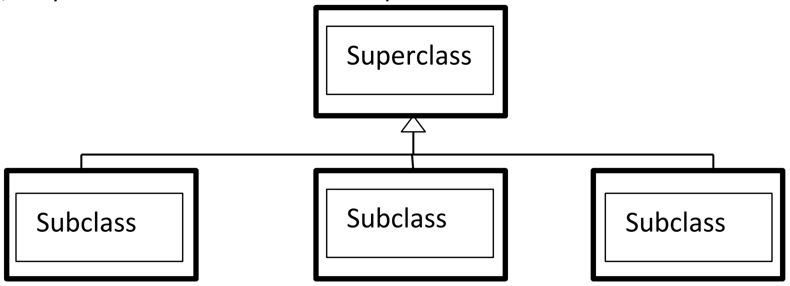
John Krasich
Megan Lusk
Baibhab Dutta

Inheritance Hierarchies

Inheritance is the use of related multiple classes that are used in a program. They are constructed of one general class called a superclass that has more specific classes beneath it. These more specific classes are known as subclasses. Together, they form an Inheritance Hierarchy.



The superclass of every class created is the Object class. When a subclass inherits a superclass, the class is able to use all of the "public" methods provided by the superclass. This is different from implementing a class, which requires the methods of the class to be written. In terms of objects, the set of objects defined in the subclass is a subset of the set of objects defined in the superclass. To inherit a superclass, the subclass must "extend" it. For example

Public class Subclass extends Superclass

While a superclass's methods can be used, any instance variable declared "private" by the superclass cannot be directly accessed by the subclass. Instead, the superclass's methods must be used to access the variable. For example, a subclass of the BankAccount class must use the getBalance() method to access the private instance variable balance. This process is known as encapsulation.

A subclass may want to have a different set of actions for a method that is inherited. This can be done two different ways.

1. Overriding a method uses the same parameters, but has a different set of actions as the superclass's method. If this is done, the superclass's method is still accessible by calling super.methodName .
2. Overloading a method uses different parameters with a different set of actions. If this is done, and the method is not overridden, the super.methodName is not required to access the superclass's method. For example, the JavaEyes program overloaded the constructor for a new Eye, either setting a default color or allowing for a parameter for colors.

While overriding a constructor, the subclass may want to use the superclass's constructor to make its subclass object. To do this, the first line of the subclass's constructor must be super(any parameter(s) required). In order for the code to work correctly, the parameters must fulfill the requirements of the superclass.

Subclasses can also be "converted" to their superclasses by simply assigning them to a superclass object. For example: Superclass other = new Subclass(parameter(s) required) . When this is done, and an overridden method is called, it might seem confusing as to which class's method is used. Java uses the dynamic method lookup to decide which type the object is, and uses that class's method. For example, any overridden method called for in the "other" class previously defined will go to the Subclass's method, because it is an object of the Subclass type. The ability to call these methods as well as the superclass's methods is known as polymorphism.

When programing, it is often smart to make sure that an object is a part of the intended hierarchy. To test this, the instanceof boolean should be used. If the subclass is a part of a superclass, the subclass instanceof superclass would return true. This should be checked before calling a superclass's method to prevent errors from occurring.

To create an object that has the exact same properties of another, the original object may be cloned. This is done by simply calling the clone method rather than new: Object clonedOriginal = (Object) original.clone();)although implementations of clone are error-prone