Interfaces and Abstract classes

Interfaces

An interface contains no executable code or methods. An interface simply contains method definitions. Here is an example of an interface:

```java
public interface Shape
{
    public double area(double x);    //no code; just the header of the method
}
```

Notice that the method definition for area shows only the signature followed by a semicolon. There are no curly braces or code as you would find in a concrete method. Notice also the interface keyword has replaced the class keyword.

You cannot run methods directly from an interface because they are abstract methods, that is, they are not fully defined. You can only put abstract methods in an interface. No concrete methods are allowed. A concrete method is one that has been defined, such as:

```java
public int test(int x)
{
    return x + 5;
}
```

So, this definition of an interface would cause a compiler error:

```java
public interface Sample
{
    public int test(int x)
    {
        return x + 5;
    }
}
```

This interface would compile correctly:

```java
public interface Sample
{
    public int test(int x);
}
```
Here is an example. This interface is called Shape. Notice that it has the first line for a method named area, but there is no code inside the method (it's an abstract method).

```java
/**
 * interface Shape
 *
 * Shape is an interface. As such, it does not actually have the code for any method. It only has the header for the method. It is up to other classes to actually implement the code.
 */

public interface Shape
{
    public double area(double x);    //no code; just the header of the method
}
```

So, who actually implements the code for the area method? The answer is: any class that implements the Shape interface. In this example, Circle is a class that implements the Shape interface. Notice that Circle writes the code for the area method. Use the `implements` keyword for a class that implements an interface.

```java
/**
 * class Circle
 *
 * Circle implements the interface Shape. It must include the method named area and its code.
 */

public class Circle implements Shape
{
    public double area(double a)
    {
        return 3.14 * a * a;           //Voila! actual code for the method.
    }
}
```

Note: an interface cannot have static methods.

A class that is declared as an interface cannot contain concrete methods; it can only contain abstract methods. An interface can specify as many abstract methods as it wants. So the Shape interface could look like this:

```java
public interface Shape
{
    public double area(double x);
```
public void display();
public double perimeter(double a);
public double area(double a, double b);  //overload the Area method
}

Interfaces are useful because Java does not allow multiple inheritance. For example, you cannot do this:

public class Dog extends Animal, Mammal

A class cannot extend more than one class.

However, a class can implement more than one interface, so if Animal and Mammal were interfaces rather than classes, this would be legal:

public class Dog implements Animal, Mammal

A class can extend a class and implement interfaces. For example,

public class Animal
{
    ... code omitted
}

public interface Mammal
{
    ... code omitted
}

public interface Pet
{
    ... code omitted
}

public class Dog extends Animal implements Mammal, Pet
{
    ... code omitted
}

Interface Examples

Here is an interface. It contains only constants.
public interface Constants
{
    double FEET_IN_MILE = 5280;
    String COMPANY_NAME = "Xenon Corporation";
    double OUNCES_IN_POUND = 16;
}

There are two different ways a class can use the interface. In this first example the class gains access to the constant by using the fully qualified name, that is, the interface name followed by a period followed by the constant (Constants.FEET_IN_MILE;)

public class TestContsantsInterface
{
    public static void main(String[] args)
    {
        double miles = 1;
        double feet;
        feet = miles * Constants.FEET_IN_MILE;
        System.out.println(feet);
    }
}

The other way to use the interface is to implement the interface. That way, the unqualified name can be used (i.e. the constant name only).

public class ImplementConstantsInterface implements Constants
{
    public static void main(String[] args)
    {
        double miles = 1;
        double feet;
        feet = miles * FEET_IN_MILE;
        System.out.println(feet);
    }
}

Here is an interface. It contains one method definition.

public interface Conversions
{
    public double feetToMiles(double feet);
}
Here is a class that implements the interface. Notice it contains the implementation of the feetToMiles method.

```java
public class TestConversions implements Conversions {
    //implementation of the method
    public double feetToMiles(double feet) {
        return feet / Constants.FEET_IN_MILE;
    }
}
```

This class actually uses the method.

```java
public class TestIt {
    public static void main(String[] args) {
        TestConversions c = new TestConversions();
        System.out.println(c.feetToMiles(33333));
    }
}
```

### Abstract classes

An abstract class is similar to an interface class with one difference: in an abstract class some of the methods can be concrete.

Use the `abstract` keyword to specify an abstract class.

Use the abstract keyword to declare an abstract method.

Here is an example. House is an abstract class. It declares two methods but only implements one. The other method is left up to the child class to implement.

```java
public abstract class House {
    public abstract void display(); //don't implement this method (abstract method)

    public void test() { //implement this one (concrete method)
        System.out.println("Test method");
    }
}
```
Here is a class that extends the abstract class. Use the `extends` keyword to extend an abstract class.

```java
public class TestAbstract extends House {
    // this class must implement the display method because it is declared abstract in the parent class
    public void display() {
        System.out.println("display method");
    }
}
```