Introduction to objects

Friday, December 29, 2023

10:46 AM

Type = class

Object = state + behavior + identity

The requests you can make of an object are defined by its interface

Type Name 
Interface 
L Cht It - 
- new Light(); 
lt.on(); 
Light 
on() 
off() 
brighten() 
dim() 

A type has a method associated with each possible request, and when you make a particular request to an object, that method is called. This process is usually summarized by saying that you “send a message” (make a request) to an object, and the object figures out what to do with that message (it executes code)

The preceding diagram follows the format of the Unified Modeling Language (UML). Each class is represented by a box, with the type name in the top portion of the box, any data members that you care to describe in the middle portion of the box, and the methods (the functions that belong to this object, which receive any messages you send to that object) in the bottom portion of the box.

Java uses three explicit keywords to set the boundaries in a class: **public, private, and protected**. These access specifiers determine who can use the definitions that follow. **public** means the following element is available to everyone. The **private** keyword, on the other hand, means that no one can access that element except you, the creator of the type, inside methods of that type. private is a brick wall between you and the client programmer. Someone who tries to access a private member will get a compile-time error. The **protected** keyword acts like private, with the exception that an inheriting class has access to protected members, but not private members.

Java also has a “default” access, which comes into play if you don’t use one of the aforementioned specifiers. This is usually called **package** access because classes can access the members of other classes in the same package (library component), but outside of the package those same members appear to be private.

**Reusing the implementation**

The simplest way to reuse a class is to just use an object of that class directly, but you can also place an object of that class inside a new class. We call this “creating a member object.” Your new class can be made up of any number and type of other objects, in any combination that you need to achieve the functionality desired in your new class. Because you are composing a new class from existing classes, this concept is called **composition** (if the composition happens dynamically, it’s usually called aggregation). Composition is often referred to as a “has-a” relationship, as in “A car has an engine.”

You should first look to composition when creating new classes, since it is simpler and more flexible. If you take this approach, your designs will be cleaner. Once you’ve had some experience, it will be reasonably obvious when you need inheritance.

**Inheritance**

It seems a pity, however, to go to all the trouble to create a class and then be forced to create a brand new one that might have similar functionality. It’s nicer if we can take the existing class, clone it, and then make additions and modifications to the clone. This is effectively what you get with inheritance, with the exception that if the original class (called the base class or superclass or parent class) is changed, the modified “clone” (called the derived class or inherited class or subclass or child class) also reflects those changes.

A type does more than describe the constraints on a set of objects; it also has a relationship with other types. Two types can have characteristics and behaviors in common, but one type may contain more characteristics than another and may also handle more messages (or handle them differently).

When you inherit from an existing type, you create a new type. This new type contains not only all the members of the existing type (although the private ones are hidden away and inaccessible), but more importantly it duplicates the interface of the base class. That is, all the messages you can send to objects of the base class you can also send to objects of the derived class. Since we know the type of a class by the messages we can send to it, this means that the derived class is the same type as the base class.

You have two ways to differentiate your new derived class from the original base class. The first is quite straightforward: You simply add brand new methods to the derived class. These new methods are not part of the base class interface.

The second and more important way to differentiate your new class is to change the behavior of an existing base-class method. This is referred to as **overriding that method**.

**The single rooted hierarchy.**

One of the issues in OOP that has become especially prominent since the introduction of C++ is whether all classes should ultimately be inherited from a single base class. In Java (as with virtually all other OOP languages except for C++) the answer is yes, and the name of this ultimate base class is simply **Object**. It turns out that the benefits of the singly rooted hierarchy are many.

**Parameterized types (generics)**

One of the big changes in Java SE5 is the addition of parameterized types, called generics in Java. You’ll recognize the use of generics by the angle brackets with types inside; for example, an ArrayList that holds Shape can be created like this:

*ArrayList shapes = new ArrayList();*

**Object creation and lifetime**

Java uses dynamic memory allocation, exclusively. Every time you want to create an object, you use the new operator to build a dynamic instance of that object.

Java provides a feature called a garbage collector that automatically discovers when an object is no longer in use and destroys it. A garbage collector is much more convenient because it reduces the number of issues that you must track and the code you must write.

**Exception handling: dealing with errors**

Java’s exception handling stands out among programming languages, because in Java, exception handling was wired in from the beginning and you’re forced to use it.

**Concurrent programming**

Java’s concurrency is built into the language, and Java SE5 has added significant additional library support