**1.What are the principle concepts of OOPS?**

There are four principle concepts upon which object oriented design and programming rest. They are:

* Abstraction
* Polymorphism
* Inheritance
* Encapsulation

**2.What is Abstraction?**

Abstraction refers to the act of representing essential features without including the background details or explanations.

**3.What is Encapsulation?**

Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

**4.What is the difference between abstraction and encapsulation?**

* **Abstraction** focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it’s inside view, where the behavior of the abstraction is implemented.
* **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
* **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up wer abstraction to suit the developer needs.

**5.What is Inheritance?**

* Inheritance is the process by which objects of one class acquire the properties of objects of another class.
* A class that is inherited is called a superclass.
* The class that does the inheriting is called a subclass.
* Inheritance is done by using the keyword extends.
* The two most common reasons to use inheritance are:
  + To promote code reuse
  + To use polymorphism

**6.What is Polymorphism?**

Polymorphism is briefly described as “one interface, many implementations.” Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

**7.How does Java implement polymorphism?**

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).  
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

* In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
* In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

**8.Explain the different forms of Polymorphism.**

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is done using inheritance and interface.  
**Note**: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

* *Method overloading*
* *Method overriding through inheritance*
* *Method overriding through the Java interface*

**9.What is runtime polymorphism or dynamic method dispatch?**

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

**10.What is Dynamic Binding?**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

**11.What is method overloading?**

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows we to implement methods that support the same semantic operation but differ by argument number or type.  
**Note**:

* *Overloaded methods MUST change the argument list*
* *Overloaded methods CAN change the return type*
* *Overloaded methods CAN change the access modifier*
* *Overloaded methods CAN declare new or broader checked exceptions*
* *A method can be overloaded in the same class or in a subclass*

**12.What is method overriding?**

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. **The key benefit of overriding is the ability to define behavior that’s specific to a particular subclass type**.  
**Note**:

* *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: We can’t override a method marked public and make it protected).*
* *We cannot override a method marked final*
* *We cannot override a method marked static*

**13.What are the differences between method overloading and method overriding?**

|  |  |  |
| --- | --- | --- |
|  | **Overloaded Method** | **Overridden Method** |
| **Arguments** | Must change | Must not change |
| **Return type** | Can change | Can’t change except for covariant returns |
| **Exceptions** | Can change | Can reduce or eliminate. Must not throw new or broader checked exceptions |
| **Access** | Can change | Must not make more restrictive (can be less restrictive) |
| **Invocation** | Reference type determines which overloaded version is selected. Happens at compile time. | Object type determines which method is selected. Happens at runtime. |

**Covariant returns:-**it is possible to override method by changing the return type if subclass overrides any method whose return type is non-primitive but it changes its return type is subclass type**.**

**14.Can overloaded methods be override too?**

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

**15.Is it possible to override the main method?**

NO, because main is a static method. A static method can’t be overridden in Java.

**16.How to invoke a superclass version of an Overridden method?**

To invoke a superclass method that has been overridden in a subclass, we must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass’ implementation of the method.

// From subclass

super.overriddenMethod();

**17.What is super?**

super is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.  
**Note**:

* *we can only go back one level.*
* *In the constructor, if we use super(), it must be the very first code, and we cannot access any*this.xxx*variables or methods to compute its parameters.*

**18.How do we prevent a method from being overridden?**

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means “this is the final implementation of this method”, the end of its inheritance hierarchy.

public final void exampleMethod() {  
//  Method statements  
}

**19.What is an Interface?**

An interface is a description of a set of methods that conforming implementing classes must have.  
**Note**:

* *We can’t mark an interface as final.*
* *Interface variables must be static.*
* *An Interface cannot extend anything but another interfaces.*

**20.Can we instantiate an interface?**

We can’t instantiate an interface directly, but we can instantiate a class that implements an interface.

## ****1.Can we create an object for an interface?****

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so we must write a class that implements the interface and fulfill all the methods defined in it.

## ****2.Do interfaces have member variables?****

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

## ****3.What modifiers are allowed for methods in an Interface?****

Only public and abstract modifiers are allowed for methods in interfaces.

## ****4.What is a marker interface?****

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. Thejava.io.Serializable interface and Cloneable are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

## ****5.What is an abstract class?****

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.  
**Note**:

* If even a single method is abstract, the whole class must be declared abstract.
* Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.
* We can’t mark a class as both abstract and final.

## ****6.Can we instantiate an abstract class?****

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

## ****7.What are the differences between Interface and Abstract class?****

|  |  |
| --- | --- |
| **Abstract Class** | **Interfaces** |
| An abstract class can provide complete, default code and/or just the details that have to be overridden. | An interface cannot provide any code at all,just the signature. |
| In case of abstract class, a class may extend only one abstract class. | A Class may implement several interfaces. |
| An abstract class can have non-abstract methods. | All methods of an Interface are abstract. |
| An abstract class can have instance variables. | An Interface cannot have instance variables. |
| An abstract class can have any visibility: public, private, protected. | An Interface visibility must be public (or) none. |
| If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. |
| An abstract class can contain constructors . | An Interface cannot contain constructors . |
| Abstract classes are fast. | Interfaces are slow as it requires extra indirection to find corresponding method in the actual class. |

## ****8.When should I use abstract classes and when should I use interfaces?****

### When to use Interface -

* We see that something in wer design will change frequently.
* If various implementations only share method signatures then it is better to use Interfaces.
* we need some classes to use some methods which we don’t want to be included in the class, then we go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

### When to use Abstract class -

* If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
* When we want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
* Abstract classes are an excellent way to create planned inheritance hierarchies. They’re also a good choice for nonleaf classes in class hierarchies.

## ****9.When we declare a method as abstract, can other nonabstract methods access it?****

Yes, other nonabstract methods can access a method that we declare as abstract.

## ****10.Can there be an abstract class with no abstract methods in it?****

Yes, there can be an abstract class without abstract methods.

## ****11.What is Constructor?****

* A constructor is a special method whose task is to initialize the object of its class.
* It is special because its name is the **same as the class name**.
* They do not have return types, not even **void** and therefore they cannot return values.
* They **cannot be inherited**, though a derived class can call the base class constructor.
* Constructor is invoked whenever an object of its associated class is created.

## ****12.How does the Java default constructor be provided?****

If a class defined by the code does **not** have any constructor, compiler will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

## ****13.Can constructor be inherited?****

No, constructor cannot be inherited, though a derived class can call the base class constructor.

## ****14.What are the differences between Contructors and Methods?****

|  |  |  |
| --- | --- | --- |
|  | **Constructors** | **Methods** |
| **Purpose** | Create an instance of a class | Group Java statements |
| **Modifiers** | Cannot be abstract, final, native, static, or synchronized | Can be abstract, final, native, static, or synchronized |
| **Return Type** | No return type, not even void | void or a valid return type |
| **Name** | Same name as the class (first letter is capitalized by convention) — usually a noun | Any name except the class. Method names begin with a lowercase letter by convention — usually the name of an action |
| ***this*** | Refers to another constructor in the same class. If used, it must be the first line of the constructor | Refers to an instance of the owning class. Cannot be used by static methods. |
| ***super*** | Calls the constructor of the parent class. If used, must be the first line of the constructor | Calls an overridden method in the parent class |
| **Inheritance** | Constructors are not inherited | Methods are inherited |

## ****15.How are this() and super() used with constructors?****

* Constructors use this to refer to another constructor in the same class with a different parameter list.
* Constructors use super to invoke the superclass’s constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

## ****16.How are this() and super() used with constructors?****

* Constructors use this to refer to another constructor in the same class with a different parameter list.
* Constructors use super to invoke the superclass’s constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

## ****17.What are Access Specifiers?****

One of the techniques in object-oriented programming is encapsulation. It concerns the hiding of data in a class and making this class available only through methods. Java allows we to control access to classes, methods, and fields via so-called access specifiers..

## ****18.What are Access Specifiers available in Java?****

Java offers four access specifiers, listed below in decreasing accessibility:

* **Public**- public classes, methods, and fields can be accessed from everywhere.
* **Protected**- protected methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
* **Default(no specifier)-**If we do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
* **Private**- private methods and fields can only be accessed within the same class to which the methods and fields belong. private methods and fields are not visible within subclasses and are not inherited by subclasses.

## ****19.What is final modifier?****

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

* ***final* Classes**- A final class cannot have subclasses.
* ***final* Variables**- A final variable cannot be changed once it is initialized.
* ***final* Methods**- A final method cannot be overridden by subclasses.

**1.What are the uses of final method?**

There are two reasons for marking a method as final:

* Disallowing subclasses to override the method in subclass.
* Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

**2.What is static block?**

Static block gets executed when the class is first loaded into JVM. Static block gets executed before the main method is called.

**3.What are static variables?**

Static keyword is used to define class members in Java . Only one copy of static member is created by the JVM and the same is shared between all the instances of that class .

**static int count ;**

where the name of the variable is **count**and its data type is **int**.  
**Note**: Static variables that are not explicitly initialized in the code are automatically initialized with a default value. The default value depends on the data type of the variables.

**4.What is the difference between static and non-static variables?**

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

**5.What are static methods?**

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.  
**Note**:The use of a static method suffers from the following restrictions:

* *A static method can only call other static methods.*
* *A static method must only access static data.*
* *A static method****cannot****reference to the current object using keywords super or this.*

**6.What is an Iterator ?**

* The Iterator interface is used to step through the elements of a Collection.
* Iterators let we process each element of a Collection.
* Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
* Iterator is an Interface implemented a different way for every Collection.

**7.How do we traverse through a collection using its Iterator?**

To use an iterator to traverse through the contents of a collection, follow these steps:

* Obtain an iterator to the start of the collection by calling the collection’s ***iterator()***method.
* Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()***returns **true**.
* Within the loop, obtain each element by calling **next()**.

**8.How do we remove elements during Iteration?**

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

**9.What is the difference between Enumeration and Iterator?**

Enumeration is used to iterate legacy collection classes.

Itrerator is fail-fast where as Enumeration is not.

**10. What is list iterator ?**

**ListIterator** is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us remove an element**.**

**11.What is the List interface?**

* The List interface provides support for ordered collections of objects.
* Lists may contain duplicate elements.

**12.What are the main implementations of the List interface ?**

The main implementations of the List interface are as follows :

* **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
* **Vector** : Synchronized resizable-array implementation of the List interface with additional “legacy methods.”
* **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

**13.What are the advantages of ArrayList over arrays ?**

Some of the advantages ArrayList has over arrays are:

* It can grow dynamically
* It provides more powerful insertion and search mechanisms than arrays.

**14.Difference between ArrayList and Vector ?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is **NOT** synchronized by default. | Vector List is synchronized by default. |
| ArrayList can use only Iterator to access the elements. | Vector list can use Iterator and Enumeration Interface to access the elements. |
| The ArrayList increases its array size by 50 percent if it runs out of room. | A Vector defaults to doubling the size of its array if it runs out of room |
| ArrayList has no default size. | While vector has a default size of 10. |

**15.How to obtain Array from an ArrayList ?**

Array can be obtained from an ArrayList using ***toArray()***method on ArrayList.

List arrayList = new ArrayList();  
Object  a[] = **arrayList.toArray()**;

**16.Why insertion and deletion in ArrayList is slow compared to LinkedList ?**

* **ArrayList**internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
* During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion.
* In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

**17.Why are Iterators returned by ArrayList called Fail Fast ?**

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**18.How do we decide when to use ArrayList and When to use LinkedList?**

If we need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection.

If, however, we need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

**19.What is the Set interface ?**

* The Set interface provides methods for accessing the elements of a finite mathematical set
* Sets do not allow duplicate elements
* Contains no methods other than those inherited from Collection
* It adds the restriction that duplicate elements are prohibited
* Two Set objects are equal if they contain the same elements

**20.What are the main Implementations of the Set interface ?**

The main implementations of the Set interface are as follows:

* HashSet
* TreeSet
* LinkedHashSet
* EnumSet

**1.What is a HashSet ?**

* A HashSet is an unsorted, unordered Set.
* It uses the hashcode of the object being inserted (so the more efficient wer hashcode() implementation the better access performance we’ll get).
* Use this class when we want a collection with no duplicates and we don’t care about order when we iterate through it.

**2.What is a TreeSet ?**

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

**3.What is an EnumSet ?**

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

**4.Difference between HashSet and TreeSet ?**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| HashSet is under set interface i.e. it  does not guarantee for either sorted order or sequence order. | TreeSet is under set i.e. it provides elements in a sorted  order (acceding order). |
| We can add any type of elements to hash set. | We can add only similar types of elements to tree set. |

**5.What is a Map ?**

* A map is an object that stores associations between keys and values (key/value pairs).
* Given a key, we can find its value. Both keys  and  values are objects.
* The keys must be unique, but the values may be duplicated.
* Some maps can accept a null key and null values, others cannot.

**6.What are the main Implementations of the Map interface ?**

The main implementations of the List interface are as follows:

* HashMap
* HashTable
* TreeMap
* EnumMap

**7.What is a TreeMap ?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key’s class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

**8.How do we decide when to use HashMap and when to use TreeMap ?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, we need to traverse the keys in a sorted order, then TreeMap is wer better alternative. Depending upon the size of wer collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

**9.Difference between HashMap and Hashtable ?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| HashMap lets we have null values as well as one null key. | HashTable  does not allows null values as key and value. |
| The iterator in the HashMap is fail-fast  (If we change the map while iterating, an exception will be thrown). | The enumerator for the Hashtable is not fail-safe. |
| HashMap is unsynchronized. | Hashtable is synchronized. |

**Note**: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

**10.How does a Hashtable internally maintain the key-value pairs?**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key’s class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

**11.What Are the different Collection Views That Maps Provide?**

Maps Provide Three Collection Views.

* **Key Set**- allow a map’s contents to be viewed as a set of keys.
* **Values Collection** – allow a map’s contents to be viewed as a set of values.
* **Entry Set** – allow a map’s contents to be viewed as a set of key-value mappings.

**12.What is a KeySet View ?**

KeySet is a set returned by the ***keySet()*** method of the Map Interface, It is a set that contains all the keys present in the Map.

**13.What is a Values Collection View ?**

Values Collection View is a collection returned by the ***values()*** method of the Map Interface, It contains all the objects present as values in the map.

**14.What is an EntrySet View ?**

Entry Set view is a set that is returned by the ***entrySet()*** method in the map and contains Objects of type Map. Entry each of which has both Key and Value.

**15.How do we sort an ArrayList (or any list) of user-defined objects ?**

Create an implementation of the *java.lang.Comparable* interface that knows how to order wer objects and pass it to *java.util.Collections.sort*(List, Comparator).

**16.What is the Comparable interface ?**

The Comparable interface is used to sort collections and arrays of objects using the Collections.sort() and java.utils.Arrays.sort() methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

interface Comparable<T>

*where T is the name of the type parameter.*

All classes implementing the Comparable interface must implement the compareTo() method that has the return type as an integer. The signature of the compareTo() method is as follows:

int i = object1.compareTo(object2)

* If object1 < object2: The value of i returned will be negative.
* If object1 > object2: The value of i returned will be positive.
* If object1 = object2: The value of i returned will be zero.

**17.What are the differences between the Comparable and Comparator interfaces ?**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| It uses the *compareTo()* method.*int objectOne.compareTo(objectTwo).* | t uses the *compare()*method.*int compare(ObjOne, ObjTwo)* |
| It is necessary to modify the class whose instance is going to be sorted. | A separate class can be created in order to sort the instances. |
| Only one sort sequence can be created. | Many sort sequences can be created. |
| It is frequently used by the API classes. | It used by third-party classes to sort instances. |

**What is Java collection framework ?**

 A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details. A collection is an object that represents a group of objects .

**Why Collection does not extend Serializable or Cloneable ?**

Think of a scenario where a collection is backed by a large database. Is it really meaningful to clone this collection ? I think no.  
I agree that some collections might require cloning , but not all . Thats why the Collection interface does not extend Cloneable but some concrete implementations do provide a public clone method.  
This holds true for Serialization as well. Serializing a collection that holds very large data is not at all good.

**Why Map interface does not extends Collection interface ?**

Map is not a collection of objects like List or Set , rather it is a collection of mappings ( key-value pair ) . Map and collection have a separate hierarchy because they are not compatible . Collection provides a method add() which can’t be used by a map.  
On the other hand , map provides collection views through the methods keySet , entrySet and values.

**What is UnsupportedOperationException?**

UnsupportedOperationException is a runtime exception that is used by the collections to indicate that the requested operation is not supported.  
**For example -** UnmodifiableCollection throws this exception for add and remove methods.

**What is read-only collection ? How can we create the same?**

The read-only collections are those whose contents cannot be modified and any such attempt will result in **UnsuppotedOperationException**.  
Collections utility class provides various methods that returns unmodifiable views of the specified collection.For example , If we create a unmodifiable list and then try to add some data in it , we get UnsuppotedOperationException.Collection readOnlyList = Collections.unmodifiableCollection(list);  
// Any attempt to modify the list will result in UnsuppotedOperationException.  
readOnlyList.add(“new data”);

**What are thread safe collections ?**

Thread safe collection classes are those which are properly synchronized to be used in a multithreaded environment. For example Vector and Hashtable. Collections class contains some utility methods to make a thread safe collection.  
For Example – **Collections.synchronizedCollection** returns a thread-safe collection backed by the specified collection.

**What are concurrent collection classes?**

Java 5 added a new Java package to the Java platform , the **java.util.concurrent** package.  This package contains a set of classes that makes it easier to develop concurrent applications in Java.  
Some examples are :  
**ConcurrentMap**  
**ConcurrentNavigableMap**  
**CountDownLatch**

**What is List interface? What are its main implementations?**

List is an ordered collection of elements ( also known as sequence ) . The ordering is index based. A list can have duplicate elements in it and permite multiple null values.  
ArrayList , LinkedList , Stack , Vector are some of its implementations.

**What is the difference between Array and ArrayList?**

* ArrayList is a growing array . That means the size of an Array is fixed but the ArrayList can grow dynamically.The size of an array need to be defined at the time of its initialization.
* Array can contain primitives or Objects whereas ArrayList can only contain Objects.
* ArrayList provides a lot more methods for insertion/retrieval of data as compared to an array.

**What is the difference between ArrayList and Vector?**

Vector is synchronized whereas ArrayList is not.

ArrayList is faster than Vector because it is not synchronized.

By default , Vector doubles the size of its array when it is re-sized internally , whereas an ArrayList increases by half of its size when re-sized.

In general ArrayList is prefered over Vector because we can easily get an unmodifiable list or a synchronized list by using methods of Collections class.

**What is the difference between ArrayList and LinkedList?**

* LinkedList stores the elements in a doubly-linked list where as ArrayList is backed by an array.
* Adding and removal of elements is faster in a LinkedList (constant time insertion or removal ) .The addition and removal of elements is slower in an array because it involves shifting of elements when an element is added in the middle.
* Searching an element is faster in ArrayList as it allows random access and thus gives constant time performance.

**How to get an array from an ArrayList?**

List contains  method **toArray** that can be used for converting a list into an array.  
**Example :**  
List<String> names = new ArrayList<String>();  
list.add(“prakash”);  
list.add(“chauhan”);  
list.add(“java”);  
list.add(“world”);String [] namesArray = names.**toArray**(new String[list.size()]);

**How to sort list in reverse order?**

Use **reverse**method of Collections utility class.  
Collections.**reverse**(list);

**How to convert an array of String to ArrayList?**

The utility class Arrays provides a method **asList**that can be used to convert a list into an array.  
**Example :**  
String[] namesArray = {“prakash”, “chauhan”, “java”, “world”};  
List names = Arrays.**asList**(words);

**What is Set interface ?**

Set is the collection of unique elements i.e it does not support duplicate elements. Set is an unordered collection.  
Some of the concrete implementations are HashSet , LinkedHashSet , TreeSet

**What is EnumSet?**

EnumSet is Set implementation to use with enum types. All of the elements in an enum set must come from a single enum type that is specified, explicitly or implicitly, when the set is created. EnumSet is not synchronized and null elements are not allowed. Any attempt to insert null element will throw **NullPointerException**.

**Difference between HashSet and TreeSet ?**

HashSet is Implemented using a hashtable. HashSet is unordered collection . The add, remove, and contains methods have constant time complexity O(1).  
TreeSet is implemented using a red-black tree. TreeSet is a sorted collection , but add, remove, and contains methods has time complexity of O(log (n)). It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.

**What is Map?**

Map is collection of mappings i.e key-value pairs. It does not extend Collection interface.  
Map is an object that maps keys to values. A map cannot contain duplicate keys and each key can map to at most one value.  
Some concrete implementations of map are HashMap, Hashtable, LinkedHashMap.

**What are different collection views provided by Map interface?**

Three collection views are provided by Map :

* **keySet() :** Returns a Set view of the keys contained in the map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.
* **values():** Returns a Collection view of the values contained in the map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.
* **entrySet():** Returns a Set view of the mappings contained in the map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

**What is the difference between HashMap and Hashtable?**

HashMap and Hashtable both implements Map interface. Below are some differences between them :

* Hashtable is synchronized but HashMap is not . Choose Hashtable if thread safety is a concern.
* HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow anything that is null.

**what is capacity and load factor in HashMap?**

The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.  
When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

**What is collision in HashMap?**

Collision occurs in a HashMap if two objects have same hashcode . Now because the hascode is same , bucket location needs to be the same for both the objects .  
In such a situation , HashMap uses a linked list to store objects.

**What is the difference between HashMap and TreeMap?**

HashMap is an unordered collection whereas TreeMap is sorted.

For inserting, deleting, and locating elements in a Map, the HashMap is the best alternative. If, however, we need to traverse the keys in a sorted order, then TreeMap is the right choice .

Depending upon the size of collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorting.

Thread

**1. What is a Thread?**

Thread is an independent sequential path of execution within a process / program. Multiple threads can run concurrently in a program . Threads are lightweight as compared to the process as at runtime , they exist in a common memory space and can share process’s resources like data and code.  
Threads make the runtime environment asynchronous , allowing different tasks to be performed concurrently . In Java , a thread is represented and controlled by an object of class java.lang.Thread.  
To know what is multitasking , see [What is Multitasking](http://www.tutorialhub.in/2014/10/12/what-is-multitasking/) .

To know more about thread creation using Runnable interface and Thread class , See [Understanding Threads in Java](http://www.tutorialhub.in/2014/10/12/understanding-threads-in-java/).

**2. What is thread priority ? How can we set the priority of a thread ?**

Thread scheduler schedules the threads based on their priority relative to other Runnable threads . A newly created java thread has the same priority as the thread which created it .

Thread priority is an int number whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

setPriority method can be used to set the thread priority .

**3. What is the difference between Thread and Process?**

The table mentioned below lists the differences between a Process and Thread.

|  |  |
| --- | --- |
| **PROCESS** | **THREAD** |
| A **Process** has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources. Each process has its own memory space. For example : When we start Microsoft word , a new process is created . | A **Thread** is also an execution environment , but is lightweight as compared to a process . Thread is also called a “lightweight process”. Thread exists within a process and shares the processes’s resources , including memory and open files. |
| Different Processes have separate address spaces . | Threads share the address space of its process. |
| Context-switching between the process is slower. | Context-switching between threads in same process is comparatively faster. |
| processes interact only through system-provided inter-process communication mechanisms | Threads can easily communicate within a process. |

**4. What are the advantages and Disadvantages of using threads ?**

**ADVANTAGES**

* Better use of system resources.
* Parallelization of tasks . Threads supports concurrent operations . For example multiple requests can be processed simultaneously by different threads.
* Enhanced performance on multi-processor machines.
* Provides better availability in GUI applications.

**DISADVANTAGES**

* Increases the overall complexity of the system . Multithreaded applications are difficult to debug , sometimes gives unpredictable results.
* Need to synchronize the shared resources ( Objects ) to maintain their state.
* Multithreading can result in potential deadlock situations.
* Some threads may not get proper time to run ( Starvation ) .

**5. What are the different ways of creating a thread?**

Threads can be created in two ways :

* Extending Thread class.
* Implementing Runnable interface.

For more details see [Understanding Threads in Java](http://www.tutorialhub.in/2014/10/12/understanding-threads-in-java/).

**6. What are the different states in thread’s life cycle?**

A thread can be in one of the following states:

**NEW**  
A newly instantiated thread that has not yet started is in NEW state.

**RUNNABLE**  
A thread in the runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as processor.

**BLOCKED**  
A thread that is blocked waiting for a monitor lock is in this state.The thread enters in this state after calling Object.wait

**WAITING**  
A thread that is waiting indefinitely for another thread to perform a particular action is in this state.   The thread enters into this state after calling one of these :

**Object.wait** with no timeout  
**Thread.join** with no timeout  
**LockSupport.park**

**TIMED\_WAITING**   
A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state. The thread enters into this state after calling one of these methods with a specified time :

**Thread.sleep**  
**Object.wait** with timeout  
**Thread.join** with timeout  
**LockSupport.parkNanos**  
**LockSupport.parkUntil**

**TERMINATED**  
A thread that has completed its execution is in this state.

**7.What is the use of synchronized keyword?**

synchronized keyword is used to control the access of multiple threads on a shared resource . synchronized keyword can be used with both static and instance methods. It can also be used with a block of code.

**// synchronized keyword on non static method.**  
public void synchronized doStuff(){}

**// synchronized keyword on static method.**  
public static void synchronized doStuff(){}

**// synchronizing a block of code.**                                                                 public void doStuff(){  
synchronized (this){  
// synchronized keyword on block of code  
}  
}

**8. Can we use synchronized keyword with classes and variables ?**

NO. Only methods can be synchronized.

**9. Can a class contain both synchronized and non-synchronized methods?**

YES.

**10. What is the difference when the synchronized keyword is applied to a static method and a non static method?**

When a non static synchronized method is called , lock is taken on the current instance , i.e the one that is used to call the method where as for synchronized static method , lock is taken on the class.

**11. What is a volatile keyword?**

volatile keyword indicates the compiler not to cache the value of a field and always read it from the main memory . The volatile keyword guarantees that –

Any thread accessing a volatile field will read its current value from main memory before continuing , instead of using a cached value.  
Volatile reads and writes establish a happens-before relationship, much like acquiring and releasing a lock .

**12. What is the difference between yield() and sleep() methods ?**

**yield()** allows the current running thread to release the lock so that other threads of same priority can acquire the lock . yield is only a hint to the scheduler and scheduler is totally free to ignore it .

* **public static void yield()**

**sleep()** method causes the currently executing thread to sleep for a specified period of time ( in milliseconds ) . The thread doesn’t release the lock.

* **public static void sleep(long millis) throws InterruptedException**
* **public static void sleep(long millis,int nanos) throws InterruptedException**

**13. What is the difference between wait() and sleep()?**

wait() method is defined in Object class whereas sleep() method is defined in Thread class.  
wait() causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object . Thread releases the lock as soon as the wait is called .

sleep() method causes the currently executing thread to sleep for a specified period of time ( in milliseconds ) . The thread doesn’t release the lock.

**14. What is difference between notify() and notifyAll()?**

notify() method wakes up a single thread that is waiting on this object’s monitor. There might be several threads waiting for this object and only one of them is chosen.

On the other hand , notifyAll() method wakes up all threads that are waiting on this object’s monitor.  
Also see , Inter thread communication using notifyAll() method.

**15. What happens if run method is directly invoked?**

Calling the start() method creates a new thread of execution . It means new thread is created by the JVM and the statements defined in the run method is executed in the newly spawned thread . By calling the run() method directly , the same will be executed in the caller’s thread and not in the new thread .

**16. What happens when start() is called?**

A new thread of execution with a new call stack starts . The state of thread changes from NEW to RUNNABLE . JVM calls the run method of this thread .

**17. Can a start() method be called multiple times on a Thread ?**

It is not legal to start a thread more than once and therefore start() method can only be called once on a thread .

**18 . Can a thread be restarted after completing its execution ?**

NO . A thread may not be restarted once it has completed execution.

**19. What is the priority of a newly created thread ?**

The priority of a newly created thread is equal to the priority of the thread which has created it .

**20. What is daemon thread?**

There are basically two type of threads : User threads and Daemon threads . Daemon threads are service provider threads that run in the background .JVM does not wait for the daemon threads to complete their execution if all user threads have completed their execution.  
A good example is Garbage Collector , which is a daemon thread and is used to reclaim the unused memory .

Thread class provides a method to make a thread daemon –  
**public void setDaemon(boolean status)**

To check if a thread is daemon –  
**public boolean isDaemon()**