1. Interfaces :: Objects and Behaviors :: Polymorphism Revisited

An object has **behaviors**, i.e., the object acts in specific ways when **instance methods** are called on it (think of it this way: an instance method is called on an object to invoke some behavior).

For example, if the *Circle* class declares and implements a method named *area*() which returns the area of the *Circle*, then when we call *circleObject.area*() the behavior of the *circleObject* is to calculate and return its area. We can say that "determine your area" is one of the behaviors of a *Circle* object.

Similarly, if we implement a method named *draw*() in the *Circle* class—that when called on a *circle* object will cause the *circle* object to draw itself on the graphical window—then "draw yourself" becomes another behavior of a *Circle* object.

Remember that objects of subclasses inherit behaviors (specifically the public and protected instance methods) from their superclasses. Consequently, the behaviors (instance methods) of objects are declared in various places in the code. For example, let *obj* be an object of some class *Sub* which is a subclass of *Super*. The instance methods (behaviors) of *obj* are declared in:

1. The class *Sub*.
2. The direct superclass *Super*.
3. Nondirect superclasses, e.g., the superclass of *Super*, the superclass of the superclass of *Super*, etc.

Note that the above three items are all related in the sense that the classes are all part of the same inheritance hierarchy.

1. Interfaces :: Objects and Behaviors :: Polymorphism Revisited (continued)

As another example, suppose we declare two classes, *Dog* and *Cat*, both of which are subclasses of abstract class *Mammal*. The *Mammal* class declares an abstract method *makeSound*() that all subclasses must implement (if not the subclass also becomes an abstract class):

```
public abstract class Mammal {     // Abstract class.
  public void makeSound();         // Abstract method. All Mammals make a sound.
}
public class Dog extends Mammal {  // Dog is a subclass of Mammal.
  @Override
  public void makeSound() {        // Overrides makeSound() inherited
    System.out.println("Bark");    // from Mammal. Dogs bark.
  }
}
public class Cat extends Mammal {  // Cat is a subclass of Mammal.
  @Override
  public void makeSound() {        // Overrides makeSound() inherited
    System.out.println("Meow");    // from Mammal. Cats meow.
  }
}
```

Since *Dog* and *Cat* are both *Mammal*s they make a sound, but a *Dog* makes a different sound than a *Cat*. Consequently, the proper way to model this behavior is to declare *makeSound*() as an abstract method in *Mammal* and require subclasses of *Mammal* to provide their own unique implementation of *makeSound*(), i.e., each subclass **overrides** *makeSound*().

1. Interfaces :: Objects and Behaviors :: Polymorphism Revisited (continued)

Furthermore, since *Dog*s and *Cat*s are both *Mammal*s we can pass a *Dog* or *Cat* object as a parameter to a method that expects a *Mammal* as the parameter:

```
public void beNoisy(Mammal pCritter) {
  ...
  pCritter.makeSound(); // This is a polymorphic method call
  ...
}
public void someOtherMethod() {
  Dog fido = new Dog();
  Cat felix = new Cat();
  beNoisy(fido);
  beNoisy(felix);
}
```

Output

```
Bark
Meow
```

If the class of the object parameter *pCritter* is *Dog* then the behavior of *pCritter* will be to "bark." If the class of the object parameter *pCritter* is *Cat* then the behavior of *pCritter* will be to "meow." Remember, this situation is called **polymorphism** and we say that we are making a **polymorphic method call** on *pCritter*.