* 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.

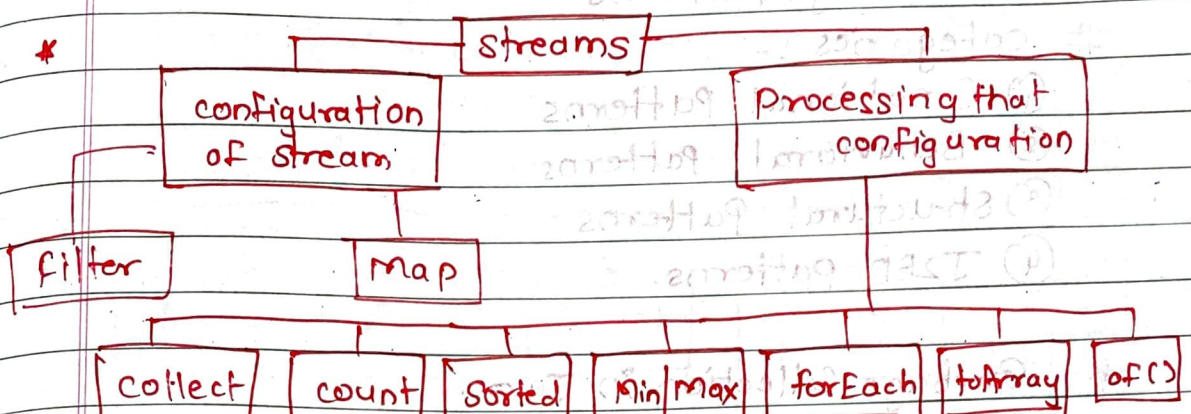
e.g.

```
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?

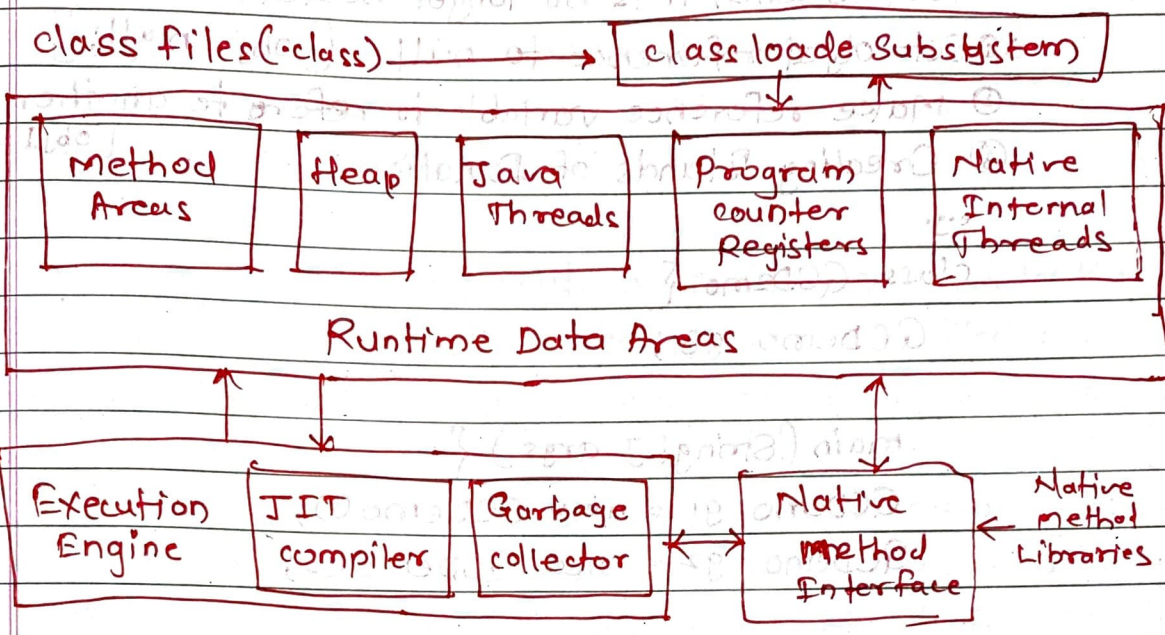
→ Heap.

* 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?

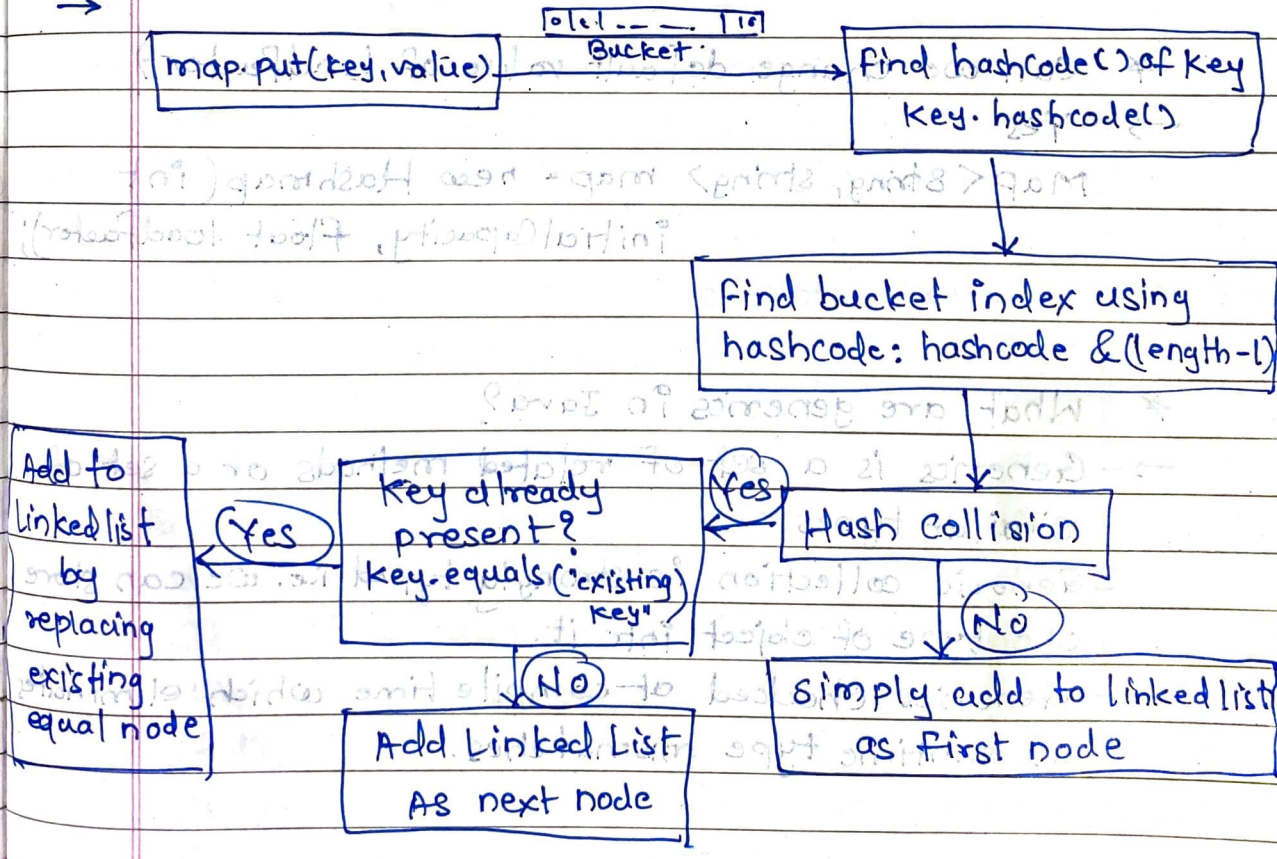
→ Hashcode & length(bucket) - 1

* 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.

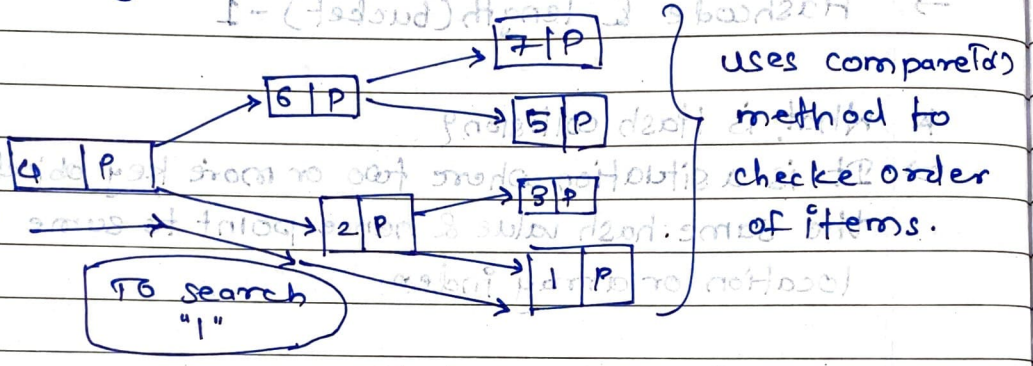
→



* 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?

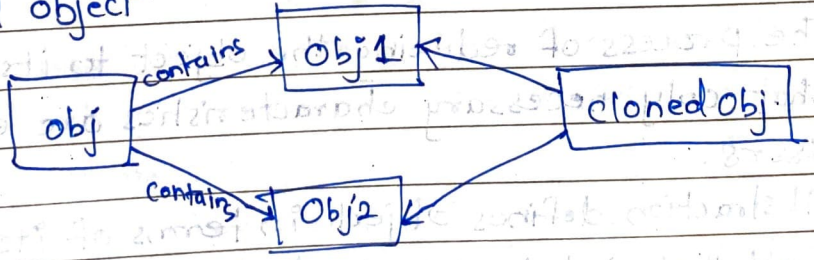
→ - clone() method saves extra processing task for creating exact copy of an object.

* 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

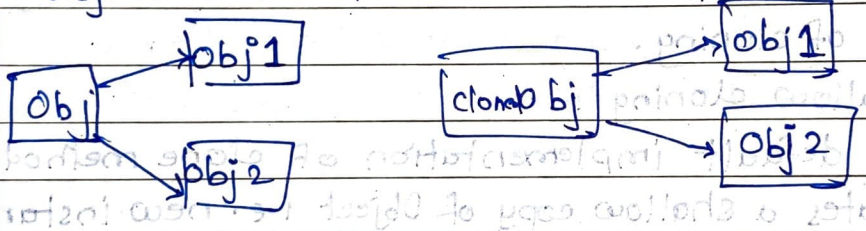
→ This clone also copies all the fields of source object like shallow clone but unlike shallow copy if the source object has any reference to object as a fields then a replica of the object is created.

→ 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.