Abstract Class:

Can and Can’t

Can’t

* We can’t create an object for abstract class.
* We can use abstract class when we need to share the same method to all non-abstract subclasses with their own implementation.
* It can have a constructor
* We can have an abstract class without an abstract method
* Not Allows private final static methods
* Multiple Inheritance is not allowed.
* Abstract methods can’t be synchronized. Overridden methods can be synchronized.

Can

* Method declared with abstract modifier has no implementation
* Can define public protected and private concrete methods.
* Can’t be declared static or private.
* Have to implement all abstract methods in subclasses
* It can also have concrete methods
* Abstract method declaration can include throws clause.

Comparison with Interface

* In interface all the fields are automatically public, static final
* You can extend only one class, but implement any number of interfaces.
* Consider using abstract classes if any of these statements apply to your situation:
  + You want to share code among several closely related classes.
  + You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  + You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
* Consider using interfaces if any of these statements apply to your situation:
  + You expect that unrelated classes would implement your interface. For example, the interfaces [Comparable](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html) and [Cloneable](https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html) are implemented by many unrelated classes.
  + You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  + You want to take advantage of multiple inheritance of type.
* An example of an abstract class in the JDK is [AbstractMap](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html" \t "_blank), which is part of the Collections Framework. Its subclasses (which include HashMap, TreeMap, and ConcurrentHashMap) share many methods (including get, put, isEmpty, containsKey, and containsValue) that AbstractMap defines.
* An example of a class in the JDK that implements several interfaces is [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html), which implements the interfaces Serializable, Cloneable, and Map<K, V>. By reading this list of interfaces, you can infer that an instance of HashMap (regardless of the developer or company who implemented the class) can be cloned, is serializable (which means that it can be converted into a byte stream; see the section [Serializable Objects](https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html)), and has the functionality of a map. In addition, the Map<K, V> interface has been enhanced with many default methods such as merge and forEach that older classes that have implemented this interface do not have to define.
* Note that many software libraries use both abstract classes and interfaces; the HashMap class implements several interfaces and also extends the abstract class AbstractMap.