HashMap, HashSet- 2

Strings-3

Interfaces-4

Comparator-5

Exception-6

Arraylist, set, List-7

Collections framework and packages-8

File Handling-9

Bit -10

Natural lang-11

Oops-12

Miscellaneous-At last

**HashMap and HashSet**

HashMap<I,I>- contiansKey, contiansValue, isEmpty, keySet- gives Set<>, size, put, get

Split-method-String Method

HashSet-add, remove,contains

Iterator<String> i = h.iterator();

Where **h** is the name of HashSet or can be set.

And i.hasNext() and i.next

Map.entrySet=return hashmap in list form

Map.Entry-interface for hashmap provides certain function equals <https://www.geeksforgeeks.org/map-entry-interface-java-example/>

Sorting in hashmap using above tech and comparator

Load Factor- filled/total. Rehashing inc array size(1.5 or 2) and again hashing all values

HashSet uses **HashMap** for storing its object internally. Value is constant variable.

Strings

String(char array) return string

**Integer.toString**- to concat int with String.

Integer.parseInt()-fails for -ve value.

Interfaces

All its functions are abstract, no need to write abstract, provides what to do, not how to do

If a class extend an interface, it is just like a contract that class must implement all its function

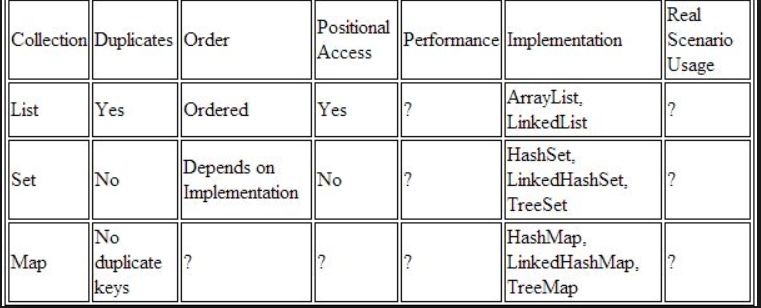
**Set** is implemented by HashSet, LinkedHashSet or TreeSet and extends collection

List- also by vector and stack- <https://www.geeksforgeeks.org/list-interface-java-examples/>

Set-<https://www.geeksforgeeks.org/set-in-java/> .All set operation like union, sub etc. **not sorted**.

SortedSet-Interface, extends Set <https://www.geeksforgeeks.org/sortedset-java-examples/>

List vs Set both extends collection interface



Class- blueprint for creating objects

Comparator

Use to sort objects

Implements comparator generic interface.

compare function and equals(one argument) function, public methods

-1 when a<b

1 when a>b sort happens

0 a==b

Then Collections.sort(collection, comparator object)/Arrays.sort()

**Comparable**-compareTo()-we can use only once, so we can only do for a particular attribute/members of class. Overriding **interface** function. Implemented by item class.

[comparable vs comparator](https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/)

# Exception

Util package

And for creation an expection one must extend **Exception** class

ArrayList

Subtype of list

<https://www.geeksforgeeks.org/arraylist-in-java/>

add(it is used to add to specific location), by shifting all by one location

List vs ArrayList-interface and class

Interface implementation support change

Vector vs ArrayList

All the methods of Vector is synchronized(only one tread), 2x. But, the methods of ArrayList is not synchronized(multi treading). 1.5x.

[syn vs asyn](https://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-does-it-really-mean)

**Collections Framework and packages**

Lang package is by default imported- methods are substring or other string function, math class, System class wrapper class [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/)

Utility class- Class used for dealing with objects ie storing and manipulating data having many structures

Collections of classes or data structures.

Collection-interface, A Collection represents a single unit of objects, i.e., a group

Collections-utility class.<https://www.javatpoint.com/collections-in-java>

File Handling

6 ways using

BufferReader, Scanner, fileReader

And for deleting of file- delete()- return true or false.

File obj with path as parameter.

BufferReader- character stream, buffer of character.

Scanner-is a simple text scanner which can parse primitive types and strings. It internally uses regular expressions to read different types. Ignore new line character, nextLine read upto new line only.

BufferedReader is synchronous while Scanner is not. BufferedReader should be used if we are working with multiple threads.

BufferedReader has significantly larger buffer memory than Scanner.

The Scanner has a little buffer (1KB char buffer) as opposed to the BufferedReader (8KB byte buffer), but it’s more than enough.

BufferedReader is a bit faster as compared to scanner because scanner does parsing of input data and BufferedReader simply reads sequence of characters.

**Bit masking**

Or and xor

A^0=A, A^1=~A

a, a-1 used

where is 1st 1 in a, there is 0 in a-1

(a<<(c+d)) + e this need to be done.

To get 1st set bit **a&(~a-1) by** and in xor 1st “1” indicate place where bits are diff in two no. equivalent to a&(~a-1) <=>a^(a-1)&a

**Natural Language**

[**https://www.youtube.com/watch?v=sa-TUpSx1JA&t=2047s**](https://www.youtube.com/watch?v=sa-TUpSx1JA&t=2047s)

some important terms

. - Any Character Except New Line

\d - Digit (0-9)

\D - Not a Digit (0-9)

\w - Word Character (a-z, A-Z, 0-9, \_)

\W - Not a Word Character

\s - Whitespace (space, tab, newline)

\S - Not Whitespace (space, tab, newline)

\b - Word Boundary

\B - Not a Word Boundary

^ - Beginning of a String

$ - End of a String

[] - Matches Characters in brackets

[^ ] - Matches Characters NOT in brackets

| - Either Or

( ) - Group

Quantifiers:

\* - 0 or More

+ - 1 or More

? - 0 or One

{3} - Exact Number

{3,4} - Range of Numbers (Minimum, Maximum)

#### Sample Regexs ####

[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

.

Oops

Primitive value not accept null value;

A **final class** is simply a **class** that can't be extended.

Wrapper class Integer is a final class. If that class is not final, then any one can extend Integer into his own class and change the basic behaviour of integer class. To avoid this, java made all wrapper classes as final classes. Derived final class (can extend non final) or function can be final.

**Method final-** nooverriding by derived(error), fixed functionality, Final methods are faster than instance methods, as there is no use of virtual table concept for final and private methods. Derived function can be final

**constructor** can't be final. They can be protected but **class** can't

The **protected** specifier allows access by all subclasses of the class in a program, whatever package they reside in, as well as to other code in the same package. The **default** specifier allows access by other code in the same package, but not by code that is in subclasses residing in different packages.

**Static vs non static(instance)**

Instance method are methods which require an object of its class to be created before it can be called. Static methods are the methods in Java that can be called without creating an object of class. Static method is declared with static keyword. Instance method is not with static keyword. Static only one copy for class. There is no cross usage ie instance can’t use static or vice versa. No opposite overriding.

But non static method can access static variable but not vice versa

Instance-dynamic binding-**overriding**, overloading

Static- static binding-**overloading**, no overriding( if both static then hiding and leads to calling class function to which we refer on left hand side).

Static variables are properties of the class. They do not belong to objects.

null.static\_funtion works but null.non\_static\_function doesn’t. [**more on null**](https://www.geeksforgeeks.org/interesting-facts-about-null-in-java/)

**Static-** [**https://www.geeksforgeeks.org/static-keyword-java/**](https://www.geeksforgeeks.org/static-keyword-java/)

Static block- This code inside static block is executed only once: the first time you make an object of that class or the first time you access a static member of that class (even if you never make an object of that class). It executed before non-static, constructor [static block](https://www.geeksforgeeks.org/g-fact-79/) static{}

Initializer block- no name only {}, as many time obj is made or constructor is invoked.

“This” and “super” (ref variable) keyword can’t be used in static methods, this is constant pointer

Super- parent class obj.

**Polymorphism**

allows us to perform a single action in different ways for ex sound of animal

It describes the concept that different classes can be **used** with the same interface

Virtual-overridden

This is known as [Polymorphism](http://java67.blogspot.com/2012/10/difference-between-polymorphism-overloading-overriding-java.html) because any virtual method will be executed from subclass only, even though they were called from super type.

All non-static methods are virtual except final ones. Overridden methods are virtual methods.

Base wala override ho jata hai. (runtime polymorphism)

**Dynamic Method Dispatch**-Polymorphism overriding ie at runtime polymorphism. (is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time).

**Ad-hoc** polymorphism is also known as overloading.

If virtual method in derived class(subclass)is private, it will raise an error. Or access modifier of derived is equal or more than of base.

By default virtual.

When data type not found, it will search for higher data type(no long then go for float and so on or no char then go for int)

**Inheritance in Java**

is a mechanism in which one object acquires all the properties and behaviours of a parent object, hierarchical- one base many derived. No multiple inheritance

**Composition-** They are absolutely different. Inheritance is an ***"is-a"*** relationship. Composition is a ***"has-a"***. Package or by creating obj instead of extend. class object to be used inside another class. Car has wheel(composition) and student is human(inheritance).

**Abstraction**: Abstraction is the act of representing essential information without including background details and explanations. **Data Hiding-** hiding internal information.

**Encapsulation**: Encapsulation is the act of **wrapping** up of attributes (represented by data members) and operations (represented by functions) under one single unit (represented by class).

Ex-Vending machine or tic tac toe-

Abstraction-we are only concerned with i/o not internal processing

Encapsulation-Vending machine is a class- data- items and functions take input/ deliver/ make payment etc They are wrapped.

Data Hiding- how many cans we have/ or how many variables

**Encapsulation is a way to implement Abstraction**

**Abstract class**

When we want every sub class must implement/make a particular function as we don’t know its ans right now. For ex vehicle class, car, bmw

We can’t create its object ie new abstract\_class\_name(). Not allowed

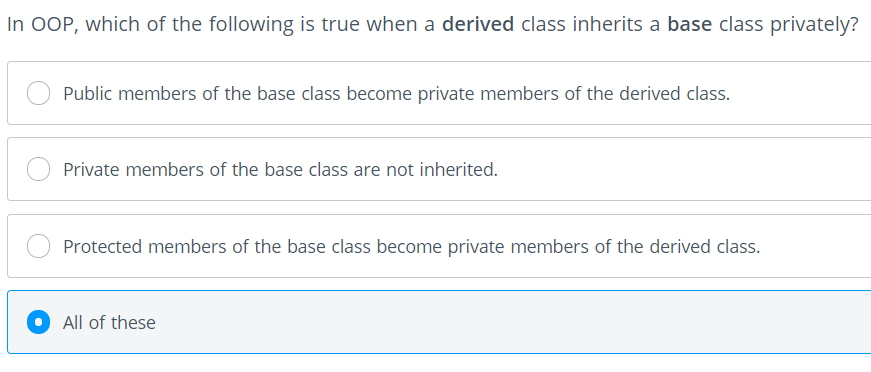
Sub class has two options- make is abstract or complete abstract function.

Abstract class cannot have abstract constructors, Abstract class cannot have abstract static methods (static method can’t be overridden, compile time polymorphism)

**C++**

Constructor- can be private, protected, copy constructor but not friend

**Friend Function-** not in java as it will break the design. It is not the member function but can access private variable. Can be called using objs.



**Miscellaneous**

Java heap size-<http://net-informations.com/java/cjava/limit.htm>

Sockets three types-

Connection-oriented- TCP(Socket class) eg-**date server**, connectionless- udp(DatagramSocket class), (MulticastSocket class)- multiple recipients ie broadcast

127.0.0.1- loopback

Inline functions-no call overhead