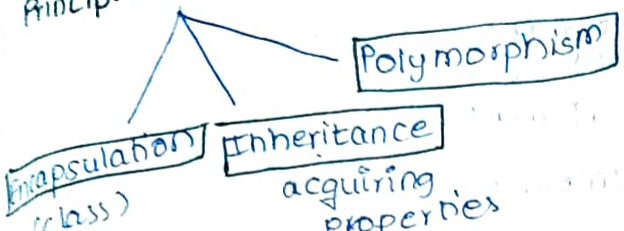
Java Summary

Unit-1

Principles of OOP



Polymorphism — ability to take more than 1 form
Eg: add method.
+ ⇒ numbers addition
↳ strings concatenation
same operation may exhibit behavior in diff instances.
It depends on type of data used.

Encapsulation (class) — binds data & methods into single unit

Inheritance — acquiring properties of the objects of another class
Advantage: Code reusability

POP	OOP
In this approch, the problem is considered as a sequence of tasks to be done. Functions are written to acomplish the task. Primary Focus: Functions & data. Languages: COBOL, FORTRAN, PASCAL, C Characterstics: • Emphasis is on doing actions • Large program are divided into smaller programs called functions • Most of functions share global data • Data move openly around the program from func to func • functions transform data from one form to another • Top-down design	decompoes the problem into a no. of ~~case~~ entities called objects & then builds data & method based on entities In this approch, it provides a way of modularizing programs by creating portioned memory area for both data & methods that can be used, create copies on demand as templates Languages: JAVA, C++ Characterstics • Emphasis is on data. • Programs are divided into wt are known as methods (objs) • Data is hidden. • Methods that operate on the data of an obj are tied together • Objects communicate with each other using methods • Reusability. • OS are designed such that they characterize the obj • Bottom-up design.

Java features or Buzz words.

1) Dynamic & Extensible:
capable of dynamic linking a new class libraries methods & objs
→ import statement.

8) High performance:
(1) speed as it is interpreted
(2) comparable to native C/C++
(3) Multi Threading enhances execution speed

1. Simple, small & Familiar:
↳ it contains many similar features other languages like C & C++ have.
↳ removes complexity as it doesnot use pointers, go to's, storage class multiple inheritance, operator overloading etc.

2 Compiled & Interpreted.:
Java code
↓ compiler
Byte code
↓ interpreter
Executable code.

Java is both compiled & interpreted language.

3. Platform independent: easily moved from one platform to another, anytime, anywhere.
↳ Changes or Updates in OS, processors, System resources doesn't force to changes in java programs

byte code instruction that can be implemented on any machine
size of the primitive data types are machine independent

4. Object oriented: (Pure or True) Almost everything in Java is object

5. Robust (strong) & secure: It provides many safeguards to ensure reliable code.
→ has strict compile time & run time checking for data types
→ has garbage collector
→ has exception handling, which captures series errors & eliminates risk of crashing system

6. Distributed: (Distributed language)
Java applications can open & access remote obj on Internet as easily as they can do on local system. This allows multiple programmers at multiple locations to collaborate & work together on single project

7) Multithreaded & Interactive: → Run time comes with Multiprocess synchronization for smoothly running interactive sys
↓
Multiple threads working same time

Sample program structure.

// → single line
/* */ → multi line
/** */ → documentation comment to explain why & what abt objects.

Documentation section

Package statement section

The statement declares package name informs the compiler that the classes defined belong to this package (optional)

Import statements

→ To load builtin classes or class that are part of other packages into our program

Interface statements

includes group of method declarations (optional)

class definition

→ can have multiple classes

main method class

{ main method definition

}

main method creates objs of various classes & establishes connection b/w them

stand alone program requires a main method as its starting point (essential part)

on reaching the end of main program terminates control pass back to OS.

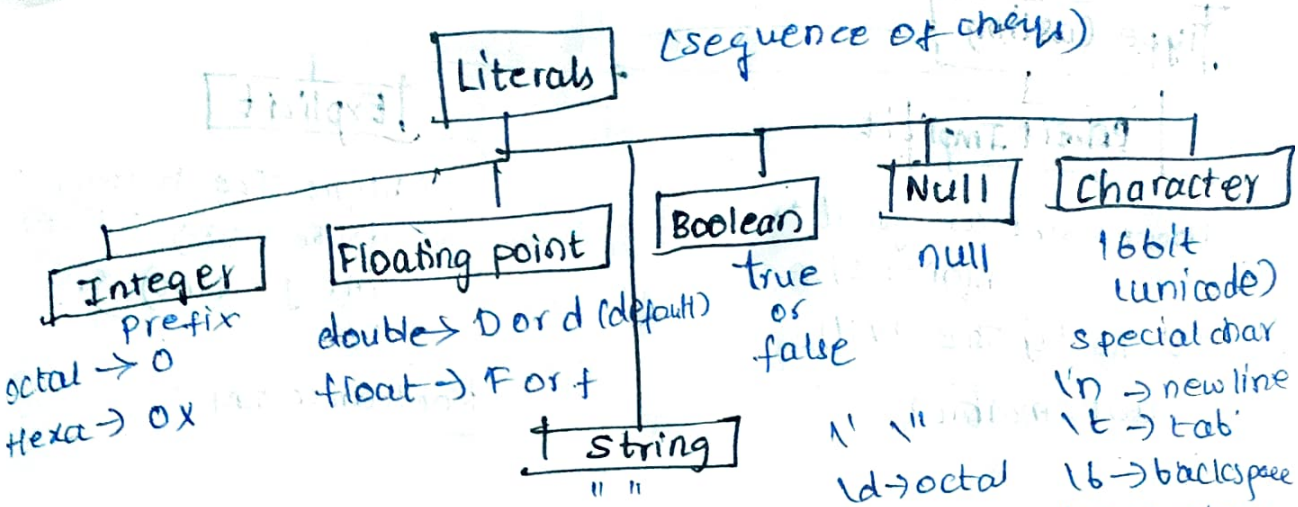
Data types

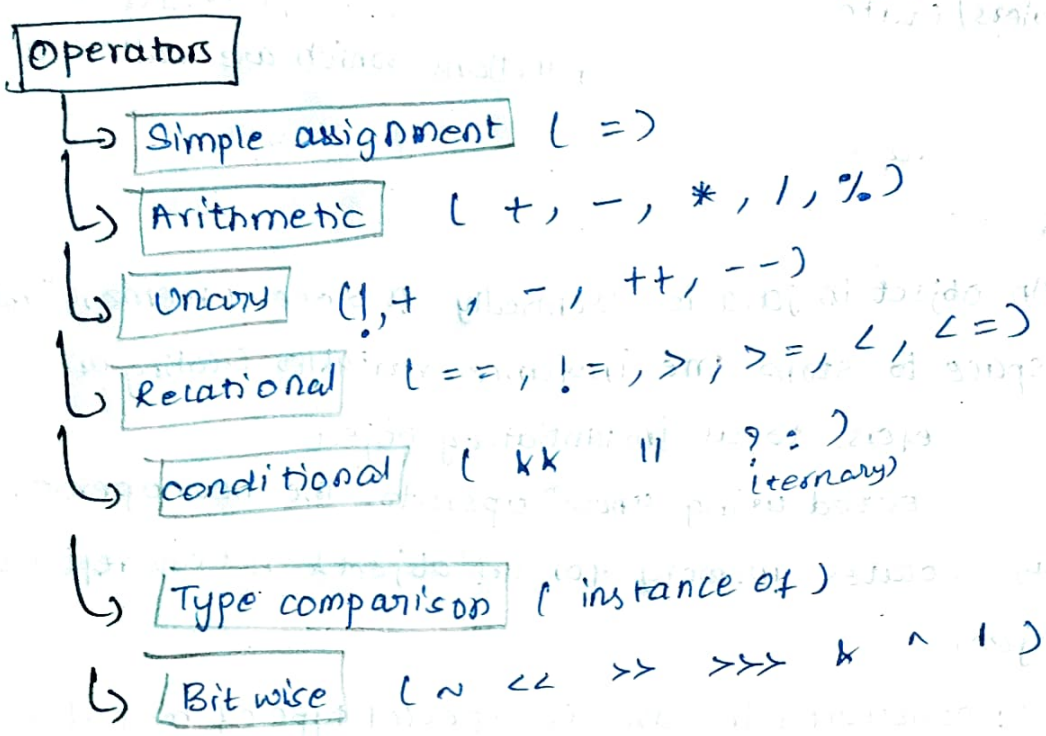
Primitive

- **Integers** — Width
  - byte — 8 — -2^7 to 2^7-1 (raw binary data)
  - short — 16 — -2^{15} to $2^{15}-1$ (rarely used in java)
  - int → loops — 32 — -2^{31} to $2^{31}-1$
  - long — 64 — -2^{63} to $2^{63}-1$ (big whole numbers)
- **Floating point numbers**
  - float — 32 — 1.4e-045 to 3.4e+0.38 (single precision)
  - double — 64 — 4.9e-324 to 1.8e+308 (faster than float representation used for high speed calculations)
- **character** — 16
  - char — 16 bits (ASCII + ISO_Latin-1)
- **Boolean** (logical values)
  - boolean (true or false)

Non-primitive

- Classes
- String
- Arrays
- Interfaces





Precedence & Associativity

1.	[] () (method call) . (member access) (array ind)	left to right
2.	++, --, +, -, ~, !	right to left
3.	(type cast) new	right to left
4.	* / %	left → right
5.	+ - , + (concat)	L → R
6.	<<, >>, >>>	L → R
7.	<, <=, >, >=, instance of	L → R
8.	==, !=	L → R
9.	&, ^, |, &&, ||	L → R
10	? :	R → L
15.	short hand assignment	R → L

Type Casting

Implicit
lower size assigned to higher size
(done by the JVM)
(automatic)

Explicit
(higher size to lower size)
double x = 3.5;
int y = (int)x;

Boolean casting

Class: A class can be defined as a template/blue print that the behaviors/states that object of its type support. Its a combination of data items & functions which are called as members of class

~~Datak~~

Object: An object in java is essentially a block of memory that contains space to store the instance variables. Creating an obj is also refers to as instantiating object.
Objects are created using "new" operator. The new operator dynamically allocates memory for an object & returns reference to that object.

Constructor: constructor in java is special type of method is used to initialize the obj.
name → class names
no return type

→ default (no params) (provides default values to vars

→ parameterized (has params) (diff vals to distinct obj

Constructor overloading: differentiated basis of their type of params or no. of param.

diff from method overloading

done to create obj in different ways.

(hierarchical classification) **Inheritance** Unit-2

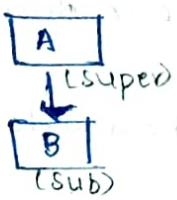
→ process by which objects of 1 class acquire the properties of other class.

Uses: 1. Reusability 2. Abstraction

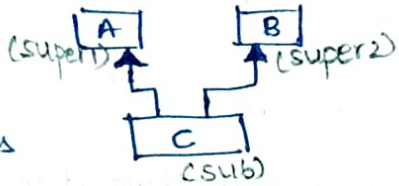
→ 'extends' keyword is used.

Types

Single (simple)
one subclass is dervied from one superclass

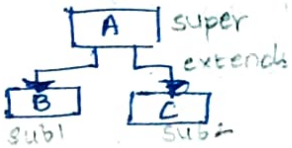


Multiple
Deriving more than 1 super class



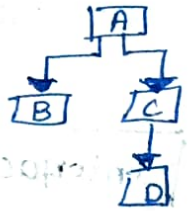
due to confusion not allowed in Java.
So, interfaces are introduced
Multiple interfaces can be implemented to achieve multiple inheritance.

Hierarchical
one base class but many derived classes



Multilevel
class is derived from derived class

Hybrid
combination of multiple inheritances



hierarchical + multilevel.

Super: used to refer to its immediate super class

2 forms
- → super(args-list) → calls super class's constructor.
- → super.member → to access super class member, that has been hidden by a member of a subclass (may be variable or method)

Abstract class:

→ A method without body is called abstract method

→ syntax: abstract datatype methodname(param-list);

→ A class that contains 0 or more abstract method is called abstract class (should be declared abstract)

→ It is possible to implement abstract methods differently in subclasses of abstract class. (based on the need of subclass)

→ It is not possible to create objects to abstract class but we can create reference of abstract class type.

static → used for class members
final → constant

many forms.

POLYMORPHISM

DYNAMIC (Runtime)
(also called as dynamic binding)

1. Method overloading
2. Method overridding

→ same name with diff params

→ method has same name in Sub class & super class with same params

STATIC (compile time)
(also called as static binding)

1. Method overloading
2. Method overridding with static methods

Interface: contains a group of constants & method declarations (no implementation)

→ allows you to specify what a class must do, but not how to do it
→ all methods must be public & abstract (by defaut)
→ constants ⇒ public, final & static
→ methods declared with no bodies end with semicolon.
→ an interface is used to allow unrelated objects to interact with each other

Implementing interface.

* "implements" keyword.
* Interface can inherit another by using "extends" keyword.

Package: Packages are containers of classes that are used to keep the class namespace compartmentalized.

Packages are stored in hierarchical manner & are explicitly imported into new class definitions

Built-in
already available in java language.
Eg: java.lang, java.util

User defined
→ created by users of java.
→ same way as built-in packages are imported

Creating a package: quite easy

→ simply include package command as 1st statement in a java source file.
→ A package statement defines a namespace in which classes are stored

→ If u omit package statement the classes are put into the default package which has no name.

Syntax: package packagename;

compile:
javac -d directory_path classname.java
- -d → create a package
- directory_path → in the directory

* 'import' keyword is used to import packages into current class

Run:
java packagename.classname

4 access modifiers.
(private, public, protected & default).

	Private	Default	Protected	Public
Same class	Y	Y	Y	Y
Same package subclass	N	Y	Y	Y
Same package non-subclass	N	Y	Y	Y
Different package subclass	N	N	Y	Y
Different package non-subclass	N	N	N	Y

ERRORS
- compile-time
  - Syntactical
  - Eg: forgetting ;
- run-time [Exceptions]
  - abnormal termination of program
  - Input errors
    - Unexpected i/p
    - (prompt user to enter correct i/p)
  - System errors
    - hardware errors
    - (Beyond programmer's control)
  - Logical errors
    - logically incorrect
    - given wrong answer
    - (Debugging)

EXCEPTION
→ abnormal condition that arises during execution of a program that causes to deviate from the normal flow of execution path.
→ Exception handling means handle the exceptions by the program to recover the computer from malfunction due to exception.

EXCEPTION HANDLING

TRY
- monitor the statements enclosed within it.
- defines the scope of any exception associated with it.
- detects exceptions

CATCH
- contains series of Java legal statements
- holds & deals with exception
- either fix or abort the execution

THROW
- used to manually throw an exception
- "throw" keyword is used

THROWS
- Any exception that is thrown out of a method must be specified as such by a throws clause

FINALLY
- Any code that shd absolutely must be executed after a try block completes is put in a finally block.
- After the exception handler has run, the runtime system passes control to the finally block.

Benefits of Exception Handling:

- allows you to fix the error
- prevents the the program from automatically terminating.
- adopts robustness to program

* Sepearating Error Handling code from "regular" one:

provides a way to separate the details of what to do when something out-of-the-ordinary happens from the normal logic flow of the program code.

* Propagating Errors up the call stack: let's the corrective action to be taken at higher level. This allows the corrective action to be taken in the method that calling that one where an error occurs.

* Grouping Errors types & Error Differentiation

Allows to create similar hierarchical structure for exception handling so groups them in logical way.

User Defined: extends Exception class
- override tostring method.

UNIT-3

MULTITHREADING

(multi-tasking)

defines separate execution ⇒ Thread (two or more parts that can concurrently)

PROCESS-BASED

Process: program that is executing

→ feature that allows ur computer to run 2 or more programs concurrently

→ Program is smallest unit of code

→ Big piture

THREAD-BASED

→ single program can perform 2 or more tasks simultaneously.

→ thread is the smallest unit of dispatchable code.

⇒ handles details.

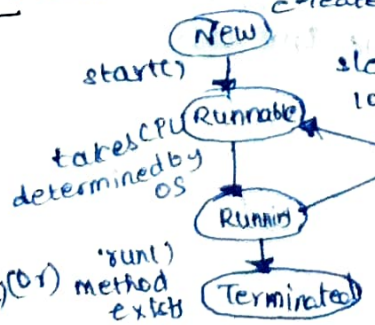
max use of the CPU, bcz idle time can be kept minimum

MULTI TASKING	MULTI THREADING
→ More than 1 process running simultaneously	→ More than 1 thread running simultaneously.
→ Its a program	→ It's part of program
→ It's a heavy weighted process	→ It's a light weighted process
→ Processes are divided into threads	→ Threads are divided into sub threads
→ Inter process communication is a difficulty	→ within the process threads are communicated.
→ context switching b/w process is costly	→ context switching b/w threads is cheaper
→ It is controlled OS	→ controlled by JVM
→ generalized form of multithreading	→ Specialized form of multitasking

THREAD LIFE CYCLE

: contains several states

- New thread
- Runnable
- Running
- Blocked
- Dead



CREATING A THREAD (2 ways)

- IMPLEMENT RUNNABLE
- EXTEND THREAD

① Implement Runnable:

* Runnable abstracts a unit of executable code
* To implement Runnable, a class need only implement a single method called run()

```
public void run()
```

Thread will begin at run(). Will end after run() returns

* After implementing Runnable, instantiate an object of type Thread from with that class.

Thread defines several constructors

```
Thread (Runnable threadOb, String threadName)
```

- threadOb: instance of class that implements Runnable
- threadName: thread name

* After the new thread is created, it will not start running until you call its start() method, which is declared within Thread.

start() executes a call to run().

Extending thread:

* Create a new class then extends Thread, & then to create an instance of that class.
* The extending class must override the run() method, which is the entry point for the new Thread.
* It must also call start() to begin execution of the new thread
* public Thread(String threadName)

Creating multiple threads: multiple threads is a concept of creating multiple threads.

Inter thread communication

* Threads are created to carry out light-weight process independently.
* In certain situations, 2 or more threads may use an object as a common resource.
* In order to avoid mix-up of the task of one thread with that of another thread, the resource obj is synchronized.
* When one thread is using the synchronized objet, the monitor is locked & another thread needing to use the object should keep waiting.
* A synchronized obj may have more than one synchronized method.
* But when a synchronized object is used by 1 thread, it cannot be accessed by any other thread, even if a different method of shared object is needed.
* It may happen that only after an action has taken place in 1 thread, the other thread can proceed.
* If currently running thread can proceed only after an action in another non-running thread the running thread has to keep waiting forever. To avoid such problem, java provides inter thread communication methods, which can send messages from one thread to another thread, which uses the same obj.

The methods used for inter-thread communication.

wait(): This method makes the calling thread to give up the monitor & go to sleep until some other thread wakes it up.

notify() This method wakes up the first thread which called wait() on the same object.

notifyAll(): This method wakes up all the threads that called wait() on the same object.

```java
class NewThreadExtendingThread extends Thread{

    NewThreadExtendingThread() {
        super("Demo Thread");
        System.out.println("Child thread created");
        start();
    }

    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Child thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Child thread interrupted");
        }
        System.out.println("Child thread exitied");
    }
}

public class ThreadExtendingThreadDemo {
    public static void main(String[] args) {
        new NewThreadExtendingThread();
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Main thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }
        System.out.println("Main thread exitied");
    }
}
```

```java
// example of creating a thread "implementing Runnable interface"
class NewThreadRunnable implements Runnable{
    Thread t;
    NewThreadRunnable() {
        t = new Thread(this, "Demo thread");
        System.out.println("Child therad created");
        t.start();
    }

    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Child thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Child thread interrupted");
        }
        System.out.println("Child thread exitied");
    }
}
public class ThreadRunnableDemo {
    public static void main(String[] args) {
        new NewThreadRunnable();
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Main thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }
        System.out.println("Main thread exitied");
    }
```

```java
class NewThread implements Runnable{
    Thread t;
    String tname;
    NewThread(String name) {
        tname = name;
        t = new Thread(this, name);
        System.out.println("New thread " + name + " created");
        t.start();
    }

    public void run() {
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println( tname + " : " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println(tname + " thread interrupted");
        }
        System.out.println(tname + " thread exitied");
    }
}
public class MultiThreadingDemo {
    public static void main(String[] args) {
        new NewThread("One");
        new NewThread("Two");
        new NewThread("Three");
        try {
            Thread.sleep(10000);
        } catch ( InterruptedException e ) {
            System.out.println("Main thread interrupted");
        }
        System.out.println("Main thread exitied");
    }
```

```
// implentation of Producer-Consumer problem

class ItemQueue {
    int item;
    boolean valueSet = false;

    synchronized int get() {
        // while value is not set wait for producer to set the value
        while(!valueSet) {
            try {
                wait();
            } catch(Exception e) {
                System.out.println("InterruptedExeption caught");
            }
        }

        System.out.println("Got: " + item);
        valueSet = false;
        notify();
        return item;
    }

    synchronized void put(int num) {
        // while value is not set wait for producer to set the value
        while(valueSet) {
            try {
                wait();
            } catch(Exception e) {
                System.out.println("InterruptedExeption caught");
            }
        }

        item = num;
        System.out.println("Put: " + item);
        valueSet = true;
        notify();
    }
}
```

```
class Producer implements Runnable {
    ItemQueue q;
    Producer(ItemQueue q1) {
        q = q1;
        new Thread(this, "Producer").start();
    }

    public void run() {
        int i = 0;
        while (i <= 5) {
            q.put(i++);
        }
    }
}

class Consumer implements Runnable {
    ItemQueue q;
    Consumer(ItemQueue q1) {
        q = q1;
        new Thread(this, "Consumer").start();
    }

    public void run() {
        int i = 0;
        while (i <= 5) {
            q.get();
            i++;
        }
    }
}

public class ProducerConsumer {
    public static void main(String[] args) {
        ItemQueue q = new ItemQueue();
        new Producer(q);
        new Consumer(q);
    }
}
```

Diff types of collections to work in a similar manner (high degree of interoperability)

Collections framework standardizes a way for what
↳ groups of objects are handled by our program.

Fundamental DS (highly efficient)

high performance

Iterator interface:

→ Iterator interface is closely related with collections framework

→ An iterator offers a general-purpose, standardized way of accessing the elements within a collection, one at a time.

→ Thus, iterator provides a means of enumerating the contents of a collection.

→ Bcz each collection implements Iterator, the elements of any collection class can be accessed through the methods defined by Iterator

→ Iterator interface provides the facility of iterating the elements in forward direction only.

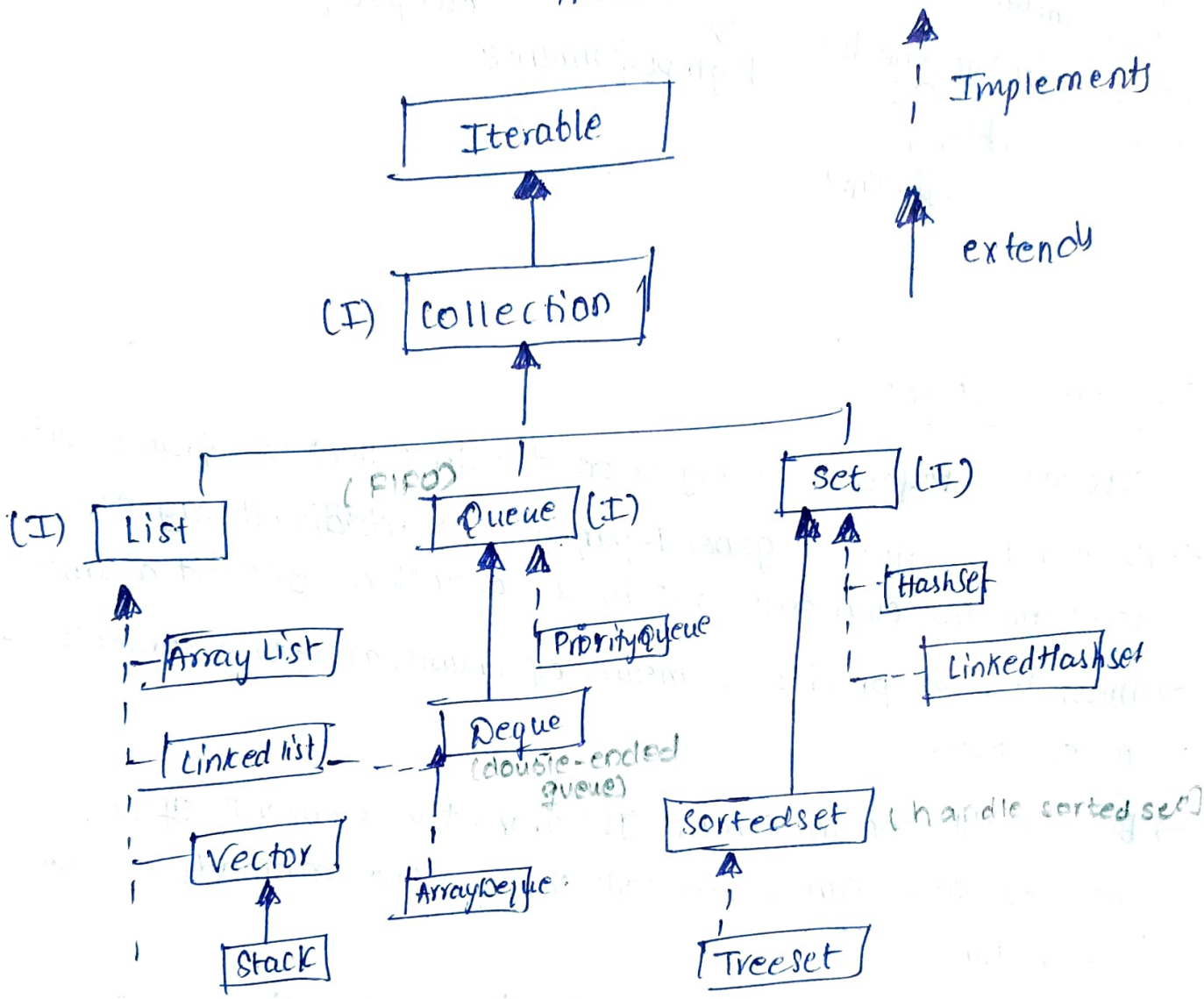
There are only 3 methods in Iterator interface. They are.

1. public boolean hasNext() ⇒ it returns true if iterator has more elements

2. public object next() ⇒ it returns the element & Moves the cursor pointer to the next element

3. public void remove() ⇒ it removes the last elements returned by the iterator. It is rarely used.

Hierarcy of collections framework



Collection Interfaces:

The collection interface:

→ foundation upon which collection framework is built.

→ declares core methods tht all collections have.

Methods:

⇒ boolean add(Object obj)

adds obj to invoked collection

Returns true if obj was added to collection

false if obj is already a member of collection or if collection does not allow duplicates

⇒ boolean addAll (Collection c)
- Adds all elements of c to the invoking collection
return true if the operation succeeded
false otherwise.

⇒ Iterator iterator() ⇒ Returns an iterator for invoking collection

⇒ boolean remove (Object obj)
removes 1 instance of obj from the invoking collection
returns true if the element is removed. false, otherwise.

⇒ boolean removeAll (Collection c)

⇒ int size() ⇒ returns no. of elements in the invoking collection

⇒ Object[] toArray()

List: (Interface)
→ extends collection
→ Elements can be inserted or accessed by their position in the list (0 based index)
→ contains duplicate elements
→ additional methods of list are below
→ several list methods throws an UnsupportedOperationException if the collection cannot be modified & a ClassCastException is generated when one object is incompatible with other

Methods:
⇒ Object get(int index)
⇒ int indexOf (Object obj)
⇒ int lastIndexOf (Object obj)
⇒ ListIterator listIterator()
⇒ ListIterator listIterator(int index)
⇒ List subList(int start, int end)

Set Interface:

→ defines a set
→ extends Collection
→ does not allow duplicate values
∴ add() method returns false if u try to add existing elem.
→ no additional methods.
→ set as a generic interface, Declaration: interface set<E>

SortedSet interface:

↳ extends Set
↳ declares the behaviour of a set sorted in ascending order
↳ Sortedset is a generic interface, Declaration: interface sortedset<E> (E: Type of obj set will hold.)
↳ It has additional methods.
↳ several methods throws NoSuchElement Exception when no items are contained in invoking set.
↳ NullPointerException, if try to add null to set bcuz it is not allowed in set
↳ first(): returns first obj in the set
last(): " last " " " "
subset(); specifying 1st & last obj in the set.
headset(): start at element and get all numbers in set from it
tailset(): Specify last element, to get all the no's in the set till the element

Navigable set Interface

↳ extends sorted set

↳ declares the behaviour of a collection that supports the retrieval of elements based on the closet match to a given value or values.

↳ generic declaration: interface NavigableSet<E>

↳ additional methods

⇒ Comparator <? super E> comparator()

⇒ E first(), E last()

⇒ SortedSet <E> headset(E end), SortedSet<E> tailset

⇒ SortedSet<E> subset(E start, E end)

⇒ E ceiling(E obj) return smallest element e >= obj ← i.e,
E floor(E obj) ↳ largest elem e <= obj if not found returns null

Iterator<E>
⇒ descending Iterator()

⇒ NavigableSet<E> descendingSet() ⇒ Returns NS that is the reverse of invoking set

⇒ E higher(obj) e>obj ⇒ E lower(obj) e<obj

⇒ E pollFirst() ⇒ E pollLast()

⇒ NavigableSet<E> subSet(E lowerBound, boolean lowIncl, E upperBound, boolean highIncl)
↳ true if include lb
↳ true if include ub.

Queue Interface

↳ Extends Collection

↳ declares behaviour of queue (FIFO)

↳ generic declaration: interface Queue<E>

Methods:

E element() ⇒ returns head of the queue (not removed)
(throws NoSuchElementException, if q is empty)

boolean offer(E obj) ⇒ attempts to add obj to queue. if added return true false otherwise

E peek() → get head (null otherwise)

E poll() → dequeue (null if queue empty)

E remove() ⇒ removes head or NoSuchElemEx

Deque Interface :

↳ extends Queue
↳ declares behaviour of double-ended queue. (FIFO & LIFO)

↳ generic & declaration:
Interface Deque <E>

→ Deque includes methods like push() & pop(), which helps it to act as Stack

→ A Deque implementation can be capacity restricted, that means limited no. of items only can be added to queue.

void addFirst(E obj)
void addLast(E obj)
Iterator<E> descendingIterator()
E getFirst()
E getLast()
boolean offerFirst(E obj)
boolean offerLast(E obj)
E peekFirst()
E peekLast()
E pollFirst()
E pollLast()
E pop()
void push(E obj)
E removeFirst()
boolean removeFirstOccurrence(Object obj)
E removeLast()

ArrayList

→ extends AbstractList implements List

→ supports dynamic arrays

→ constructors:

ArrayList()
ArrayList(Collection<? extends E> c)
ArrayList(int capacity)

LinkedList:

↳ extends AbstractSequentialList

↳ implements List, Deque, Queue.

↳ provides LL data structure.

↳ Constructors: LinkedList()
LinkedList(Collection<? extends E> c)

Hashset

↳ extends Abstractset implements the set in

→ It creates a collection that uses a hash table for storage.

→ It allows the execution time of add(), contains(), remove() & size() to remain constant

→ Constructors

Hashset()
Hashset(Collection<? extends E> c)
Hashset(int capacity)
Hashset(int capacity, float fillRatio)

↑ load factor b/w 0.0 & 1.0 (default 0.75)

→ no additional methods

→ no guarantee of the order of elements, bcz the process of hashing doesn't usually lend itself to the creation of sorted sets.

→ If u need sorted storage ⇒ TreeSet better choice

The LinkedHashset

- extends Hashset
- maintains a linked list of the entries in the set, in the order in which they are inserted
- It allows insertion-order iteration over the set.

Treeset:

- extends AbstractSet & implements the Navigableset.
- It creates a collection that uses a tree for storage.
- objects are stored in sorted, ascending order
- Access & retrevial is quite fast
- which makes Treeset an excellent choice when storing large amounts of sorted info that must be quickly found.

- Constructors:

```
Treeset()
Treeset(Collection<? extends E> c)
Treeset(Comparator<? super E> comp)
Treeset(SortedSet<E> ss)
```

PriorityQueue Class

- extends AbstractQueue
  implements Queue
- creates a queue that is prioritized based on the queue's comparator.
- dynamically growing as necessary
- 6 constructors :

PriorityQueue() — capacity → 11 (by default)
PriorityQueue(int capacity)
PriorityQueue(int capacity, Comparator<? super E> comp)
PriorityQueue(~~int capacity~~, Collection<? extends E> c)
PriorityQueue(PriorityQueue<? extends E> c)
PriorityQueue(SortedSet<? extends E> c)

- If no comparator is specified, default ascending order is taken thus the head of queue smallest val.

ArrayDeque Class

- extends AbstractCollection implements Deque
- no additional methods
- creates dynamic array (no capacity)
- Constructors :

ArrayDeque() → starting cap (by default = 16)
ArrayDeque(int size)
ArrayDeque(Collection<? extends E> c)

push()
peek()
pop()

Maps

obj that
→ stores b/w key & value pairs
→ Given a key, find its val
⇒ Both keys & vals are objs
→ keys must be unique, but values may be duplicated.
→ maps can accept null key & val.
→ they don't implement iterable interface.
→ So, no for-each

Map interfaces

Map
Map.Entry
NavigableMap
SortedMap

Map interface:

Interface Map<K, V>

```
void clear()
boolean containsKey(Object k)
boolean containsValue(Object v)
Set<Map.Entry<K,V>> entrySet()   → set view of invoking map
boolean equals(Object obj)
V get(Object k)
int hashCode()   → hash code of invoking map
boolean isEmpty()
Set<K> keySet()
V put(K k, V v)
void putAll(Map<? extends K, ? extends V> m)
V remove(Object k)
int size
Collection<V> values()
```

The SortedMap Interface

↳ extends Map

↳ ensures ascending order based on keys.

K firstKey()

K lastKey()

SortedMap<K,V> headMap(K end)

SortedMap<K,V> subMap(K start, K end)

SortedMap<K,V> tailMap(K start)

NavigableMap

↳ extends SortedMap

↳ declares the behaviour of a map that supports the retrieval of entries based on the closest match to a given key or keys.

Map.Entry<K,V> ceilingEntry(K obj)

K ceilingKey(K obj)

NavigableSet<K> descendingKeySet()

" Map<K,V> " Map()

Map.Entry<K,V> firstEntry()

" floorEntry(K obj)

K floorKey(K obj)

NavigableMap<K,V> headMap(K upperBound, boolean incl)

higherKey()

lowerKey()

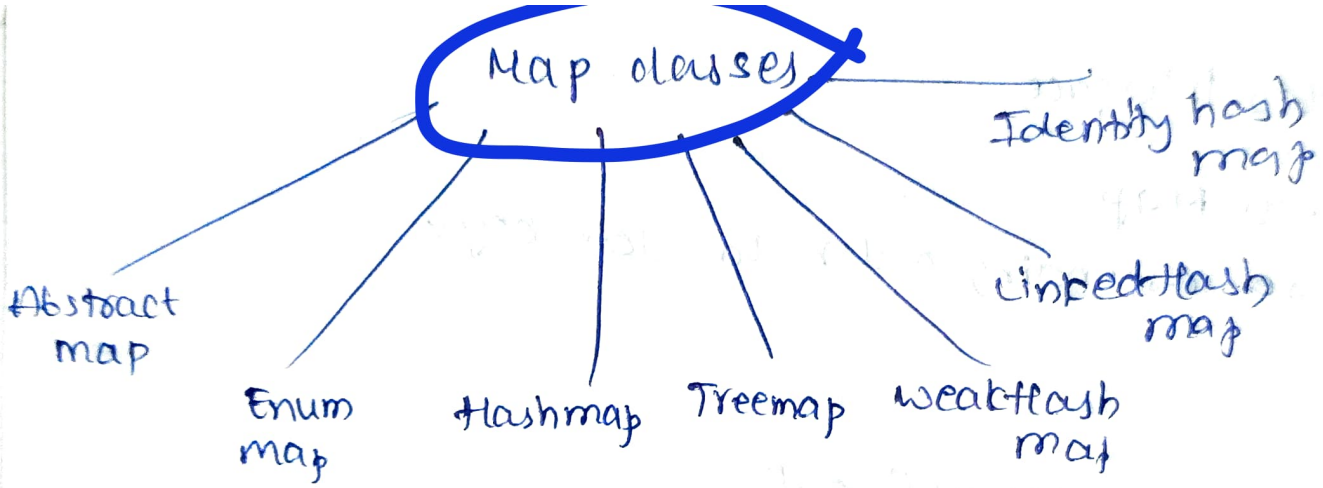
pollFirstEntry()

pollLastEntry()

subMap(lower, lowIncl, upper, upperIncl)

tailMap(lb, incl)

navigableKeySet()


Map classes
Identity hash map
Linked Hash map
Weak Hash map
Tree map
Hash map
Enum map
Abstract map

```java
import java.io.*;

public class DisplayFile {
    Run | Debug
    public static void main(String[] args) throws IOException {
        FileInputStream fi;
        try {
            fi = new FileInputStream(args[0]);
        } catch (FileNotFoundException e) {
            System.out.println("File not found");
            return;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Usage: DisplayFile File");
            return;
        }

        int i;
        do {
            i = fi.read();
            if (i != -1) {
                System.out.print((char)i);
            }
        } while( i != -1);

        fi.close();
    }
}
```

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyFile {
    Run | Debug
    public static void main(String[] args) {
        int i;
        FileInputStream fi;
        FileOutputStream fo;

        try {
            // open input file
            try {
                fi = new FileInputStream(args[0]);
            } catch (FileNotFoundException e) {
                System.out.println("File not found");
                return;
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Usage: CopyFile sourceFile destFile");
                return;
            }

            // open output file
            try {
                fo = new FileOutputStream(args[1]);
            } catch (FileNotFoundException e) {
                System.out.println("Error in opening output file");
                return;
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Usage: CopyFile sourceFile destFile");
                return;
            }

            // copy the file
            try {
                do {
                    i = fi.read();
                    if (i != -1) {
                        fo.write(i);
                    }
                } while( i != -1);

            fi.close();
            fo.close();

            } catch(IOException e) {
                System.out.println("File Error");
            }
        } catch (Exception e) {
            System.out.println("Error");
        }

    }
```

```java
// Java program to demonstrate Iterator
import java.util.ArrayList;
import java.util.Iterator;

public class IterDemo
{
    Run | Debug
    public static void main(String[] args)
    {
        ArrayList<Integer> al = new ArrayList<Integer>();

        for (int i = 0; i < 10; i++)
            al.add(i);

        System.out.println(al);

        // at beginning itr(cursor) will point to
        // index just before the first element in al
        Iterator itr = al.iterator();

        // checking the next element availabilty
        while (itr.hasNext())
        {
            //  moving cursor to next element
            int i = (Integer)itr.next();

            // getting even elements one by one
            System.out.print(i + " ");

            // Removing odd elements
            if (i % 2 != 0)
                itr.remove();
        }
        System.out.println();
        System.out.println(al);
    }
}
```

Comparator (I) :-

→ Comparator Interface present in java.util package & defines the following 2 methods.

① public int compare(Object obj1, Object obj2):

- → returns -ve iff obj1 has to come before obj2
- → returns +ve iff obj1 has to come after obj2
- → returns 0 iff obj1 & obj2 are equal (duplicate).

obj1 ⇒ which object we are trying to add
obj2 ⇒ already existing object

② public boolean equals(Object obj)

→ When ever we are implementing Comparator Interface Compulsary we should provide implementation for compare(), 2nd method .equals() implentation is optional, because it is already available for our class from Object class through Inheritance.

public abstract class **MouseAdapter**
extends Object
implements MouseListener

An abstract adapter class for receiving mouse events. The methods in this class are empty. This class exists as convenience for creating listener objects.

Mouse events let you track when a mouse is pressed, released, clicked, when it enters a component, and when it exits. (To track mouse moves and mouse drags, use the MouseMotionAdapter.)

Extend this class to create a `MouseEvent` listener and override the methods for the events of interest. (If you implement the `MouseListener` interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you can only have to define methods for events you care about.)

Create a listener object using the extended class and then register it with a component using the component's `addMouseListener` method. When a mouse button is pressed, released, or clicked (pressed and released), or when the mouse cursor enters or exits the component, the relevant method in the listener object is invoked and the `MouseEvent` is passed to it.

See Also:
`MouseEvent`, `MouseListener`, Tutorial: Writing a Mouse Listener, Reference: The Java Class Libraries (update file)

Constructor Summary

MouseAdapter()

Method Summary

void	**mouseClicked**(MouseEvent e) Invoked when the mouse has been clicked on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

```
// A Java program for a Serverside
import java.net.*;
import java.io.*;

public class ServerSide {
    // initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;

    // constructor with port
public Server(int port)
{
// starts server and waits for a connection
try{
server = new ServerSocket(port);
System.out.println("Server started");
System.out.println("Waiting for a client ...");
socket = server.accept();
System.out.println("Client accepted");
// takes input from the client socket
in = new DataInputStream(
new BufferedInputStream(socket.getInputStream()));
String line = "";
// reads message from client until "Over" is sent
while (!line.equals("Over"))
{
try
{
line = in.readUTF();
System.out.println(line);



}
catch(IOException i)
{
System.out.println(i);
}
}
System.out.println("Closing connection");
// close connection
socket.close();
in.close();
}
catch(IOException i){
System.out.println(i);
}
}

    public static void main(String args[]) {
        Server server = new Server(5000);
    }
}
```

```
// A Java program for a ClientSide
import java.net.*;
import java.io.*;
public class ClientProgram {
    // initialize socket and input output streams
    private Socket socket = null;
    private DataInputStream input = null;
    private DataOutputStream out = null;

    // constructor to put ip address and port
public Client(String address, int port)
{
// establish a connection
try
{
socket = new Socket(address, port);
System.out.println("Connected");
// takes input from terminal
input = new DataInputStream(System.in);
// sends output to the socket
out = new DataOutputStream(socket.getOutputStream());
}
catch(UnknownHostException u)
{
System.out.println(u);
}
catch(IOException i)
{
System.out.println(i);
}// string to read message from input
String line = "";
// keep reading until "Over" is input
while (!line.equals("Over"))
{
try
{
line = input.readLine();
out.writeUTF(line);
}
catch(IOException i)
{
System.out.println(i);
}
}
// close the connection
try
{
input.close();
out.close();
socket.close();
}
catch(IOException i)
{
System.out.println(i);
}
}

    public static void main(String args[]) {
        Client client = new Client("127.0.0.1", 5000);
    }
}
```

S.NO	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.