Inheritance and Interfaces in Java and UML

Sep 6, 2002

Interfaces

The second I in UML class diagrams are interfaces. The Java programming language also has a concept called an interface that, thankfully matches the UML concept, for all practical purposes. Interfaces are a form of inheritance that differs from that represented by the extends keyword in Java.

Multiple Inheritance

The sort of inheritance represented by the extends keyword in Java is inheritance of interface and implementation. This type of inheritance is sometimes called *sub-classing*. Unlike some object-oriented programming languages, Java classes may extend only a single class. The designers of UML needed to support as many object-oriented programming languages as possible, so they needed to include support for multiple-inheritance, as found in languages such as C++. Creating a class diagram from existing Java source code (often called reverse engineering) does not cause a problem. However, a Java developer asked to implement a UML class diagram that

contains multiple inheritance has a problem. Where the superclasses are pure abstract classes, they can be represented as Java interfaces. However, if this trick proves insufficient, then it is best to go back to the producer of the model and ask them to work with you to refactor the model into a single inheritance model; to do so on your own runs the risk of introducing a misunderstanding of the requirements somewhere.

Java Interfaces

Although Java does not support multiple inheritance in the same way as languages such as C++, it does support the idea of implementation of multiple *interfaces.* The form of inheritance represented by the *implements* keyword in Java is inheritance of interface only. This type of inheritance is sometimes called *sub-typing*. In UML, the relationship between a class implementing an interface and the interface definition is called a *realization relationship*, and it is drawn as a dashed line with a closed arrowhead from the implementing class to the interface. Interfaces are drawn using the same symbol as a class but with an additional keyword <<interface>> above the class name. Figure 4 shows a reworked version of Figure 1 with the Payment class replaced by an interface.

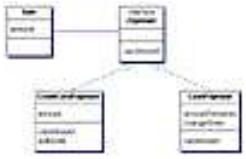

Figure 4 Realization relationships between the definition and implementations of an interface (the equivalent of Java implements).

Interfaces can extend one or more other interfaces. A UML generalization relationship (solid line with closed arrowhead) is used to represent this sort of relationship. Figure 5 illustrates.

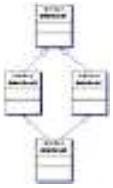

Figure 5 UML generalization relationships, used to indicate that one interface extends one or more others.

UML also supports an alternative notation for interfaces. This shows interfaces as circles (often informally known as the lollipop notation), but is used far more frequently in UML component diagrams than UML class diagrams.

A Shorthand Notation, but Non-Standard

Showing realization relationships to a number of popular interfaces can cause a large class diagram to become a bit of a rat's nest. A shorthand notation—proposed by Peter Coad (and others) in their book, *Java Design*[1], and adopted by Jill Nicola (and others) in their book, *Streamlined Object Modeling*[2]—reduces the clutter by writing the name of interfaces implemented by a particular class in its operations compartment (replacing the list of operations that form that interface). Unfortunately, I don't know of any tools that support this neat little convention. Figure 6 is faked using TogetherSoft's Together ControlCenter[3].

Figure 6 A diagram using the non-standard shorthand notation for interface implementation.

Save To Your Account

< Back **Page 2** of 3 Next >