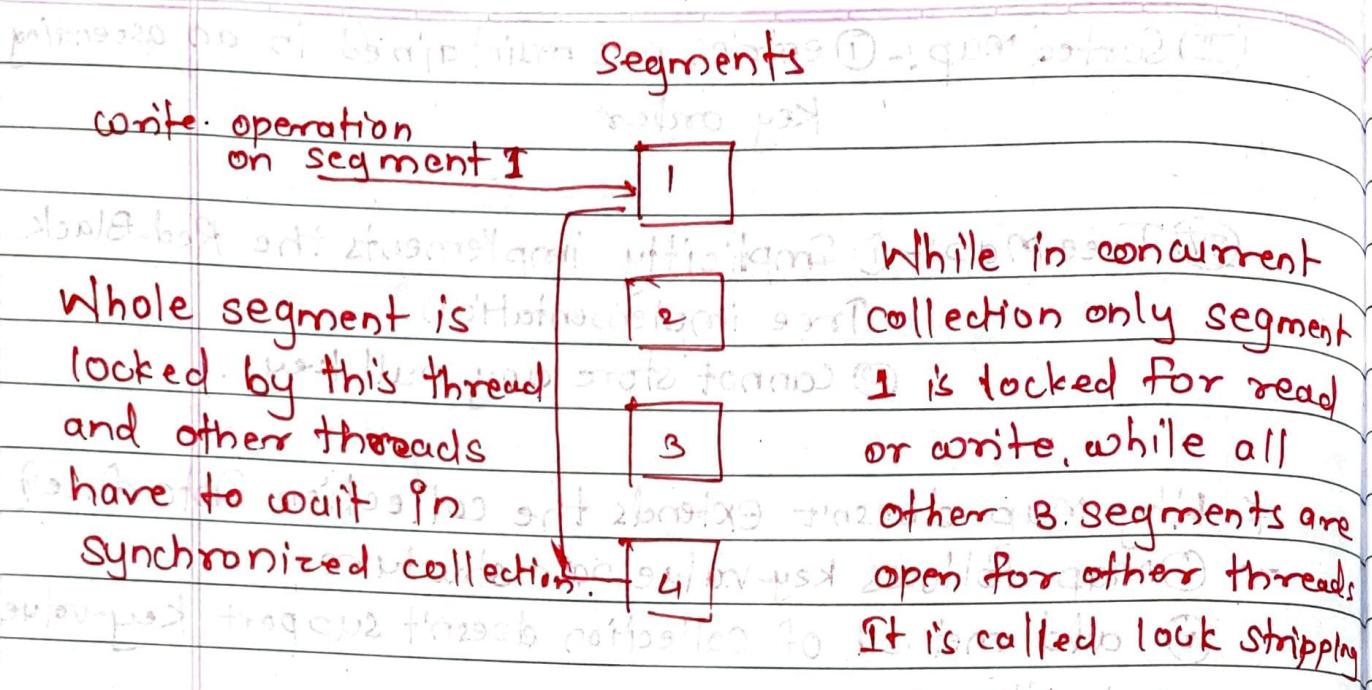
* Difference between fail-fast and fail-safe Iterators.

→ Fail-fast :- When one Thread is iterating over collection object & other thread structurally modify collection either by adding, removing or modifying objects on underlying collection, is called fail-fast because they try to immediately throw Exception (when they encounter failure). (ConcurrentModificationException).

Fail-Safe :- fail-safe iterator doesn't throw any exception if collection is modified structurally while one thread is modifying it. because they work on clone of collection instead of original collection.

* Difference between synchronized & Concurrent collection.

- ① Both provide thread-safety.
 - ② difference between them comes in performance, scalability & how they achieve thread-safety.
- e.g. \$ Synchronized Hashmap is slower than concurrent - Hashmap.



* Is Java pass by value or pass by reference?

→ Pass by Reference is sending memory address of a variable or pointer to the memory location into a function.

Pass by Value is sending copy value of variable to function into another memory location.

→ Java is pass by value (if we are sending object of a class in function then we send reference as value in function arg.)

* Why Comparable & Comparator interface required in Java?

→ To sort custom objects. Existing sorting methods can be used only for sorting primitive data type.

* Comparable v/s comparator.

~~Java also has another interface Comparable & Comparator.~~

~~↳ Provides single sorting - provides multiple sorting sequence i.e. sorts the sequence and sorts the collection on the basis of collection on the basis of single element or multiple elements.~~

- affects the original (- doesn't affect the class) original class
- uses compareTo() method. - uses compare() method.
- Java.lang package - java.util package.
- Sort elements by - sorts elements by

* Equals() and HashCode() in Java.

→ (i) Equals():- This method is used to compare equality of two objects. The equality can be compared in two ways:-

(i) Shallow comparison:- The default implementation of equals method is defined in java.lang.Object class which simply checks if two object references refer to the same ~~class~~ object i.e. `obj1 == obj2`.

(ii) Deep Comparison:- When a class provides its own ~~comparison~~ equals() method in order to compare the objects of the class w.r.t. state of objects is called Deep comparison. Fields are compared with one another i.e. `obj1.getId() == obj2.getId()`

Syntax:- `public boolean equals(Object obj) {`
`if (obj == null || getClass() != obj.getClass())`
`return false;`
`if (o == this) { return true; }`
~~if (class c = (Class)o;~~
`return (this.getId() == c.getId());`

(ii) hashCode(): - It returns the hashcode value as an integer. Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable, etc.

This must be overridden in every class which overrides equals() method.

Syntax:-

```
public int hashCode() { }
```

Contract:-

- ① If two objects are equal according to Equals() method then the hash code for both the object must be same integer value.
- ② It's not necessary that if you have same hashCode for 2 object means those two objects are equal.
- ③ If it's invoked on the same object more than once during an execution of Java application, the hashCode method must consistently return the same integer.

* Can we restrict visibility of derived methods?

- We cannot reduce the scope of modifiers of derived or overridden methods, but we can increase or make it more accessible than parent.
- public → private or default → X
default → public → ✓

Object-Oriented Programming

* Shadowing Java.

→ Both parent and child class methods are static.
In this case, even if we create child class reference & using parent object the also parent class method will get called.

* Association

- Relationship between two classes.

Type:-

① Aggregation (weak Association):- Loosely coupling between two objects. Both objects can exist independently.

② Composition (strong Association):- Tight coupling between two objects. One object cannot exist without another object.

* Covariant return Type

→ In overriding, child class method can have child of return type of parent method as return type.

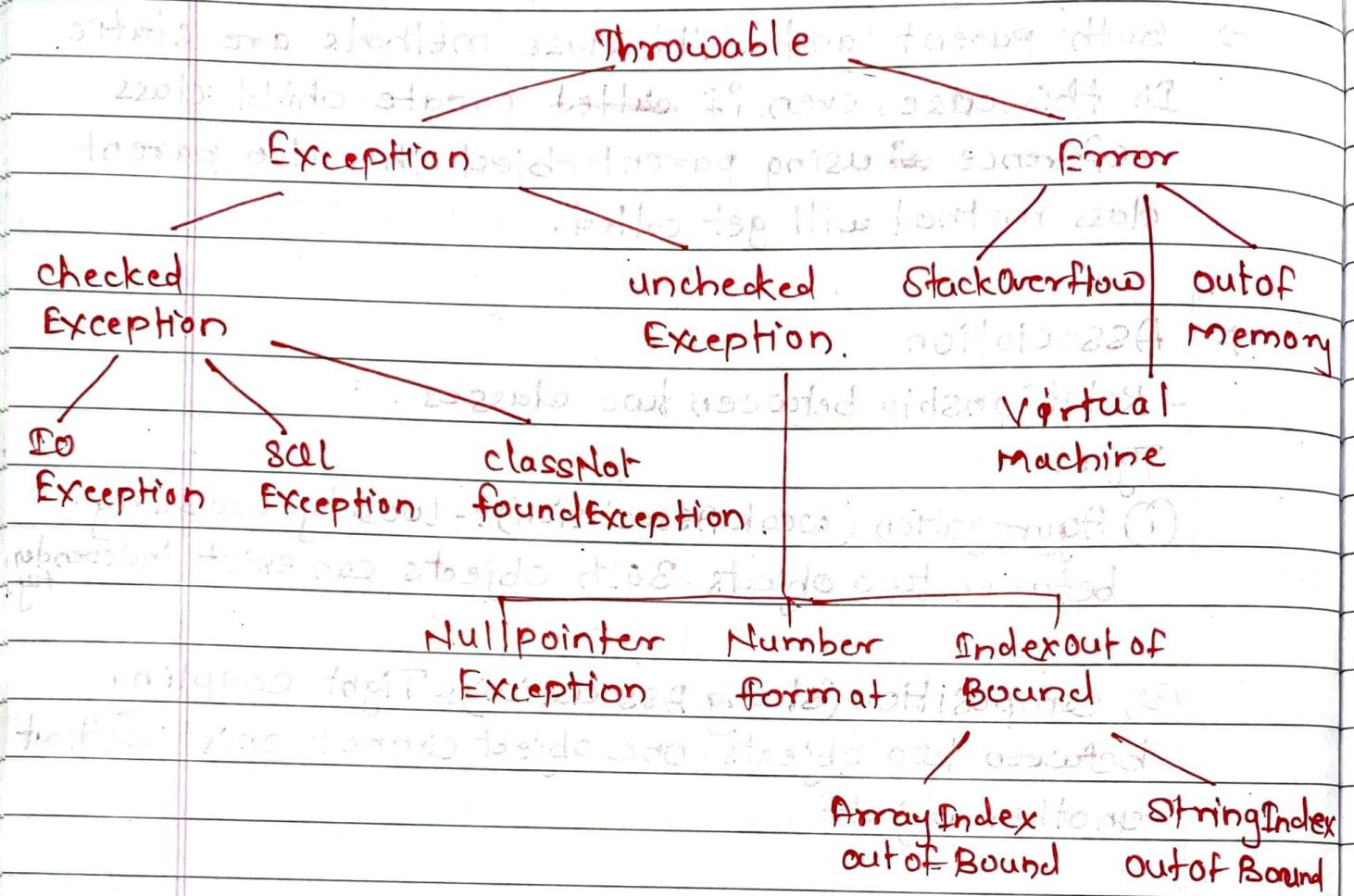
e.g. `Parent` → `child` parent class

return type → Object → String ✓

* Exception :- Exception is abnormal condition which occurs during the execution of a program & disrupts program flow.

In Exceptions are handled using try, catch, finally.

* Hierarchy of Exception



* Can we write only try block w/o catch & finally block?

- ① No. either catch or finally must be written.
- ② If we don't write then compile time error will display saying "insert finally to complete try statement".

* Throw vs Throws.

- ① Throw used to explicitly throw an exception
Throws is used to declare an exception.
- ② Throw used within the method.
Throws used with method signature.
- ③ checked Exception cannot be propagated using throw only. checked Exception be propagated with throws.
- ④ multiple exception can't throw using throw keyword.

* Unreachable ~~catch~~ catch block error.

→ This error comes when we keep super classes first and sub-class later in inheritance. (Priority)

e.g. `class A {
 void m() {
 System.out.println("A");
 }
}
class B extends A {
 void m() {
 System.out.println("B");
 }
 public static void main(String[] args) {
 try {
 B b = new B();
 b.m();
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}`

Hence order of catch block

`} catch (NullPointerException ne) {}` matters.

* Promotion of types in constructor

→ `Byte → Short → Int → Long → Float → Double.`

* Java 8 Features

① Lambda Expression.

② Stream API

③ Default methods in the Interface.

④ Static methods in the Interface.

⑤ Functional Interface. (Not in our book)

⑥ Optional Interface. (Not in our book)

⑦ Method references. (Not in our book)

⑧ Date API

⑨ Nashorn. Javascript Engine.

* Lambda Expression. ($\lambda \rightarrow \text{return part}$)

→ Lambda Expression is anonymous function i.e. no name, return type, and access modifier.

e.g. Normal code ~~Normal code~~ Lambda expression

`public void add(int a, int b){
 System.out.println(a+b);
}` $(a,b) \rightarrow \text{System.out.println}(a+b);$

* Functional interfaces (@FunctionalInterface)

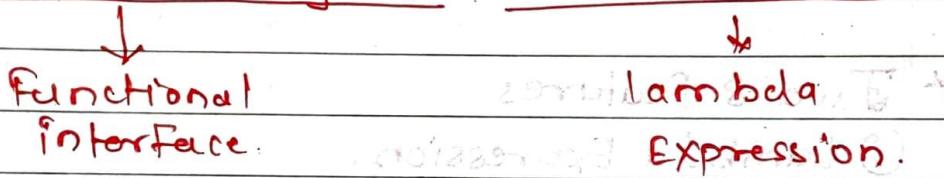
- ① Functional interfaces are interfaces which can have only one abstract methods.
- ② It can have any no. of static, default methods.
- ③ e.g. Comparable, Runnable

* How Lambda expression & Functional Interfaces are related?

- Functional interface is used to provide reference to lambda expression.

e.g. `Comparator<String> c = (s1, s2) → s1.compareTo(s2);`

Comparator<String> c = (s1, s2) → s1.compareTo(s2);



- * Custom Functional interface.
- ① Create an interface.
- ② Annotate that with @FunctionalInterface.
- ③ Define only one Abstract method.

* Method Reference in Java 8. (::)

- ① Method reference is replacement of lambda expression.
- ② It is used to refer method of functional interface to an existing method.

`Object o = List.of(1, 2, 3).stream().filter(i → i > 1).
collect(Collectors.toList());`

* Default methods.

→ ① Default method is a way for adding new methods to interface without affecting implementing classes. offence with default method Java defined many compile time errors that may arise due to unimplemented methods of interface.

③ default method provide implementation. if we are ok with that implementation then we can use same. If not satisfied then we can override & provide own implementation.

* Is default keyword one of the access modifier?

→ No. default is not access modifier.

* How to override default method

→ ① keep the method signature same (name + arguments)
 ② Remove default keyword and public modifier to overridden method.

* Can we use hashCode() default implementation in Interface?

→ No, because we are not allowed to override object class's method as default methods in interface else will get compile time error.

* How Default methods in interface cope up with Diamond problem.

→ ① If one class implements two or more interfaces having same ^{default} method, then compiler will get confused that which method should consider. This is the Diamond problem.

② In such situation IDE, will ask to override any of interface's default method.

e.g.

Interface 1 & 2

default void m1();

class implements Interface1 & 2

@Override
public void m1();

↳ Got sub implement of 2 Interface super.m1();

↳ Go to 2nd abstract method

↳ Go to 3rd abstract method

* Static methods in ~~Java 8~~ interface in Java 8

→ ① Static methods in interface is that you can call those methods with just interface name.

No need to create class & then its object.

* Reason to introduced static methods in interface in Java 8

- ↳ Interface can never contain:

① Constructors ② Static blocks ③ Nothing costly in terms of memory & performance.

- Also, we don't need to create object.

* Static Methods in interface can't be overridden.

* What are Predicates?

→ ① Predicate is a predefined functional Interface

② Only abstract method of predicate is test(T t);
public boolean test(T t);

③ Used to check boolean condition.

④ Return type of predicate is boolean.

* Predicate Joining

→ ① Combine predicates in serial predicate.

② Using AND, OR, Negate.

* Functions in Java 8.

→ ① Function is predefined functional interface.

② Only abstract method of function is apply($T t$).

$R \text{ apply}(T t);$

③ Takes one input and returns one output.

Syntax :- $\text{Function}\langle T_1, T_2 \rangle f = \text{lambda} \dots;$

e.g.

$\text{Function}\langle \text{Integer}, \text{Integer} \rangle \text{sq} = \text{lambda} i \rightarrow i * i;$

$\text{sq. apply}(5); \rightarrow 25.$

* Predicates vs Function

Predicate

minimally functional

- return type \rightarrow Boolean \rightarrow Return Type \rightarrow Object

- accepts single argument - accepts single argument

i.e. $\text{Predicate}\langle T \rangle$ & return boolean only but $\text{Function}\langle T, R \rangle$.

- test() method. - apply() method.

* Functional Chaining

- We can combine/chain multiple functions together with andThen and compose methods.

e.g. $f1 \circ f2 \circ f3 \circ f4 \circ f5 \circ f6 \circ f7 \circ f8 \circ f9 \circ f10$

① andThen

$f1 \cdot \text{andThen}(f2) \cdot \text{apply}(\text{Input}); \rightarrow \text{first } f1 \& \text{ then } f2$

② compose

$f1 \cdot \text{compose}(f2) \cdot \text{apply}(\text{Input}); \rightarrow \text{first } f2 \text{ then } f1.$

- Multiple functions can be chained together like:

$f1 \cdot \text{andThen}(f2) \cdot \text{andThen}(f3) \cdot \text{compose}(f4) \cdot \text{apply}(\text{Input});$