Special Topic 10.1

Abstract Classes

When you extend an existing class, you have the choice whether or not to override the methods of the superclass. Sometimes, it is desirable to *force* programmers to override a method. That happens when there is no good default for the superclass, and only the subclass programmer can know how to implement the method properly.

Here is an example. Suppose the First National Bank of Java decides that every account type must have some monthly fees. Therefore, a `deductFees` method should be added to the `BankAccount` class:

```
public class BankAccount
{
   public void deductFees() { . . . }
   . . .
}
```

But what should this method do? Of course, we could have the method do nothing. But then a programmer implementing a new subclass might simply forget to implement the `deductFees` method, and the new account would inherit the do-nothing method of the superclass. There is a better way—declare the `deductFees` method as an **abstract method**:

```
public abstract void deductFees();
```

An abstract method is a method whose implementation is not specified.

An abstract method has no implementation. This forces the implementors of subclasses to specify concrete implementations of this method. (Of course, some subclasses might decide to implement a do-nothing method, but then that is their choice—not a silently inherited default.)

You cannot construct objects of classes with abstract methods. For example, once the `BankAccount` class has an abstract method, the compiler will flag an attempt to create a `new BankAccount()` as an error. Of course, if the `CheckingAccount` subclass overrides the `deductFees` method and supplies an implementation, then you can create `CheckingAccount` objects.

An abstract class is a class that cannot be instantiated.

A class for which you cannot create objects is called an **abstract class**. A class for which you can create objects is sometimes called a **concrete class**. In Java, you must declare all abstract classes with the reserved word `abstract`:

```
public abstract class BankAccount
{
   public abstract void deductFees();
   . . .
}
```

A class that declares an abstract method, or that inherits an abstract method without overriding it, *must* be declared as abstract. You can also declare classes with no abstract methods as abstract. Doing so prevents programmers from creating instances of that class but allows them to create their own subclasses.

Note that you cannot construct an *object* of an abstract class, but you can still have a *variable* whose type is an abstract class. Of course, the actual object to which it refers must be an instance of a concrete subclass:

```
BankAccount anAccount; // OK
anAccount = new BankAccount(); // Error—BankAccount is abstract
```

```
anAccount = new SavingsAccount(); // OK
anAccount = null; // OK
```

The reason for using abstract classes is to force programmers to create subclasses. By specifying certain methods as abstract, you avoid the trouble of coming up with useless default methods that others might inherit by accident.

Abstract classes differ from interfaces in an important way—they can have instance variables, and they can have concrete methods and constructors.