Sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its

subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that

the subclasses must implement.

You may have methods that must be overridden by the subclass in order for the subclass to have any meaning.

In this case, you want some way to ensure that a subclass does, indeed, override all necessary methods. Java's solution

to this problem is the abstract method.

You can require that certain methods be overridden by subclasses by specifying the abstract type modifier.

```
abstract type name(parameter-list);
```

These methods are sometimes referred to as subclass's responsibility because they have no implementation specified in

the superclass.

Thus, a subclass must override them—it cannot simply use the version defined in the superclass.

Any class that contains one or more abstract methods must also be declared abstract.

There can be no objects of an abstract class.

You cannot declare abstract constructors, or abstract static methods.

You can declare static methods in abstract class.

Because there can be no objects for abstract class. If they had allowed to call abstract static methods,

it would that mean we are calling an empty method (abstract) through classname because it is static.

Any subclass of an abstract class must either implement all of the abstract methods in the superclass,

or be declared abstract itself.

Abstract classes can include as much implementation as they see fit i.e.there can be concrete methods(methods with body)

in abstract class.

Although abstract classes cannot be used to instantiate objects, they can be used to create object references,

because Java's approach to run-time polymorphism is implemented through the use of superclass references.

A public constructor on an abstract class doesn't make any sense because you can't instantiate an abstract class directly

(can only instantiate through a derived type that itself is not marked as abstract)

Check: https://stackoverflow.com/questions/260666/can-an-abstract-class-have-a-constructor

Abstract class vs Interface:

Type of methods:

Interface can have only abstract methods.

Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.

Final Variables:

Variables declared in a Java interface are by default final.

An abstract class may contain non-final variables.

Type of variables:

Abstract class can have final, non-final, static and non-static variables.

Interface has only static and final variables.

Implementation:

Abstract class can provide the implementation of interface.

Interface can't provide the implementation of abstract class.

Inheritance vs Abstraction:

A Java interface can be implemented using keyword "implements"

and abstract class can be extended using keyword “extends”.

Multiple implementation:

An interface can extend another Java interface only,

an abstract class can extend another Java class and implement multiple Java interfaces.

Accessibility of Data Members:

Members of a Java interface are public by default.

A Java abstract class can have class members like private, protected, etc.