1. JVM basic

1.1 class loader

a. Loading :

The Class loader reads the *.class* file, generate the corresponding binary data and save it in method area.

After loading *.class* file, JVM creates an object of type Class to represent this file in the heap memoryThis Class object can be used by the programmer (use *getClass()* method of Object class) for getting class level information like name of class, parent name, methods and variable information etc.

b. Linking

Performs verification, preparation, and resolution.

Verification : It ensures the correctness of .class file i.e. it check whether this file is properly formatted and generated by valid compiler or not. If verification fails, we get run-time exception java.lang.VerifyError.

Preparation : JVM allocates memory for class variables and initializing the memory to default values.
Resolution : It is the process of replacing symbolic references from the type with direct references. It is done by searching into method area to locate the referenced entity.

c. Initialization

In this phase, all static variables are assigned with their values defined in the code and static block(if any). This is executed from top to bottom in a class and from parent to child in class hierarchy.

1.2 JVM Memory

a. **Method area :** In method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
b. Heap area : Information of all objects is stored in heap area. There is also one Heap Area per JVM. It is also a shared resource.
c. Stack area : For every thread, JVM create one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which store methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminate, it's run-time stack will be destroyed by JVM. It is not a shared resource.
d. PC Registers : Store address of current execution instruction of a thread. Obviously each thread has separate PC Registers.
d. Native method stacks : For every thread, separate native stack is created. It stores native method information.

1.3 Execution Engine

Execution engine execute the *.class* (bytecode). It reads the byte-code line by line, use data and information present in various memory area and execute instructions. It can be classified in three parts

2. Key Words and Statement in Java

a. Static

The static key word works similarly in Java as it is in c++, however, below are the difference.

1) **Static Block:** Unlike C++, Java supports a special block, called static block which can be used for static initialization of a class. This code inside static block is executed only once. It is executed when the first time you make an object of that class or the first time you access a static member of that class.
2) **Static Local Variables:** Unlike C++, Java doesn't support static local variables. **Static class:** See inner class

b. **Enum** in Java compared to c++

1) Enums in C/C++ are plain constant Integers.
2) Enums in Java are objects - they can have methods (with different behavior from one enum instance to the other).
3) Every enum internally implemented by using Class.

```
Class Color
{
        Public static final Color RED = Color();
        Public static final Color GREEN = Color();
}
Color c1 = Color.RED;
```

4) enum can contain constructor and it is executed separately for each enum values. All the values are static so the constructor is called at the time of enum class loading. The constructor is private so user can not create objects on their own. In the Color example, there are only two Color objects existing which are red and green, no other enum objects are created.
5) enum type can be passed as an argument to switch statement.
6) values(), ordinal() and valueOf() methods :
   These methods are present inside java.lang.Enum.
   values() method can be used to return all values present inside enum.

ordinal() method return the index according to the order it is defined.

7) Enum can have abstract method and each of the values can override it.

c. Final

Similar to const in c++, the difference is

1) It can be initialized after declaration
2) Final key word in front of class member function mean no overriding.Final can not put after the function name as it is in c++
3) Final key word in front of class definition means no extend(inheritance)

d. Package

Package is a mechanism to encapsulate a group of classes, sub packages and interfaces. The files with package keyword followed by the same package name belong to the same package.

e. Modifier: public private protected

	Same Class	Package	Derived class same package	Derived class different package	world
Default(package)	Yes	Yes	Yes	No	No
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Private	Yes	No	No	No	No

f. Abstract vs interface

Abstract	Interface
Can have abstract and non-abstract methods.	Can only have abstract methods.
Does not support multiple inheritance.	Supports multiple inheritance.
Can have non-final, non-static variable	Only have static and final variable.
Can provide the implementation	

of interface.	

3. Garbage Collector

a. To make an object eligible for garbage collector

3.1 Object created inside a method and is not return as a reference to the object

3.2 Reassigning the reference variable.

3.3 Nullifying the reference variable.

3.4 Anonymous object : The reference id of an anonymous object is not stored anywhere. Hence, it becomes unreachable.

b. Request JVM to run garbage collector

When JVM runs Garbage Collector, we can not expect. So we can also request JVM to run Garbage Collector. There are two ways to do it :

3.4.1 Using *System.gc()* method :

3.4.2 Using *Runtime.getRuntime().gc()* method :

c. Mark and sweep:

1) All the objects have their marked bits set to false.
2) Traverse all the objects starting with root object, then mark all the objects that have been reached to true.
3) Non reachable objects are cleared from the heap.
   Disadvantage:
   Normal program execution is suspended while the garbage collection algorithm runs.

4. Initialization, Copy, Assignment

4.1 Initialization

a. Constructor in Java compared to C++

1) Java constructor is only called by new operator. While in c++ constructor is automatically inserted by the compiler when a variable is defined.
2) There is no initialization list in java constructor.

b. Inline Initialization

1) Java can initialize a member variable using inline field initialization
2) Static fields can initialized in static block or inline filed initialization.

c. **Initialization Block**

Initialization Block is needed when

1) We have many constructor with different parameters then we can put the common pieces that every constructor has into the initialization block so we avoid writing the same code for every constructor.
2) Anonymous class since there is no explicit object.

d. Order of initialization

1) static initialization blocks of super classes
2) static initialization blocks of the class
3) instance initialization blocks of super classes
4) constructors of super classes
5) instance initialization blocks of the class
6) constructor of the class.

5. Override:

a. Definition and Usage

All non-static method in java are by default “virtual”(as in C++), which allows us to override. There are 3 types of functions that are not allowed to be overridden.

1) Functions with final cannot be overridden. This is what final means.

2) Functions with private keyword cannot be overridden as the derived class has no access to it.

3) Functions with static keyword cannot be overridden as it belongs to the class itself and does not involve object instance. A static method in the derived class that overrides the method in base class hides the method in base class.

4) Override must have the same argument list and return type.

5) The overriding method in derived class has to have access modifier with more access that the method in the base class

6) If the super-class overridden method does not throws an exception, subclass overriding method can only throws the unchecked exception, throwing checked exception will lead to compile-time error. If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception.

b. Frequently Used Override functions

1) Equals
2) toString

6. Inner Class

a. Definition and usage

Inner class is used when this class is solely needed by the outsider class. In other words, the outsider class owns the inner class.

b. Static inner class

What static means for inner class is similar to static field and method. Static inner class belongs to the whole outsider class, and we do not need to have an instance of outsider class to access the inner class object.

c. Scope

The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class *NestedClass* does not exist independently of class *OuterClass*.

A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.

7. Array

a. Key difference in array between c++ and Java

1) Java array is aware of its size while a C++ array is not. Every Java array has a field named length.

2) A Java array prevents a programmer from indexing the array out of bounds while a C++ does not. The range checking comes at the prices of having a small amount of overhead.

3) In C++ it is possible to create an array of fully constructed objects in a single operation while it is not possible to do this in Java.
   Example:
   Employee[] emp = new Employee[5]

//comment: emp is a reference which points to an array of reference of Employee. The array has a size of 5 and the reference of Employee is initialized to null.

```
for(int i=0; i< emp.length; i++)
{
        emp[i] = new Employee();
 }
```

// comment Each object is created or instantiated one at a time with the "new" operator and the address of each object is stored in the array

b. Frequently used functions

(1) sort

Arrays.sort(T[] a, Comparator<? Super T> c)

The parameter are an array of primitives or objects, and a comparator

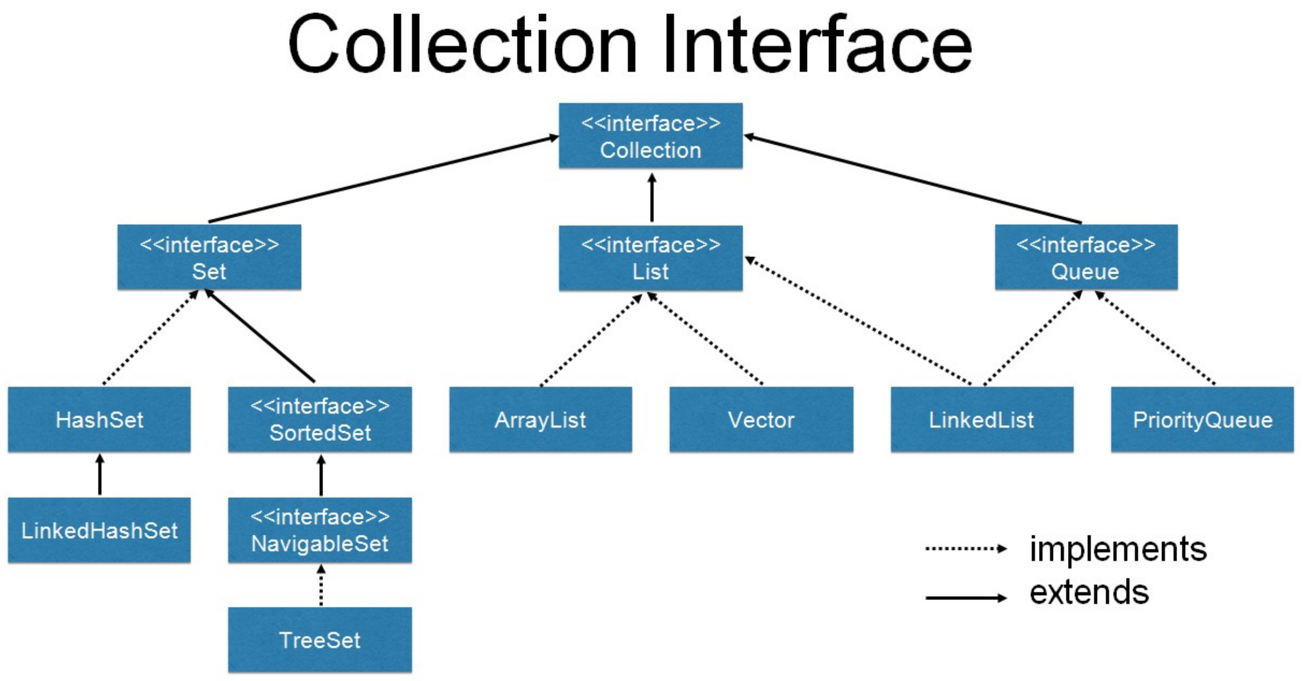
(2) toString()

Returns a string representation of the contents of the specified array. The string representation consists of a list of the array’s elements, enclosed in square brackets (“[]”). Adjacent elements are separated by the characters “, ” (a comma followed by a space). Returns “null” if a is null.

(3) List.toArray()

ArrayList and LinkedList(Both implement List) have toArray() method to convert List to java Array

(4)Arrays.asList()
Convert primitive array to list.



8. Java Data Structure

8.1 Collection Framework

- **Interface** java.lang.Iterable
  - **Interface** Collection
    - **Interface** List
      - **Class** ArrayList
      - **Class** LinkedList (also implements Deque)
      - **Class** Vector
        - **Class** Stack (legacy class, use Deque, which is more powerful)
    - Interface Set
      - Class HashSet
        - Class LinkedHashSet
      - Interface SortedSet
        - Interface NavigableSet
          - Class TreeSet

            - Class EnumSet
        - Interface Queue
            - Class PriorityQueue
            - Interface Deque
                - Class LinkedList (also implements List)
                - Class ArrayDeque
            - Interface BlockingQueue
                - Class ArrayBlockingQueue
                - Class PriorityBlockingQueue
                - Interface BlockingDeque
                    - Class LinkedBlockingDeque
- Interface Map
    - Class HashMap
    - Interface SortedMap
        - Interface NavigableMap
            - Class TreeMap

8.2 ArrayList, Linkedlist, Vector, ArrayDeque

a. Function summary

	ArrayList	LinkedList	ArrayDeque
size(), isEmpty()	Y	Y	Y
contains()	Y	Y	Y
Iterator(), .hasNext(),.next()	Y	Y	Y
add()	Y	Y	Y
get(), set()	Y	Y	N
getFirst(),getLast()	N	Y	Y
offerFirst(), offerLast()	N	Y	Y
pollFirst(), pollLast()	N	Y	Y
peekFirst(), peekLast()	N	Y	Y
removeFirst(), removeLast(),	N	Y	Y

b. ArrayList vs. LinkedList

	ArrayList	LinkedList
Interface	Java List	Java List and Queue
Underlying data structure	Resizable Array	Doubly linked list
Fast insertion	No O(n)	Yes O(1)
Fast access	Yes O(1)	No O(n)

c. ArrayList vs. Vector

	ArrayList	Vector	ArrayDeque
Synchronized	No	Yes	No
Increment when full	50% of current size	100% of current size	100% of current size
Fast in operation	Yes	No, because it is synchronized in multithreading	

d. ArrayList vs LinkedList vs ArrayDeque

The difference of these three is similar to vector, list, and deque in c++

8.3 Set(HashSet, LinkedHashSet, TreeSet, EnumSet) vs Map(HashMap, LinkedHashMap, TreeMap, EnumMap,)

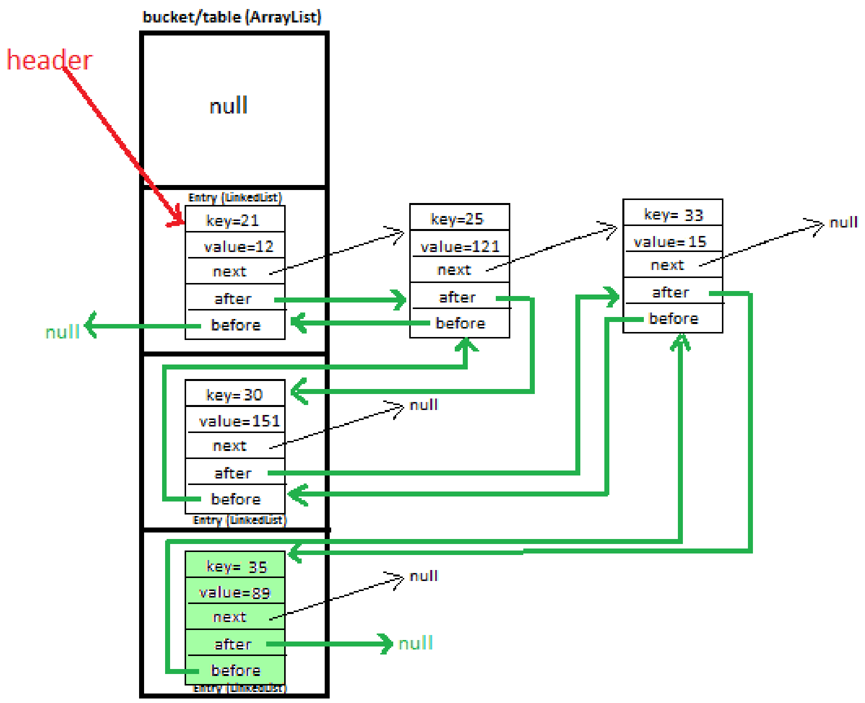
a. Functions

	Set	Map
Size(), isEmpty()	Y	Y
iterator	Y	N
hashCode()	Y	Y
EntrySet(), keySet(), value(),	N	Y

Contains()	Y	N
containsKey(), containsValue(), get()	N	Y
add()	Y	N
Put()	N	Y
Remove()	Y	Y
Replace()	N	Y

b. LinkedHashSet vs LinkedHashMap

Internally has a before and after reference to track the insertion order



c. Difference between HashSet/Map, LinkedHashSet/Map, TreeSet/Map and HashTable

	HashSet/Map	HashTable	LinkedHashSet/ Map	TreeSet/ Map
Interface	Set/Map	Set/Map	Map	Set-> SortedSet Map-> SortedMap

Underlying Data Structure	Array of chains of key or object <key, value> while the key and array index are linked by Hashfunction	Array of chains of object <key, value> while the key and array index are linked by Hashfunction	Array of chains of objects(like HashSet/Map and HashTable) plus a doubly linkedlist store before and after objects	Red-Black binary tree of keys or object<key, value>
Way to compare	Need to override equal() and hashCode()	Need to override equal() and hashCode()	Need to override equal() and hashCode()	Need to have compare()
Lookup insertion time complexity	O(1)	O(1)	O(1)	O(log(n))
Allowing Null Key	Yes	No	Yes	No, but allows null values
Unique key only	Yes	Yes	Yes	Yes
Iteration Oder	Random	Random	Insertion Order	RB Tree
Synchronized	No	Yes	No	No

d. Usage guideline

1) Suppose you were creating a mapping of names to Person objects. You might want to periodically output the people in alphabetical order by name. A TreeMap lets you do this.
2) A TreeMap also offers a way to, given a name, output the next 10 people. This could be useful for a "More"function in many applications.
3) A LinkedHashMap is useful whenever you need the ordering of keys to match the ordering of insertion. This might be useful in a caching situation, when you want to delete the oldest item.
4) Generally, unless there is a reason not to, you would use HashMap. That is, if you need to get the keys back in insertion order, then use LinkedHashMap. If you need to get the keys back in their true/natural order, then use TreeMap. Otherwise, HashMap is probably best. It is typically faster and requires less overhead.

9. Comparable vs Comparator

	Comparable	**Comparator**
Sorting logic	Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects	Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc.
Implementation	Class whose objects to be sorted must implement this interface.	Class whose objects to be sorted do not need to implement this interface. This class objects can be sorted by a third class which implements Comparator of this class.*(see below)
Sorting method	int compareTo(Object o1) 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1	int compare(Object o1,Object o2) Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1
Calling method	Collections.sort(List) Here objects will be sorted on the basis of CompareTo method	Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator

```
public class CountrySortByIdComparator implements Comparator<Country>{
  @Override
 public int compare(Country country1, Country country2) {
   return (country1.getCountryId() < country2.getCountryId() ) ? -1:
 (country1.getCountryId() > country2.getCountryId() ) ? 1:0 ;
     }
```

}

9.1 Priorityqueue

Priorityqueue in java uses priority heap data structure.

a. Constructor with Comparator
   PriorityQueue(int initialCapacity, Comparator<? super E> comparator)
b. Functions
   contains(Object o)
   clear()
   Iterator<E>
   Offer(E e)
   Peek() //retrieve not remove
   Poll() // retrieve and remove
c. Complexity
   Offer and Poll both log(n)
d. The data in the priority queue is not sorted, but only keeps the heap structure.

10. String

10.1 String and StringBuilder

a. Mutability
   A String is immutable is java, while StringBuilder is mutable in Java. When we concatenate two strings in Java, a new string object is created in the string pool.
b. Equality
   We can use equals() method in String for comparing two string in java, but we can not use equals() method in StringBuilder as StringBuilder does not override the equals() method.
c. Comparable
   String class implements the Comparable interface, while StringBuilder does not.
d. Constructor
   We can create a String object without using new operator, which is not possible with a StringBuilder.

String str = “abc” is equivalent to
Char data[] = {‘a’, ‘b’, ‘c’};
String str = new String(data);

e. Performance
   StringBuilder is very fast than String while performing concatenations. Because String is immutable in Java and concatenation of a two String objects involves creation of a new object.
f. Length
   Since String is immutable, its length is fixed. But StringBuilder has setLength() method which can be used to change the StringBuilder object to the specified length.

11. OOP

11.1 Inheritance

1) In C++, we can create a class that doesn’t inherit from anything. In Java, all classes inherit from the Object class directly or indirectly.
2) In Java, members of the grandparent class are not directly accessible. So we can not call super.super.xxx()
3) In Java, protected members of a class “A” are accessible in other class “B” of same package, even if B doesn’t inherit from A.
4) Java uses *extends* keyword for inheritance. Unlike C++, Java doesn’t provide an inheritance specifier like public, protected or private. Like C++, private members of base class are not accessible in derived class.
5) In Java, methods are virtual by default. In C++, we explicitly use virtual keyword
6) Unlike C++, Java doesn’t support multiple inheritance.
7) In C++, default constructor of parent class is automatically called, but if we want to call parametrized constructor of a parent class, we must use Initializer list. Like C++, default constructor of the parent class is

automatically called in Java, but if we want to call parametrized constructor then we must use super to call the parent constructor.

12. Design Pattern

12.1 Singleton Pattern

a. Def

This pattern is needed when we would like to have only ONE instance of the class. This is done by

(1) Make the constructor private

(2) Have a static instance of the class itselfIt is usually used in the logger.

b. Usage and Examples

(1) Logger class that writes the log file

(2) java.lang.Runtime

12.2 Factory Pattern

a. Def

b. Example

```
abstract class Product
{
        public abstract Product createProduct();
        ...
}

class OneProduct extends Product
{
        ...
        static
        {
                ProductFactory.instance().registerProduct("ID1", new
OneProduct());
        }
        public OneProduct createProduct()
        {
                return new OneProduct();
        }
        ...
}

class ProductFactory
{
        private HashMap m_RegisteredProducts = new HashMap();
        public void registerProduct(String productID, Product p)    {
                m_RegisteredProducts.put(productID, p);
        }

        public Product createProduct(String productID){

        ((Product)m_RegisteredProducts.get(productID)).createProduct();
        }
}
```

12.3 Observer Pattern

a. Def

   If Class B needs to change something as long as A has some changes, we can name a field in class A called “Obserers”, and put B as an observer of A. Then A has a notify method such that when A has some change, notify method is called to ask B to run the update function to get the corresponding update.

b. Examples

```
class Class A
{
    ArrayList<Observer> observerList;
    public A() {
        observerList = new ArrayList<Observer>();
    }

    @Override
    public void registerObserver(Observer o) {
        observerList.add(o);
    }

    @Override
    public void unregisterObserver(Observer o) {
        observerList.remove(observerList.indexOf(o));
    }

    @Override
    public void notifyObservers()
    {
        for (Iterator<Observer> it =
              observerList.iterator(); it.hasNext();)
        {
            Observer o = it.next();
            o.update();
        }
    }
```