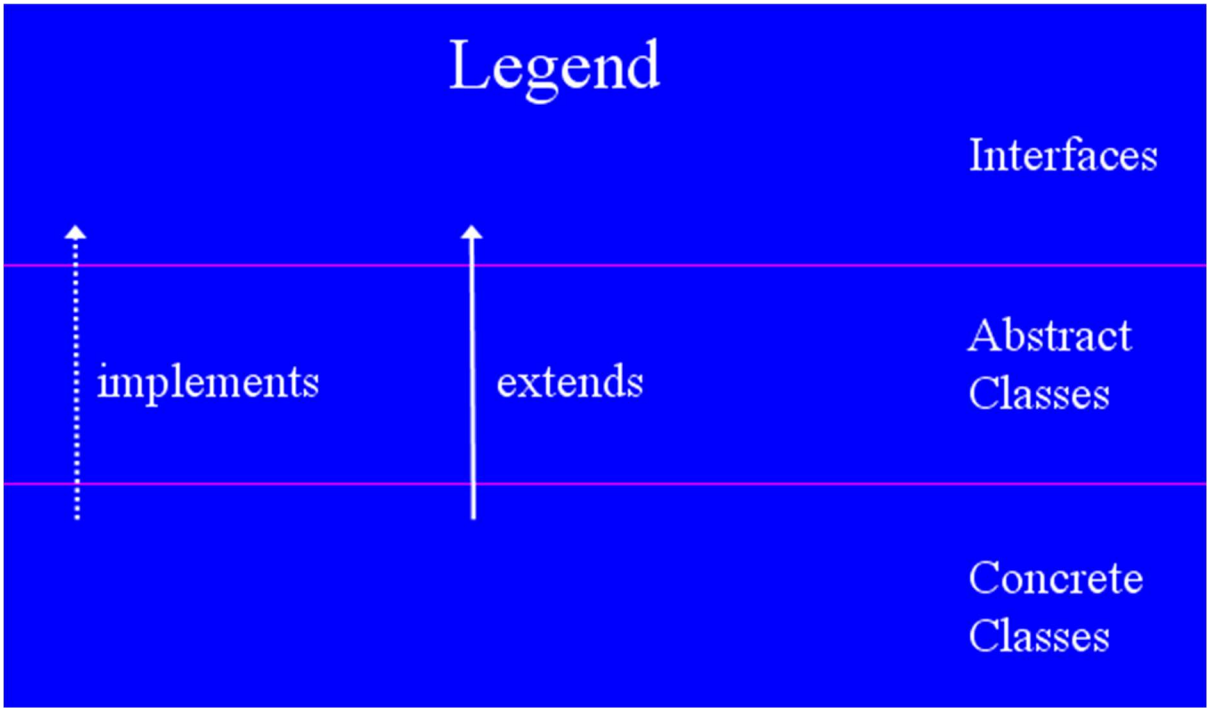
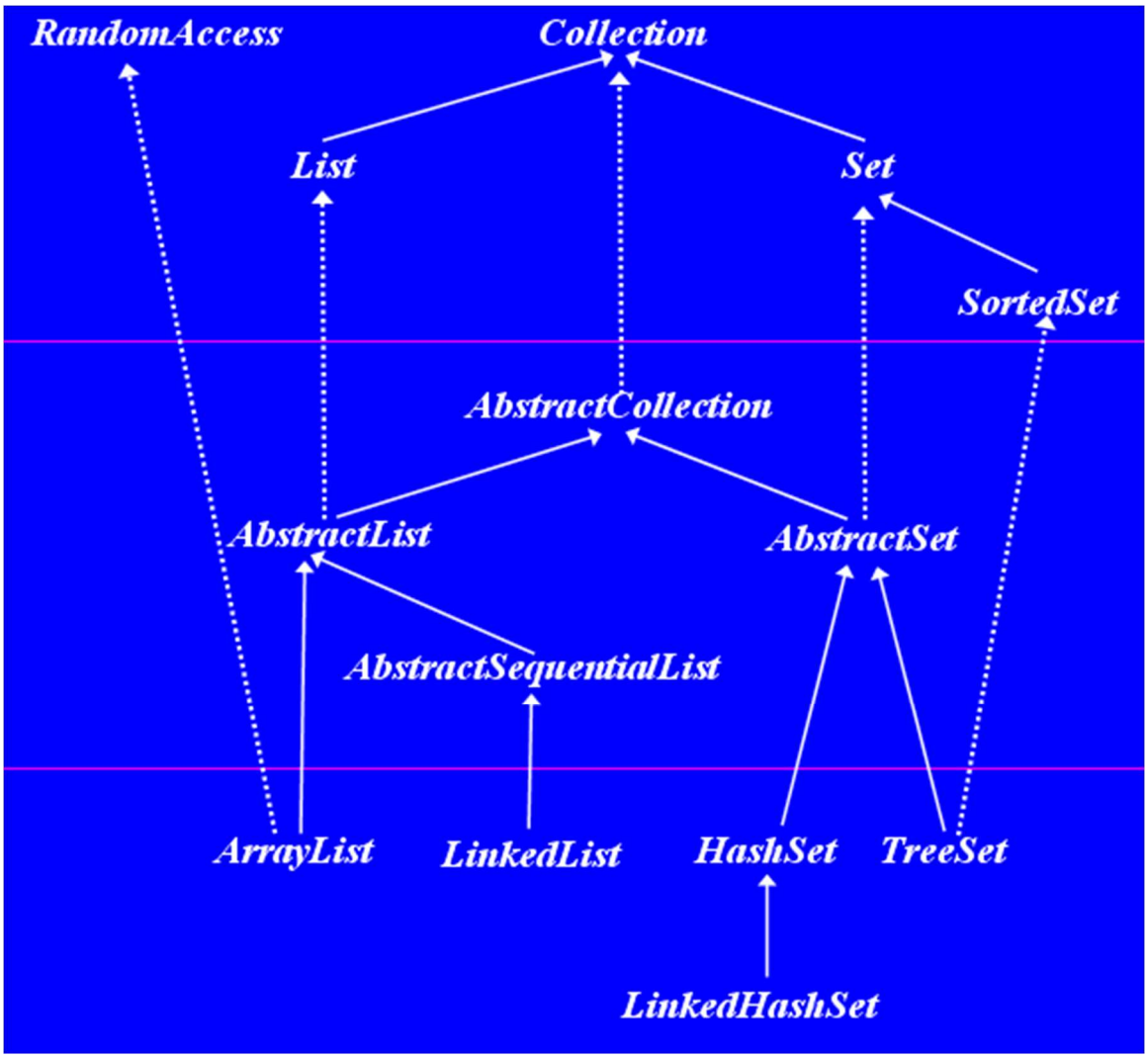
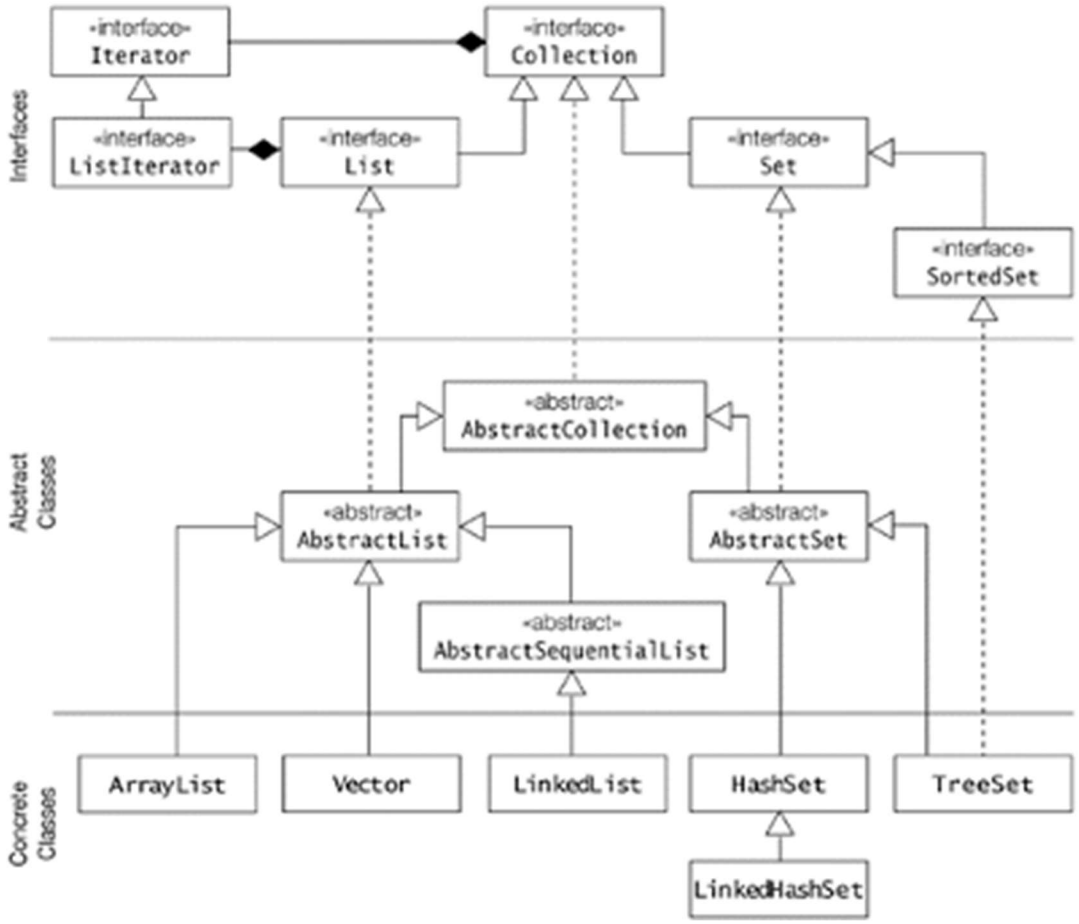
In this section we will examine in detail the common design for all classes that implement the **Collection** interface. We will examine the relationship between this interface and the abstract class that implements it. In the following sections we will study further interfaces, abstract classes, and finally the concrete classes that implement this interface. Throughout our study, these features naturally arrange themselves into three vertical levels in a hierarchy. The following legend explains the three levels and some of the notation used.




For lists and sets, we will explore the following hierarchy of interfaces, abstract classes, and concrete classes. Note that some concrete **List** classes also implement the **RandomAcesss** interface; and some concrete **Set** classes also implement the **SortedSet** interface.




These interfaces are implemented by abstract classes (two or three levels, which supply some but not all of the needed methods) which are extended by concrete classes that inherit some behvaior from the abstract classes and concretely define all their abstract methods. Recall that concrete subclasses automatically implement the interfaces that their superclasses implement, and [abstract]classes can implement more than one interface (but can extend only one superclass); again, this is a fundamental difference between interfaces and classes.


Interfaces
«interface»
Iterator
«interface»
Collection
«interface»
ListIterator
«interface»
List
«interface»
Set
«interface»
SortedSet
Abstract Classes
«abstract»
AbstractCollection
«abstract»
AbstractList
«abstract»
AbstractSet
«abstract»
AbstractSequentialList
Concrete Classes
ArrayList
Vector
LinkedList
HashSet
TreeSet
LinkedHashSet

# Difference between Abstract Class and Interface


1. `abstract` keyword is used to create an abstract class and it can be used with methods also whereas `interface` keyword is used to create interface and it can't be used with methods.
2. Subclasses use `extends` keyword to extend an abstract class and they need to provide implementation of all the declared methods in the abstract class unless the subclass is also an abstract class whereas subclasses use `implements` keyword to implement interfaces and should provide implementation for all the methods declared in the interface.
3. Abstract classes can have methods with implementation whereas interface provides absolute abstraction and can't have any method implementations.
4. Abstract classes can have constructors but interfaces can't have constructors.
5. Abstract class have all the features of a normal java class except that we can't instantiate it. We can use `abstract` keyword to make a class abstract but interfaces are a completely different type and can have only public static final constants and method declarations.
6. Abstract classes methods can have access modifiers as public, private, protected, static but interface methods are implicitly public and abstract, we can't use any other access modifiers with interface methods.
7. A subclass can extend only one abstract class but it can implement multiple interfaces.
8. Abstract classes can extend other class and implement interfaces but interface can only extend other interfaces.
9. We can run an abstract class if it has `main()` method but we can't run an interface because they can't have main method implementation.
10. Interfaces are used to define contract for the subclasses whereas abstract class also define contract but it can provide other methods implementations for subclasses to use.

Thats all for the ***difference between interface and abstract classes***, now we can move on to know when should we use Interface over Abstract class and vice versa.

## Interface or Abstract Class

Whether to choose between Interface or abstract class for providing contract for subclasses is a design decision and depends on many factors, let's see when Interfaces are best choice and when can we use abstract classes.

1. Java doesn't support multiple class level inheritance, so every class can extend only one superclass. But a class can implement multiple interfaces. So most of the times Interfaces are a good choice for providing base for class hierarchy and contract. Also coding in terms of interfaces is one of the best practices for coding in java.
2. If there are a lot of methods in the contract, then abstract class is more useful because we can provide default implementation for some of the methods that are common for all the subclasses. Also if subclasses don't need to implement particular method, they can avoid providing the implementation but in case of interface, the subclass will have to provide implementation for all the methods even though it's of no use and implementation is just empty block.
3. If our base contract keeps on changing then interfaces can cause issues because we can't declare additional methods to the interface without changing all the implementation classes, with abstract class we can provide the default implementation and only change the implementation classes that are actually going to use the new methods.

## Use Abstract classes and Interface both

Actually most of the times, using Interfaces and abstract classes together is the best approach for designing a system, for example in JDK `java.util.List` is an interface that contains a lot of methods, so there is an abstract class `java.util.AbstractList` that provides skeletal implementation for all the methods of List interface so that any subclass can extend this class and implement only required methods.

We should always start with an interface as base and define methods that every subclasses should implement and then if there are some methods that only certain subclass should implement, we can extend the base interface and create a new interface with those methods. The subclasses will have option to choose between the base interface or the child interface to implement according to its requirements. If the number of methods grows a lot, its not a bad idea to provide a skeletal abstract class implementing the child interface and providing flexibility to the subclasses to choose between interface and abstract class.

## Java 8 interface changes

From Java 8 onwards, we can have method implementations in the interfaces. We can create default as well as static methods in the interfaces and provide implementation for them. This has bridge the gap between abstract classes and interfaces and now interfaces are the way to go because we can extend it further by providing default implementations for new methods. For more details, check out Java 8 interface default static methods.