The OOP Principles

All object-oriented programming languages provide mechanisms that help you implement the object-oriented model. They are encapsulation, inheritance, and polymorphism. Let's take a look at these concepts.

Encapsulation

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

In java the basis of encapsulation is the class. The following brief discussion will be helpful now. A class defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of given class contains the structure and behavior defined by the class, as if it were stamped out by a mold in the shape of the class. For this reason, objects are some times referred to as instances of a class. Thus, a class is logical construct; an object has physical reality.

When you create a class, you will specify the code and data that constitute that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods. (if you are familiar with C/C++, it may help to know that what a java programs, the method define how the member variables can be used. This means that the behavior and interface of a class are defined by the methods that operate on its instance data.

Each method or variable in a class may be marked private or public. The public interface of a class represents everything that external users of the class need to know, or may know. The private methods and data can only be accessed by code that is a member of the class. Therefore any other code that is not a member of the class cannot access a private method of variable.

Inheritance

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. Most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then any subclass will have the same attributes plus any that it adds as part of its specialization.

Polymorphism

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation. Consider a stack (which is a last-in, first-out list). You might have a program that requires three types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. The algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in java you can specify a general set of stack routines that all share the same names.

More generally, the concept of polymorphism is often expressed by the phrase “ one interface, multiple methods. “ This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the

same interface to be used to specify a general class of action. It is the compiler's job to select the specific action (that is , method) as it applies to each situation.

The differences between abstract class an interface as fallows:

1. Abstract class has the constructor, but interface doesn't.

2. Abstract classes can have implementations for some of its members (Methods), but the interface can't have implementation for any of its members.

3. Abstract classes should have subclasses else that will be useless..

4. Interfaces must have implementations by other classes else that will be useless

5. Only an interface can extend another interface, but any class can extend an abstract class..

6. All variable in interfaces are final by default

7. Interfaces provide a form of multiple inheritance. A class can extend only one other class.

8. Interfaces are limited to public methods and constants with no implementation. Abstract classes can have a partial implementation, protected parts, static methods, etc.

9. A Class may implement several interfaces. But in case of abstract class, a class may extend only one abstract class.

10. Interfaces are slow as it requires extra indirection to to find corresponding method in in the actual class. Abstract classes are fast.

11. Accessibility modifier(Public/Private/internal) is allowed for abstract class. Interface doesn't allow accessibility modifier

12. An abstract class may contain complete or incomplete methods. Interfaces can contain only the signature of a method but no body. Thus an abstract class can implement methods but an interface can not implement methods.

13. An abstract class can contain fields, constructors, or destructors and implement properties. An interface can not contain fields, constructors, or destructors and it has only the property's signature but no implementation.

14. Various access modifiers such as abstract, protected, internal, public, virtual, etc. are useful in abstract Classes but not in interfaces.

15. Abstract scope is upto derived class.

16. Interface scope is upto any level of its inheritance chain.