What is an interface?

An interface is similar to a class, but it is a collection of abstract methods. These "implicitly" declared abstract methods are required to be implemented in the subclass/es which "implements" them. The methods inside an interface must not be concrete, meaning they must not contain a body.

//Also, everything within an interface is, by default, public.

Name some advantages of using interfaces.

Interfaces provide a standard set of methods for a group of classes. They are sort of a "contract." These methods must be implemented in the subclass/es which "implements" them, and therefore allow us to not forget to implement the important methods required.

Interfaces support multiple inheritance, compared to extending classes which allows us to only extend one. A class can "implement" a lot of interfaces using the "implements" keyword.

The compiler will issue us an error message if the methods in the interface we "implement" do not have an implementation in the subclass we are in.

What are the differences of an abstract class and an interface?

All methods in an interface are implicitly abstract. An interface also supports multiple inheritance. It doesn't have a constructor. Its methods are not concrete, but are only declared as "signatures". Member of an interface cannot be static. It cannot have variable declarations. It uses the keyword "implements."

Not all methods in the abstract class, on the other hand, are abstract. A method in an abstract class can only become abstract when declared so. Also, an abstract class cannot support multiple inheritance. Some of its methods are concrete, some are declared abstract. An abstract can have variable declarations or instances. It uses the keyword "extends" in inheriting behaviors from other classes.

What is the difference of having an interface vs. an abstract class having only abstract methods? Explain your answer.

Actually there is more or less none, given that there are no instance variables and concrete methods declared in that certain abstract class. The only vital difference is that, an abstract class cannot inherit multiple classes. An abstract class, when inheriting, should use the keyword "extends" too. (Plus, having to declare the methods in that abstract class as "abstract" methods one-by-one, since they are not implicitly abstract still, is too tiresome.)

In general, when should we use interfaces and not abstract classes to indicate inheritance and polymorphism?

We use interfaces when we want to have a certain set of standard methods that we want our sub-classes to implement. We should not declare a certain class as an interface when certain instance variables and concrete methods need to be implemented in that class. We should also use interfaces when we want to allow/support multiple inheritance.

Give your own example of a scenario where interfaces can be used. (do not include examples given in class)

A class "Car" can have different types (its sub-classes) like the "Bus", "Jeep", and "Multicab", etc. Under the class "Car" we can declare a set of standard methods like "setColor()", "getColor()", "setYear()", "getYear()", "setBrand()", "getBrand()", "setModel()", "getModel()", etc. The sub-classes "Bus", "Jeep", and "Multicab" will then "implement" the class "Car". This is just to ensure that those standard methods that we set are implemented in the sub-classes.

Can an interface extend an abstract class? Explain your answer.

No. Since an abstract class may contain concrete methods and instance variables. Besides, an interface can only extend a fellow interface.

Do you have any questions for the instructor?

Topics that need more explanation and exercises