Object-oriented programming (OOP) languages use abstraction as one of its main ideas. Its major purpose is to deal with complexity by obfuscating superfluous information from the user. This allows the user to build more complicated logic on top of the offered abstraction without having to comprehend or consider all of the underlying complexity.

Interfaces and abstract classes are used in Java to accomplish abstraction. Using interfaces, we can accomplish complete abstraction.

Abstract methods and abstract classes:

* An abstract class is one that has the abstract keyword in its name.
* The term "abstract method" refers to a method that is defined but not implemented.
* All abstract methods may or may not be present in an abstract class. Some of these might be concrete procedures.
* A method that is specified as abstract must always be redefined in the subclass, forcing overriding OR making the subclass abstract.
* Any class having one or more abstract methods must also include the abstract keyword in its declaration.
* An abstract class can't have any objects. An abstract class, on the other hand, cannot be directly created with the new operator.
* The default constructor is always present in an abstract class, and it can contain parameterized constructors.

An example of when to utilize abstract classes and abstract methods

In certain cases, we'll wish to construct a superclass that specifies the structure of an abstraction without giving a complete implementation of all methods. That is, there will be occasions when we wish to construct a superclass that just defines a generalization form that all of its subclasses will use, allowing each subclass to fill in the details.

Consider the famous "shape" example, which may be used to a computer-aided design system or a gaming simulation. "Shape" is the base kind, and each shape has its own color, size, and other characteristics. Specific sorts of shapes—circle, square, triangle, and so on—are derived(inherited) from this, each with its own set of properties and behaviors. Certain forms, for example, may be turned inside out. When you wish to determine the area of a form, for example, some behaviors may change. The type hierarchy encapsulates the forms' similarities and variances.

Diagram

Description automatically generated