Java

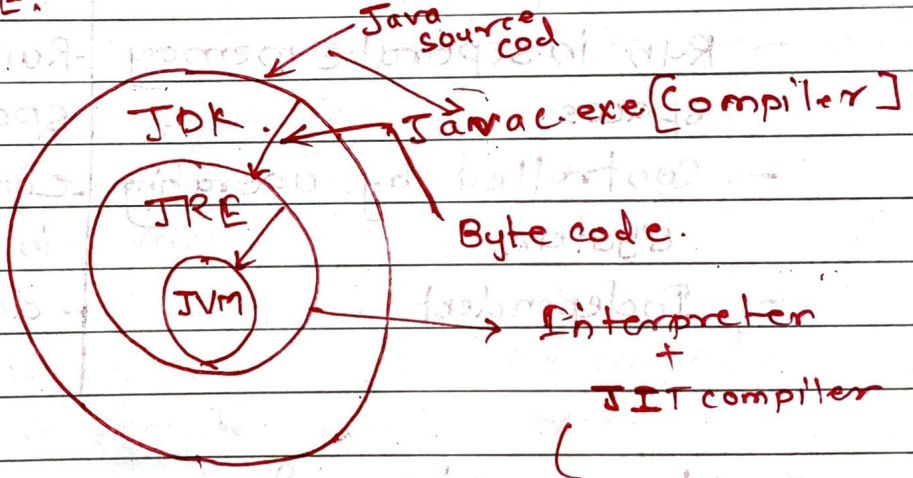
S.D.S.
Page :
Date : / /

* 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	Thread
- An executing instance of program.	- A thread is subset of process.
- Processes can only control their child process	- Threads can control threads of the same process.
- Change in parent process does not affect the child process	- Change in main thread affect the other threads in the same process.
- Run in seperate memory spaces.	- Run in shared memory spaces.
- Controlled by operating system.	- controlled by programmer in a program.
- Independent	- dependent.

* Wrapper classes?
- Java primitives converting into the respective reference types and are called Wrapper classes.
- Examples.

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

#constructing object in Wrapper class. (Boxing)

→ int i = 10; // single value container

Integer iRef = new Integer(i); // Boxing (int → Integer)

Extracting the value from object (Unboxing)

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

Autoboxing

→ Integer kRef = i;

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

Auto-Unboxing

→ 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.

StringBuffer	StringBuilder
- Operations are thread-safe	- operations are not thread-safe.
- Multiple threads can work on same String	- ~~cannot~~ Multiple threads cannot work on same String i.e. it works on single threaded enviroment

→ Performance of StringBuilder is faster compared to StringBuffer because of no overhead of synchronized.

* Heap vs Stack Memory.

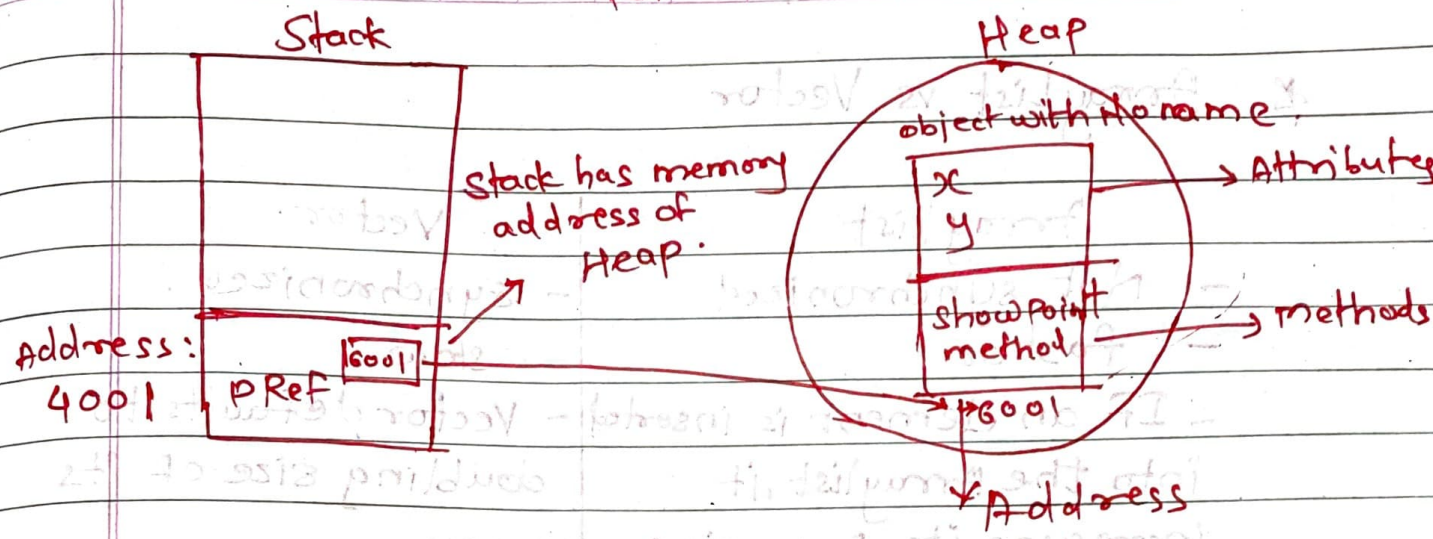
- Example:-

```
class Point {
    int x;
    int y;

    void showPoint (){
        s.o.pln ("point is :"+x+" - "+y);
    }

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

        Point pRef = new Point();
        s.o.pln ("pRef is :"+ pRef);
    }
```

x, y & method showPoint will reside in Heap memory

pRef object will reside in the stack memory & have address of Heap.

RAM



- 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.

Stack	Heap
- used only by one thread of execution.	- used by all the parts of the application.
- Stack memory can't be accessed by other threads	- ~~Heap~~ 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 alway stored in the Heap space.

* ArrayList vs Vector

ArrayList	Vector.
- Not synchronized	- synchronized.
- faster	- slow
- If an element is inserted into the Arraylist, it increases its Array size by 50%.	- Vector defaults to doubling size of its array.
- can only use Iterator for transversing an Arraylist.	- can use both Enumeration & Iterator.

* Hashmap vs Hashtable.

Hashmap	Hashtable.
- ~~synch~~ non-synchronized & not-thread safe.	- synchronized & thread-safe.
- Allows one null key & multiple null value.	- doesn't allow any null key or value.
- faster than Hashtable	- slower than Hashmap.
- can make as synchronized by calling Collections.synchronizedMap (HashMap).	- can't be unsynchronized.
- inherits AbstractMap class	- inherits Dictionary class.
- transversed by iterator	- transersed by Enumerator & Iterator

* Difference equals() & == operator.

→ Example:

```
String str1 = new String("ABC");
String str2 = new String("ABC");
```

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

```
    S.O.pln("str1 = str2");
} else {
    S.O.p.ln("str1 != str2");
}
```

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

```
    S.O.pln("str1 equals str2");
} else {
    S.O.p.ln("str1 NOT equals str2");
}
```

* Abstract class vs Interfaces.

Abstract class	Interface.
- provides partial implementation.	- provides no-implementation but the definition.
- may extend only one abstract class.	- may implement several interfaces.
- can have non-abstract methods.	- All methods are abstract.
- can have instance variables	- cannot have instance variable.
- public, private, protected	- must be public.
- contains constructors	- cannot contain constructors

* Polymorphism:
→ one interface and many implementations.
i.e. assign different usage to same method.

* Runtime polymorphism / dynamic method dispatch.
→ It is nothing but overriding.

* Compile time polymorphism / static
→ overloading.

* Difference b/w overloading & overridding.

Overloading	overridding.
- same class, ~~but~~ 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	- inheritence 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 inheritence & 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 fuctions 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 information or meta-data to provided to compiler.

* Does "finally" always execute in Java?

→ Except in case of
1) "system.exit()" fuction.
2) system crash.

* What methods does the object class have?

→ ① clone()
② equals()
③ finalize()
④ getclass()
⑤ to string()
⑥ hascode()
⑦ notify() & notifyAll()
⑧ wait
⑨ wait (long timeout)
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 it's value can be assinged 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.

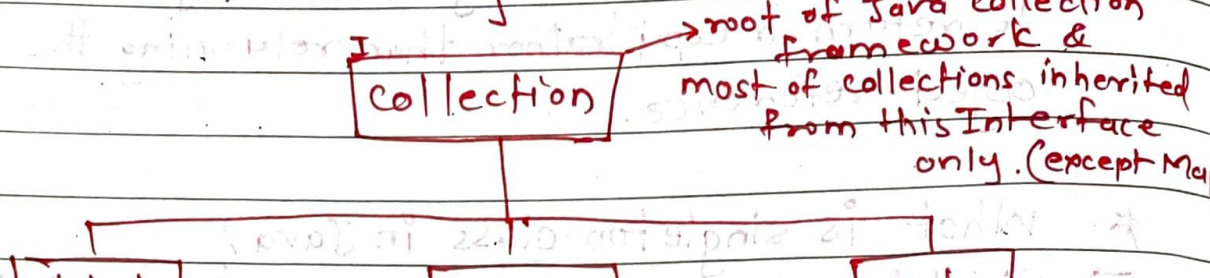
* What is singleton class in Java?
→ Singleton class is a class whose only one instance can be created at any given time in one JVM.

* 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 {
    private static Test INSTANCE = null;
    private Test() { }

    public static Test getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new Test();
        }
        return INSTANCE;
    }
```

* Collection Hierarchy.



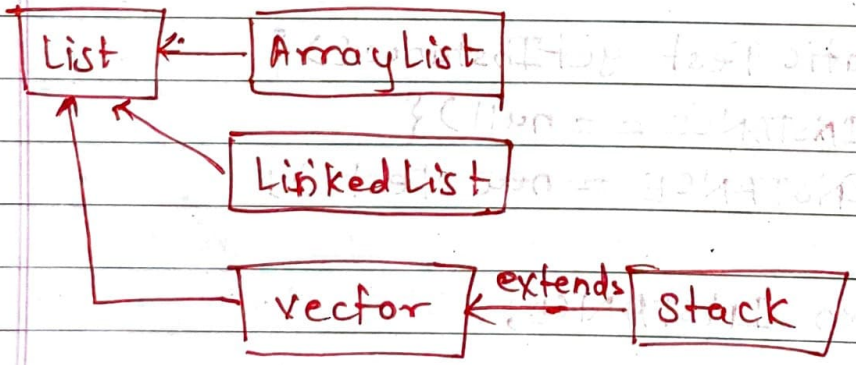
List
- ordered elements
- includes duplicates
- support indexed-based search, random access & elements can be easily inserted

Queue
- follow FIFO approach.

Set
- doesn't define any order
- doesn't support index based search
- duplicates not allowed

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

* List Interface.



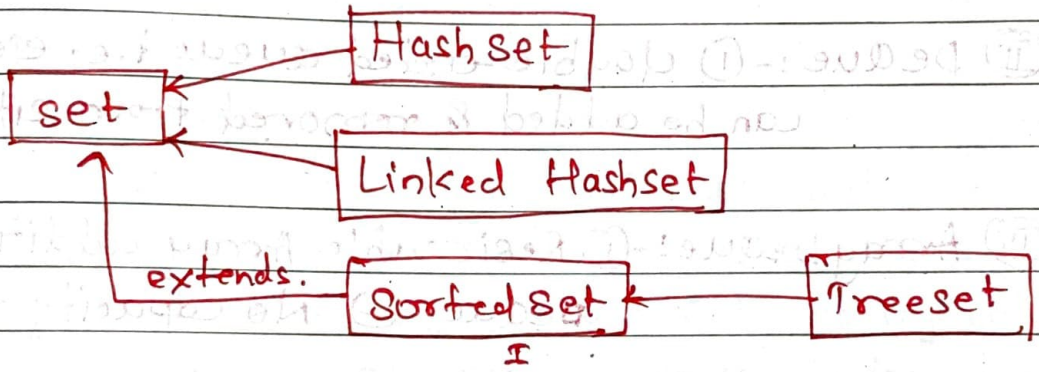
(I) ArrayList :- ① Dynamic resizing 50% of original size
② Non synchronized.

(II) Linked List :- 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 ~~is~~ extends vector.

* Set Interface.



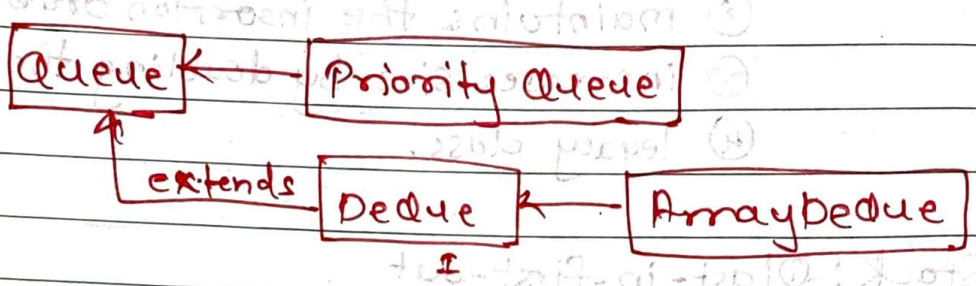
(I) Hashset :- ① implicity implements a hashtable.
② contains only unique elements.
③ one null element can be added.
④ unordered.

(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) Tree set :- ① Uses Tree for storage
② objects are stored in sorted & ascending order.

* Queue Interface.

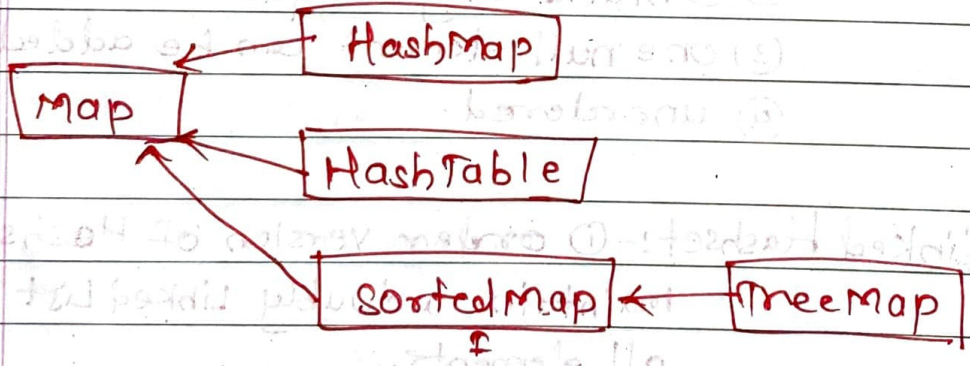


(I) Priority Queue:- ① high priority element is served before low priority element. Irrespective of insertion order.

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

(III) ArrayDeque:- ① Resizeable Array addition to the Deque. ② No capacity restriction.

* Map Interface.



(I) HashMap:- ① Non-synchronized.
② one null key & multiple null values.
③ 16 buckets created by default

(II) 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) Tree Map :- 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).

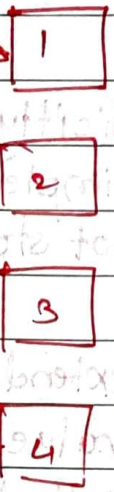
Fail-safe :- fail-safe iterator doesn't throw any exception if collection is modified structurally while one thread is ~~modifying~~ Iterating over it because they work on clone of collection instead of original collection.

* 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.

Page:
Date: / /

Segments

write operation on segment 1



Whole segment is locked by this thread and other threads have to wait in synchronized collection.

While in concurrent collection only segment 1 is locked for read or write, while all other 3 segments are open for other threads. It is called lock stripping

* 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..

S.D.S.
Page :
Date : / /

* Comparable v/s comparator:-

Comparable	comparator.
- provides single sorting sequence i.e. sorts the collection on the basis of single element.	- provides multiple sorting sequence. i.e. sorts the collection on the basis of multiple elements.
- affects the original class	- doesn't affect the original class.
- uses compareTo() method.	- uses compare() method.
- Java.lang package	- java.util. package.
- Sort elements by collections.sort (List)	- sorts elements by collections.sort (list, comparator)

* 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() != o.getClass())
        return false;
    if (o==this) { return true; }
    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) Its not neccessary that if you have same hashcode for 2 object means those two objects are equal.
(3) If it is envoked 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 → ✓

* Shadowing Java.

→ Both parent and child class methods are static. In this case, even if ~~called~~ create child class reference ~~as~~ 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 overridding, child class method can have child of return type of parent method as return type.

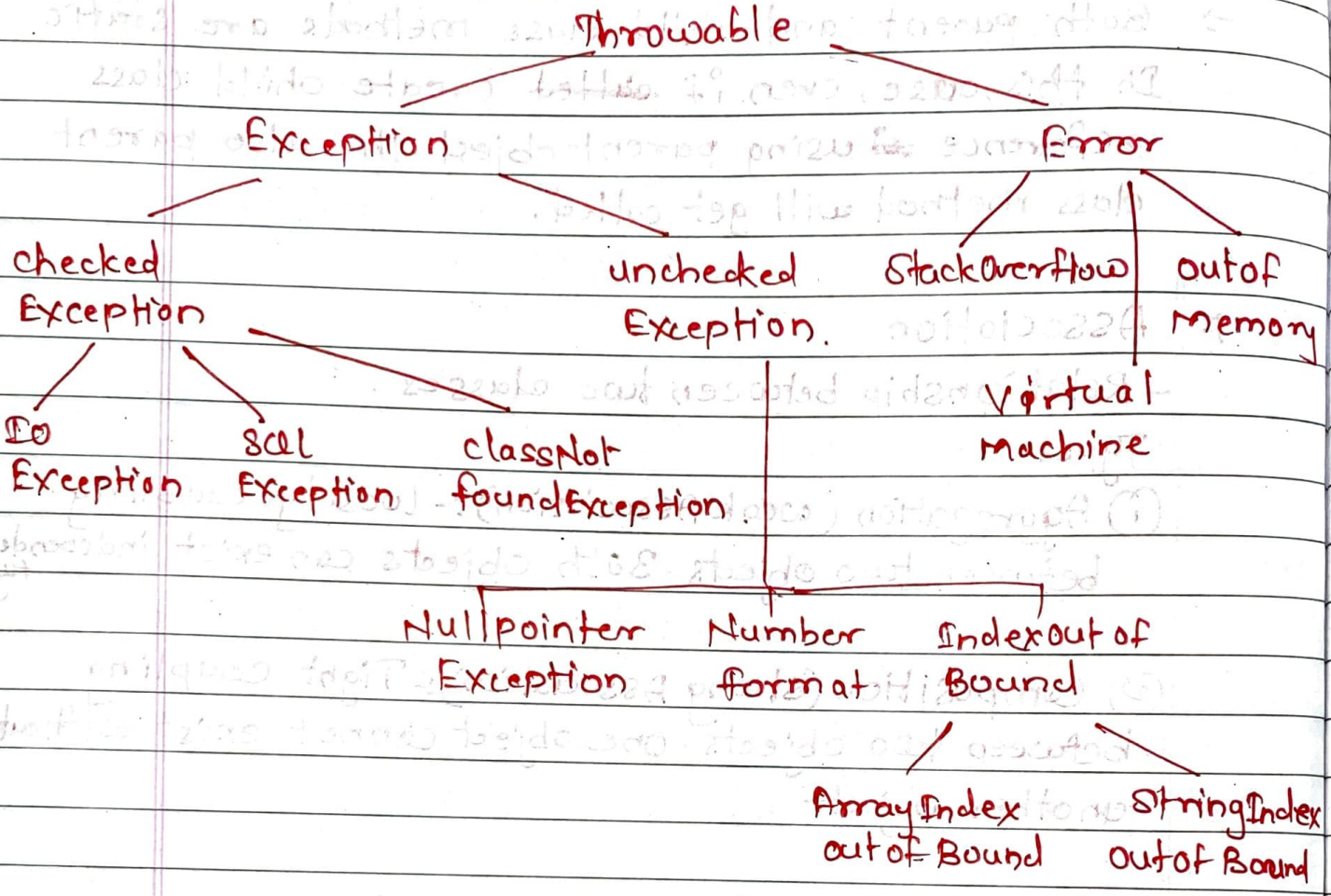
e.g.

	Parent		child
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 explicity 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 propogated using throw only. checked Exception be propogated 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.

e.g.

```
{ catch ( Exeption e ) {

} catch ( NullpointerExleption ne ) {
}
```

} Hence order of catch block 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
⑤ Functional Interface.
⑥ optional
⑦ method references
⑧ Date API
⑨ Nashorn, Javascript Engine.

* Lambda Expression. (→)

→ lambda Expression is anonyms fuction i.e. no name, return type, and access modifier

e.g. Normal code

```
public void add(int a, int b){
  s.o.pln (a+b);
}
```

lambda expression

```
(a,b) → s.o.pln (a+b);
```

* Fuctional 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 → Functional interface.

(s1, s2) -> s1.compareTo(s2); → lambda Expression.

* 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.

class :: method.

* Default methods.

→ ① Default method is a way for adding new methods to interface without affecting implementing classes.

② Hence with default method Java defended 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. (over-ridde)

* 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 implementating 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	class implements interface 1 & 2
default void m1(){	@Override
}	public void m1(){
	Interface.super.m1();
	}

* 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
- ~~Ex~~ 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.

S.D.S
Page :
Date : / /

* Functions in Java 8.

→ ① Function is predefined functional interface.

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

R apply(T t);

③ Takes one input and returns one output.

Syntax :- Function<T1, T2> f = //code;

e.g.

Function<Integer, Integer> sq = i -> i * i;

S.o.pln(sq.apply(5)); → 25.

* Predicates vs Function

Predicate	Faction
- Return type → Boolean	- Return Type → Object
- accepts single argument	- accepts single argument
i.e. Predicate<T> & return boolean only	but Function<T, R>.
- test() method.	- apply() method.

* Functional Chaining

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

e.g.

① andThen

f1.andThen(f2).apply(Input); → first f1 & then f2

② compose

f1.compose(f2).apply(Input); → first f2 then f1.

- Multiple functions can be chained togethe like:

f1.andThen(f2).andThen(f3).compose(f4).apply(Input);

* Consumer functional Interface.

→ Consume<T> → It will consume Item but never return anything.

e.g. take any object & save its details in DB. & don't return anything.

Syntax:-

```
Interface Consumer<T>{
    public void accept(T t);
}
```

* Consumer chaining.

→ - We can combine/chain multiple consumers together with andThen.

- No compose() in consumer

- Syntax:-

c1.andThen(c2).~~apply~~ accept(Input) → first c1 then c2.

Consumer<T>

* Supplier functional interface.

→ - Supplier<R> → It will just supply required objects and will not take any input.

- syntax:.

```
Interface Supplier<R>{
    public R get();
}
```

- No chaining available as it doesn't take any input

There is no input for Supplier, Hence so BiSupplier ~~exist~~ not available.

* BiConsumer, BiFunction, BiPredicate ~~& B+~~

→ BiPredicate :-

BiPredicate <Integer, Integer> p = //code ;

BiFunction :-

BiFunction<Integer, Integer, Integer> f = //code ;

2 input, 1 Return Type.

BiConsumer :-

BiConsumer <Integer, Integer> = //code ;

* Streams

→ - When we want process bulk objects of collection then go for streams.

- Its a special iterator class that allows processing collections of objects in a functional manner.

e.g.

List.stream().filter(x -> //code).collect(Collectors.toList());

* Difference b/w Streams (Java 8) & java.io.Streams ?

→

Streams (Java 8)	java.io.streams.
- Java 8 streams are not related to files, they are related to collection object.	- IO streams is a sequence of characters or binary data used to written a file or read data from file.

* Difference b/w streams and collection.

→

Collection	Streams
To represent group of entity as a single entity then use collection.	- To perform operation on bulk objects in collection then use streams.

* Steps to create and process stream.

→ ① Streams s = collectionObj.stream(); →Stream object.

② Once we get stream object we can process the object of collection.

③ Processing of stream consist of 2 steps.

i) Configuration of stream.

ii) Processing that configuration.

④ Configuration can be done by i) Map ii) filter.

e.g.

```
Stream s = arraylist.stream();
s.filter(); || s.map();
```

* filter the stream objects :- filter out from existing collection

→ Stream s = collectionObj.stream().filter(i -> i%2==0);

s.forEach(x -> s.o.println(x));

↓

Arraylist.stream().filter(i -> i%2==0).forEach(x -> s.o.println(x));

* Map the stream objects.

→ When we want to create new object agaist each existing stream object based on some function.

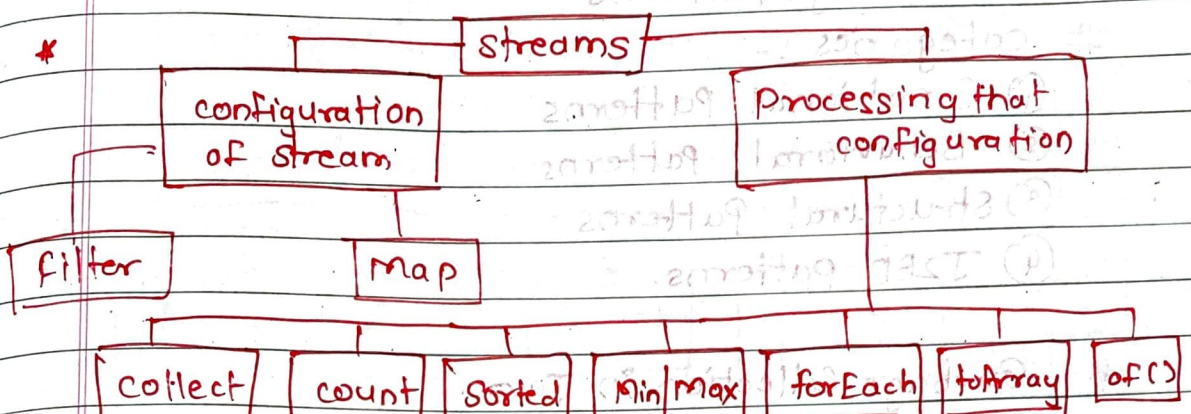
e.g.

```
stream s = arraylist.stream().map(i -> i*i);
s.forEach(x -> s.o.println(x));
```

* filter vs Map

→ filter → filter & get objects in original collection

map → new object of collection is created.



* Interface changes.

→ ① Static Method, ② Default Method } In Java 8, we can have default method & static methods for method declaration & method implementation in the interfaces.

* Runtime Polymorphism of data members.

① Instance variable :- It is not possible to achive the overridding with instance variable.

e.g.

```
Parent p = new child();
p.variable;
```

→ This will give value of parent class variable only

→ So, it runtime polymorphism or overridding is possible only with the method.

* Java Design Patterns

categories :-

1. Creational Patterns
2. Behavioral Patterns
3. Structural Patterns
4. J2EE patterns.

* Garbage Collection in Java.

→ - Garbage collection in java is an automatic process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused object.
- Garbage collection provides automatic removing the burden of manual memory allocation & deallocation

* Where objects created in memory ?

→ - When object is created, it is always stored in the Heap space and stack memory contains the reference to it.
- Stack memory only contains local primitive variables & reference variables to objects in heap space.

* Which part of memory is involved in Garbage Collection?

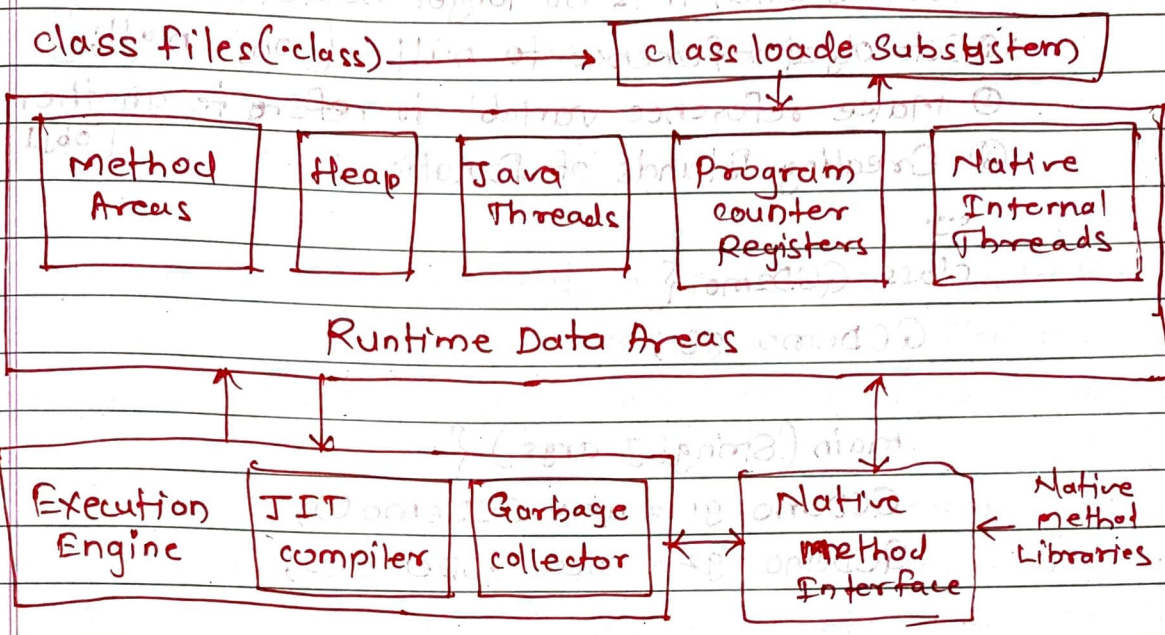
→ Heap.

* Garbage collector

→ Garbage collector is program which scans the Heap memory and find out which object is currently alive. If some object is no more referenced, Garbage collector dumps it collects it, dumps it and used that memory to store another object.

* Mark-Sweep algorithm is most common-garbage collection algorithm.

* JVM Architecture.



* Who manages Garbage collector.

→ JVM.

- JVM runs the Garbage collector when it realize that the memory is running low.

* How can the Garbage collection be requested?

→ ① Call the System class System.gc() method which will request the jvm to perform GC. [System.gc();]

② The methods to perform GC are present in the Runtime class provided by java.

The Runtime class is Singleton for each java main program.

The method getRuntime() return singleton instance of Runtime class

The method gc() can be invoked using this instance of Runtime to request the Garbage collection.

[Runtime.getRuntime().gc();]

* What are the different ways to make an object eligible for GC when it is no longer needed?

→ ① Set object reference to null [obj = null;]
② Make reference variable to refer to another object. [obj1 = obj2]
③ Creating Islands of Isolation.

e.g.

```
class GCDemo {
  GCDemo g3;

  main (String[] args) {
  GCDemo g1 = new GCDemo();
  GCDemo g2 = new GCDemo();

  g1.g3 = g2;
  g2.g3 = g1;
  g1 = null;
  g2 = null;
```

* Purpose of overridding finalize() method?

→ ① Finalize method in Java also called finalizer is a method defined in Java.lang.Object.
② It is called by Garbage collector just before ~~calling~~ collecting any object which is eligible for GC.
③ finalize() provides last chance to object to do cleanup & free remaining resources.
④ garbage collector calls finalize() method only once just before garbage collection.

* Daemon Thread

→ ① It is low-priority thread which runs behind the application.

Hotspot → oracle's JVM
Generational GC strategy → used by oracle for garbage collection { categorize objects by age.

④ It is started by JVM
⑤ This thread stops when all non-daemon/foreground threads stop.
⑥ Garbage Collector is daemon Thread.

* How Garbage collection works?
→ GC works in 2 steps:-
① Marking :- unreferenced objects in heap are identified and marked as ready for garbage collection.
② Deletion / Deletion+ compaction :- marked objects are deleted.
compaction means after Deletion of object memory that was occupied by deleted objects is compacted i.e. remaining objects are in contiguous blocks at the start of heap memory in sequential manner.

* Processing of streams.
→ ① collect()
→ collect the elements of stream after filtering or mapping and them to required collection.
e.g. List.stream().filter(i->code).collect(Collectors.toList());

② count()
→ count() method returns count of collection object
e.g. list.stream().filter(i->code).count();

③ sorted()
→ sorting inside stream.
① e.g. list.stream.filter(i->code).sorted(); ← natural sorting
OR list.stream.sorted();
② list.stream.filter(i->code).sorted((i1,i2)→ i2.compareTo(i1));
→ Custom sorting.

③ Min, Max
→ Min and max will return value based on the defined comparator.
e.g. list.stream().min((i1, i2) → i1.compareTo(i2)).get();
or
max.

④ forEach()
→ Does not return anything.
- This method takes lambda Expression as an argument
e.g. list().stream().forEach(x → s.o.p ln(x));

⑤ toArray()
→ used to copy elements present in stream to Array.
e.g. list.stream().toArray();
- Return type is corresponding Array object.

⑥ of() ⟶ of(T... values).
→ Stream concept is not applicable just for the collections its also applicable for "Any Group of value."
- It can also use for arrays.
→ of() method can take any group of elements and convert them to Stream.
e.g.
① Stream.of(1, 11, 111, 1111).forEach(x → s.o.pln(x));
② String[] strArr = {"A", "B", "C"};
Stream.of(strArr).forEach(x → s.o.pln(x));

* Load factor in Hashmap / Hashtable & its default value?

→ It is a measure that decides when to increase the Hashmap capacity. The default load factor of Hashmap is 0.75f i.e. 75% of the map size.

* How does bucket index is calculated in Hashmap?

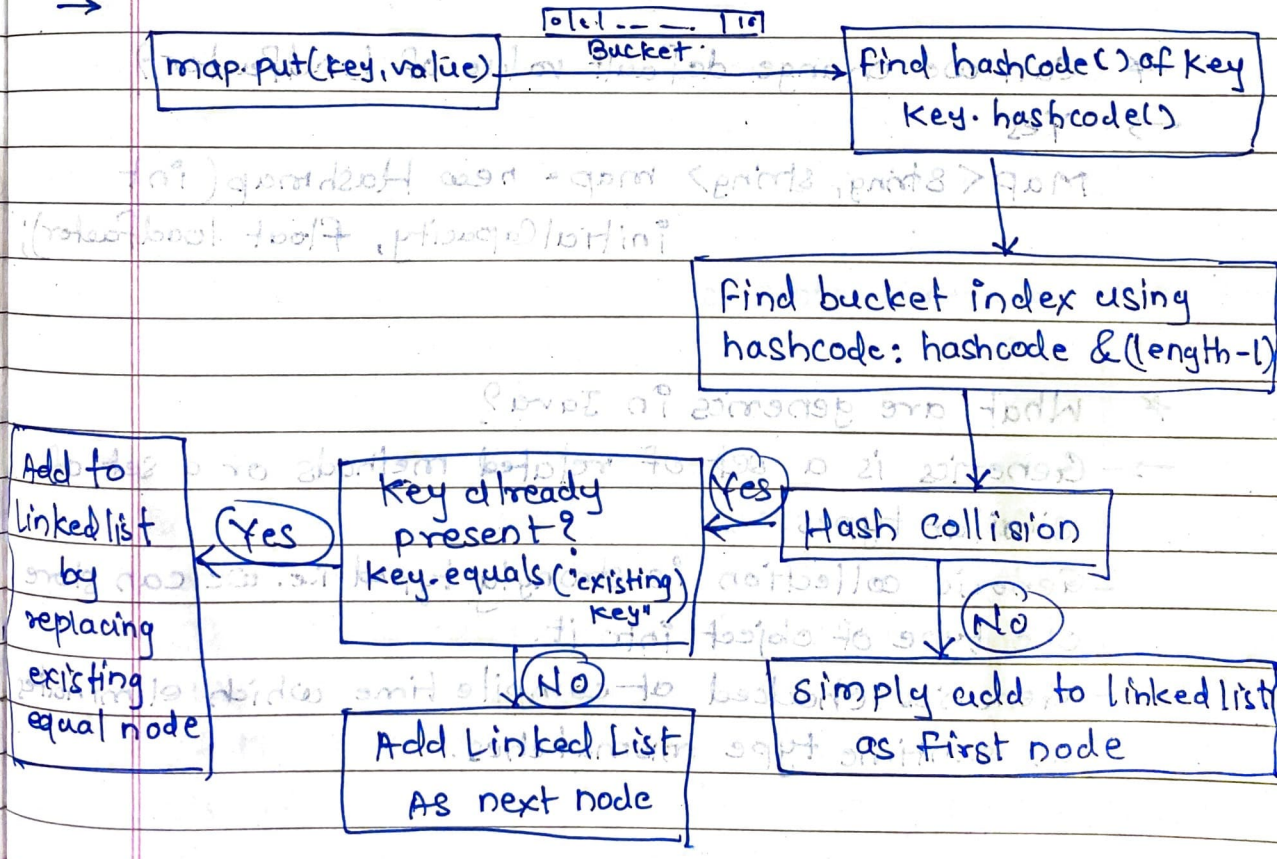
→ Hashcode & length(bucket) - 1

* What is Hash collision?

→ It is a situation where two or more key objects produce the same hash value & hence point to same bucket location or array index.

* Internal coorking of Hashmap.

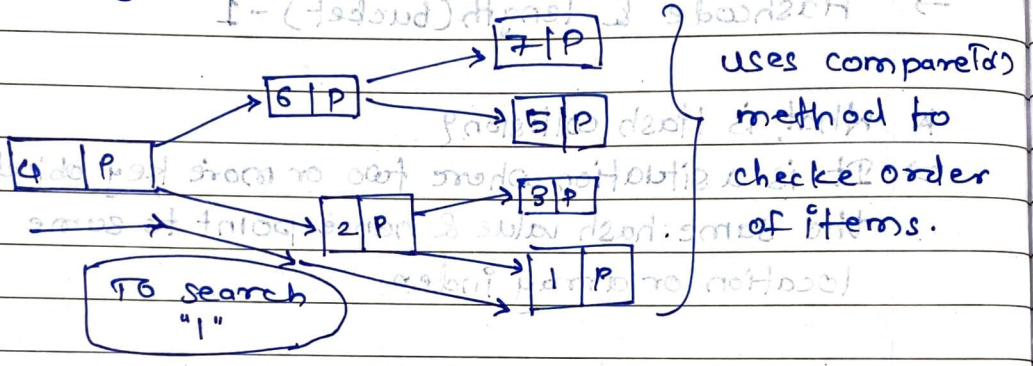
→



* Time complexity of hashmap get method → O(1)

* What is enhancement in Hashmap in Java 8?

→ - In Java 8, Hashmap replace linked list with binary (B-tree) tree when the number of elements in bucket reaches certain threshold.
- Binary tree structure.



* Can we change default value of loadfactor?

→ Yes

```
Map<String, String> map = new Hashmap( int
                          initialCapacity, float loadfactor);
```

* What are generics in Java?

→ - Generics is a set of related methods or a set of similar types.
- Generic collection is strongly typed i.e. we can store one type of object into it.
- Generics checked at compile time which eliminates an runtime type mismatches.

* What is erasure type in collections?
→ It is a process in which the compiler replaces a generic parameter with actual class or bridge method

* Serialization
→ - It is a mechanism of converting the state of an object into byte stream.
- The byte stream created is platform inelependent.
- To make object serializable we can implement the java.io.Serializable interface.
- The ObjectOutputStream class contains writeObject() method for serializing an object.
- Serializable is a marker interface. It used to "mark" java classes so that classes may get certain capability.

* Deserialization.
→ - It is a reverse process where byte stream is used to recreate the actual java object in memory.
- This mechanism is used to persist/save the state of the object.
- The ObjectInputStream class contains readObject() method for deserializing the object.

* SerialVersionUID
→ - The serialization runtime associates a version number with each Serializable class called a serialVersionUID.
- It is used during Deserialization to verify the sender and receiver of serialized object.
- If the receiver has loaded a class for the object that has a different UID than that of corresponding sender's class, the Deserialization will throw InvalidClassException.

- A Serializable class can declare its own UID explicitly by declaring a field name. It must be static, final and of type long.
e.g. private static final long serialVersionUID = 42L;
It can be any access modifier

* Concurrent Modification
→ Means to modify an object concurrently when another task is already running over it.

* How fail-fast iterator works?
→ To know whether the collection is structurally modified or not, fail-fast iterator use an intenal flag called modCount.
- modCount is updated each time a collection is modified.
- Fail-fast iterators checks the modcount flag whenever it gets nextValue (using next() method).
- If modCount has been modified after this iterator has been created, it throws ConcurrentModificationException.

* Difference between sleep and wait?
→

wait()	sleep()
- belongs to Object class.	- belongs to Thread class.
- wait() method releases lock during Synchronization.	- sleep() method does not release the lock of object synchronization.
- wait() method tells the calling thread to wait until another thread invokes the notify() or notifyAll().	- sleep() method is used to pause the execution of current thread for a specificied time.
- Not a static method	- static method.

* Internal working of ArrayList.

→ - ArrayList is a resizable array implementation in Java.
- The data structure of Arraylist is an array of object class and Object array is transient.
- ArrayList implements List interface, RandomAccess, cloneable and java.io.serializable interfaces (marker)

e.g.
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, cloneable, java.io.serializable

- Internally Arraylist uses array of object[] for the operations like adding, deleting & updating elements.

e.g. transient Object[] elementData;

- To create an ArrayList, there are 3 types of constructors in Java.

① Arraylist :- To initialize an empty List with an initial capacity of 10.

- // Constructor with empty ArrayList
public ArrayList() {
this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;

② ArrayList(int capacity) :- To create ArrayList with the initial capacity
- public ArrayList(int initialCapacity) {
if(initialCapacity > 0) {
this.elementData = new Object[initialCapacity];
} else if(initialCapacity == 0) {
this.elementData = EMPTY_ELEMENTDATA;
} else {
throw new IllegalArgumentException();
}

③ ArrayList(Collection<? extends E> c): To create an array list initialized with the elements from the collection passed into the constructor

```
- public ArrayList(Collection<? extends E> c) {
      elementData = c.toArray();
      if ((size = elementData.length) != 0) {

          // c.toArray might not return Object[]
          if (elementData.getClass() != Object[].class) {
              elementData = Arrays.copyOf(elementData, size,
                                          Object[].class);
          }
      } else {
          // replace with empty array.
          this.elementData = EMPTY_ELEMENTDATA;
      }
  }
```

Add method of ArrayList:

→ When we call add() method in arraylist then internally checks for the capacity to store the new element or not.

If not then size is incremented by 50%.

- int newCapacity = (oldCapacity * 3)/2 + 1;

* ExecutorService

→ - It is the interface which allow us to execute tasks on threads asynchronously.

- present in java.util.concurrent package.
- Helps in maintaing a pool of threads & assign them tasks.

* Object cloning in Java

→ - It is a way to create exact copy of an object.
The clone() method of object class is used to clone an object.

→ The java.lang.cloneable interface must be implemented by the class whose object clone we want to create.
Otherwise, clone() method throws cloneNotSupported-Exception. [public Object clone() throws cloneNotSupportedExcep]

* Why to use clone() method if we can directly create clone using new operator?

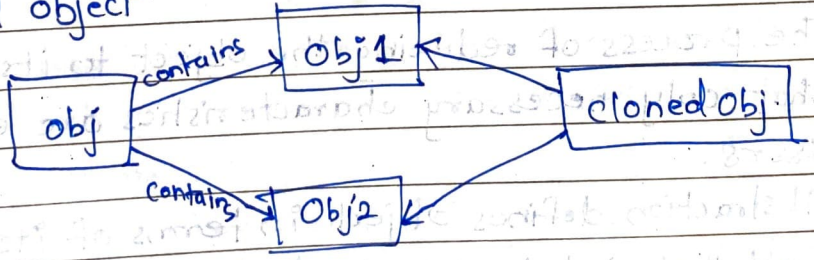
→ - clone() method saves extra processing task for creating exact copy of an object.

* Types of cloning.

→ ① Shallow cloning :-
- This default implementation of clone method.
- Creates a shallow copy of Object i.e. new instance is created which copies all the fields to new instance & returns new object of type "Object".
- And this type needs to be ~~impli~~ explicitly type casted.

e.g.
obj2 = (Object) obj1.clone();

- Shallow clone is copying the ~~reference~~ pointer to the object i.e. new object is pointing to same reference of old object.



② Deep clone

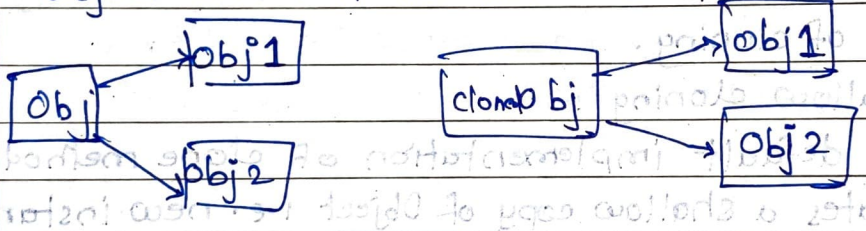
→ This clone also copies all the fields of source object like shallow clone but unlike shallow copy if the source object has any reference to object as a fields then a replica of the object is created.

→ i.e. source and destination objects are independent of each other.

e.g.

```
Employee emp = (Employee) super.clone();
emp.dept = (Department) dept.clone();
```

- In Deep clone, new memory is allocated for the object and contents that are copied.



* How create ~~cust~~ custom Exception?

→ ① Create custom exception class and extend that with Exception class.

② Create argumented constructors ~~that~~ having message as parameter.

* Abstraction

→ - The process of reducing the object to its essence so that only necessary characteristics are exposed to users.

- Abstraction defines object in terms of its properties (attributes), behaviour (methods) and interfaces.
- Purpose of abstraction is hiding the unneccary details from the users.

Page : S.D.S.
Date : / /

* Encapsulation.

→ - Encapsulation is a mechanism of wrapping data (variables) and code acting on the data (methods) together into a single unit.
- In encapsulation, the variables of class will be hidden from other classes, and can access only through the methods (setter & getter methods).

* Constructors

→ - A constructor in Java ~~first~~ is similar to method that is invoked when an object of class is created.
- It has the same name as that of class and doesn't have any Return type.
- Used to initialize the object.

* ~~Array~~List vs Linked List

→ ArrayList	Linked List
→ - Internally uses a dynamic array to store elements.	- Internally uses a doubly linked list to store elements.
- act as a list only because it implements List only	- act as a list and queue both because it implements List and Deque interfaces.
- better for storing & accessing	- better for manipulation.

* What is static block and what is use of it?

→ - A static block is used to initialize the static variables
- This block gets executed at the time of class loading in the memory
- A class can have multiple ~~class~~ static blocks which will execute in the same sequence as they have written.

* Use of constructor in abstract class
→ To initialize the non-abstract methods and instance variables.

* How can we call a constructor of abstract class?
→ - We can't call an abstract class constructor with a class instance creation.
- Constructors of abstract classes can be only called within subclass constructors.

* Externalization in Java.
→ - It is used when we need to customize the serialization mechasism.
- To ~~extend~~ externalize the class, we need to implement Externalizable Interface & it has two methods.
① writeExternal() → for custom serialization
② readExternal() → for custom deserialization.

- for primitive types we can use.
readBoolean(), readByte(), readInt(), readLong() for deserialization.
- And writeBoolean(), writeByte(), writeInt(), writeLong() for serialization.

* SpiltIterator in Java.
→ - It is one of the four iterators in Java like Enumeration, Iterator, ListIterator and spiltIterator.
- Like Iterator & ListIterator it is used to iterate elements one-by-one from a List implemented object.
- Unlike Iterator and ListIterator

(i) It supports parallel programming or processing of data and also sequential processing of data.
(ii) And provides better performance.
(iii) It is iterator introduced in Java 1.8 for Collection and stream API except Map interface.

* Internal working of Hashset?
→ - Hashset uses Hashmap internally to store its object.
- Whenever we create a Hashset object, one Hashmap associated with it is also created
- Element we are adding to the Hashset is used as the key in the backing Hashmap. And for the value dummy value is used i.e. object PRESENT = new Object()
- So, Hashset's contain(element) simply calls Hashmap's containskey(element).

* How to break singleton pattern?
→ (1) Reflection:-

```
Class<?> singletonClass = Class.forName("pakage.SingletonClassName");
Constructor<SingletonClass> c = singletonClass.getDeclaredConstructor();
                                (Type cast with Constructor<Singleton>)
c.setAccessible(true);   → Imp
Singleton brokenInstance = c.getInstance();
```

i.e. by creating new constructor and setAccessible(true)

(2) Serialization:- Serialization is used to convert an object of byte stream and save in a file or send over the network. So, if we serialize object and deserialize that it will create new instance.