1. Parent class and child class constructor relationship.

A child class always has to call it’s parent’s constructor. This is true even in the case of no-arg constructor in the parent class. If a no-arg constructor has not been explicitly defined then an empty(no-arg) constructor will be implicitly provided by the compiler.

In the case of arguments in the parent class, the child class must provide all the arguments to the parent class using ‘super()’. It may accept extra arguments for its use but not at the expense of parent class’s required arguments.

2. What is toString() method ?

The toString() method is defined in the Object class and all classes inherit from the Object class.

The toString() method returns the String representation of an object. Hence we return custom string representation to decide the output when the class is printed to standard console. We can override the method and give it our own implementation. The method has public access specifier by default and we cannot change it.

3. How to prevent instantiation of a class ?

Declare the constructor as private. This explicit declaration prevents the compiler from generating a public no-arg constructor implicitly. Since no code outside the class can access the constructor, instantiation is prevented.

4. Can we change access specifier of method while method overriding ?

Yes but we can only increase the scope of access and not weaken it. For example a public method cannot be overridden to private whereas vice versa is true.

5. Types of Access Modifier ?

**Private:** fields and methods can be accessed only within the class and not from outside the class.

**Default:** fields and methods can be accessed only within the same package and not outside the current package. It is the access modifier provided implicitly to state and methods if we do not explicitly specify the access modifier. Also we cannot explicitly define default access specifier.

**Protected:** fields and methods can be accessed within the same package and also outside the current package with the help of child class only. So inheritance is must for accessing it outside the package.

**Public:** fields and methods can be accessed from anywhere – within the class, outside the class, within the current package and outside the current package.

6. What is functional Interface ?

Functional Interface is an interface which has exactly one abstract method. However, it can have a number of static and default methods. Example is provided. The @FunctionalInterface annotation is an informative annotation that indicates whether or not an interface type declaration is meant to be a functional interface.

7. What are Lambda Expressions ?

Lambda expressions provides implementation of functional interface. Basically the interface should have only one abstract method, hence functional interface. If we have more than one abstract method then we cannot use lambda expression. In such a situation anonymous inner class is preferred. Example is provided.

8. What are anonymous inner classes ?

A class that has no name, anonymous inner class is an inner class without a name for which only a single object is created. It should be used to override a method of class or interface. Anonymous class has an advantage over lambda expressions in the sense that anon inner classes can provide implementation to more than one abstract method whereas lambda expressions cannot.

9. What are anonymous object ?

10. Can classes implementing the interface not provide the abstract method implementation ?

If a class implementing the interface does not provide the method implementation then the class has to declared as an abstract class. It will be the job of another class extending the abstract class to provide the implementation. This pattern will continue until one of the class is concrete and does provide the required implementation.

11. Can objects of Interfaces and abstract classes be created ?

No because it is against the very nature of interface and abstract classes. They both contain abstract methods which lack implementation so creating objects makes no sense. However both abstract class and interface references can be created.

12. Can anonymous classes have static members ?

No they can only have static members if they are constant, i.e. static final int x = 10;

13. Why use Integer.compare(x,y) instead of x-y in comparators or anywhere else ?

To avoid Integer overflow issue due to INT\_MAX - (-1), this will overflow on the MAX side resulting in a INT\_MIN negative value.

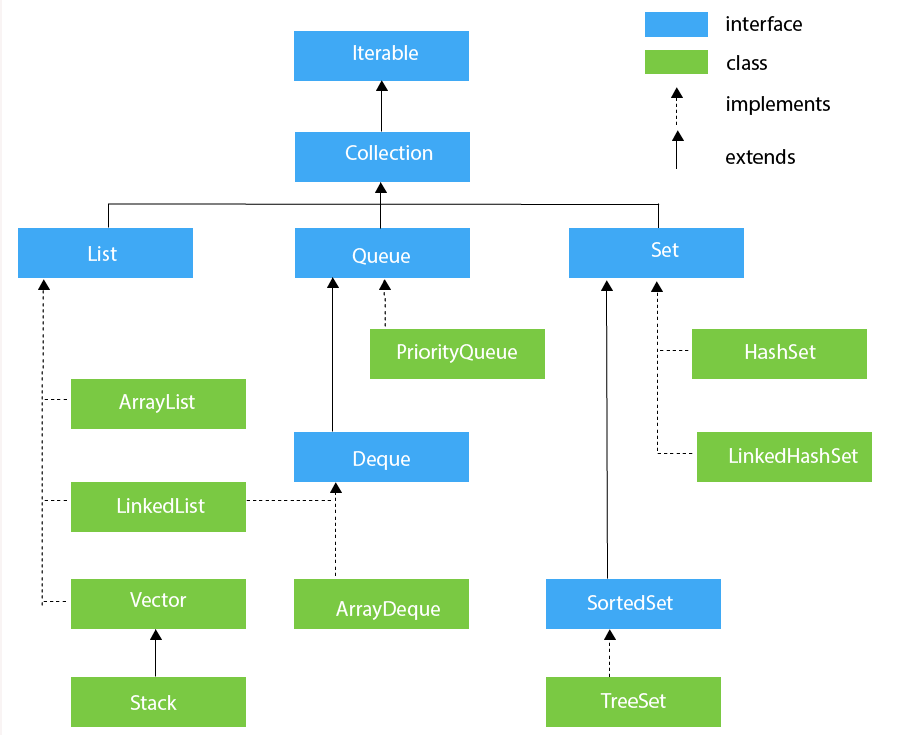
14. What are default methods ?

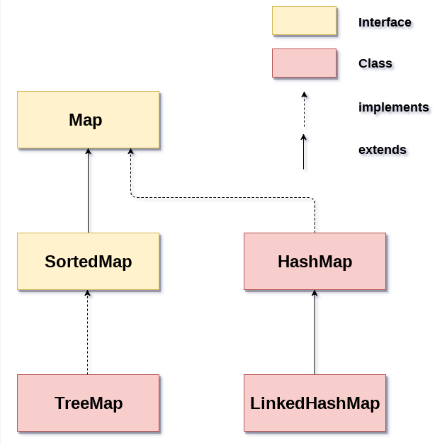
Default methods are non abstract methods which allow us to add a method in the interface which becomes available in all the implementations of the interface, basically the classes. Default methods cannot be abstract but they can definitely be overridden in the classes implementing the interface or another interface extending the previous one.

15. Explain forEach() method ?

It is method to iterate through a collection of items. It is a default method in Iterable interface & Stream Interface. This method takes a single parameter which is a functional interface.

16. Collections Framework hierarchy ?





17. Difference between Comparator and Comparable ?

Comparator is a good choice when we can’t modify the source code of original class of whose objects we want to compare. Using Comparators we can avoid adding additional code to our domain class. We can also define multiple different comparison strategies, which isn’t possible in Comparable.

18. What are the different intermediate and terminal operation in stream ?

**Intermediate –**

It returns a stream and can be chained together to form a pipeline of operations. They are lazy loaded and do not produce end result.

Example **-** map(), filter(), distinct(), sorted(), limit(), skip(), flatMap(), peek()

**Terminal –**

They return non-stream values and can’t be chained together. Pipelined operations can have a max of one terminal operation that too at the end only. Terminal operations are eagerly loaded and they produce end result.

Example - forEach(), toArray(), reduce(), collect(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()

19. Difference between map() and flatMap() ?

Both map and flatMap can be applied to a stream<T> and return a stream<R>. The difference is that the map operation produces one output value for each input value, whereas the flatMap operation produces an arbitrary number of values(0 or more) for each input value.

The map method takes a FunctionalInterface, which is called for each value in the input stream and produces one result value, which is sent to output stream.

The flatMap() takes a FunctionalInterface