**Lab 10 Using an interface to share methods**

It is often the case that two or more classes share a common set of methods. For programming purposes we might wish to treat the objects of those classes in a similar way by invoking some of their common routines.  
  
For example, the Dog and Cat classes listed below agree on the void method speak. Because Dog and Cat objects have the ability to “speak,” it is natural to think of putting both types of objects in an ArrayList and invoking speak on every object in the list. Is this possible? Certainly we could create an ArrayList of Dog that would hold all the Dog objects, but can we then add a Cat object to an ArrayList of Dog?  
  
Try running the main program below as it is written. Run it a second time after uncommenting the line that instantiates a Cat object and tries to add it to the ArrayList.

import java.util.\*;  
  
public class AnimalRunner  
{  
 public static void main(String[] args)  
 {  
 ArrayList<Dog> dogcatList = new ArrayList<Dog>();  
 dogcatList.add(new Dog("Fred"));  
 // dogcatList.add(new Cat("Wanda"));  
 }  
}  
  
-------------------  
public class Dog  
{  
 private String name;  
  
 public Dog(String name)  
 {  
 this.name = name;  
 }  
  
 public void speak()  
 {  
 System.out.println("Woof! Woof!");  
 }  
  
 public String toString()  
 {  
 return "Dog: " + name;  
 }  
}  
  
-------------------  
public class Cat  
{  
 private String name;  
  
 public Cat(String name)  
 {  
 this.name = name;  
 }  
  
 public void speak()  
 {  
 System.out.println("Meow! Meow!");  
 }  
  
 public String toString()  
 {  
 return "Cat: " + name;  
 }  
}

Our experiment to add Cat objects to an ArrayList of Dog objects failed. Perhaps we should try using the original Java ArrayList without generics? Try running the code below as it is written along with the Dog andCat classes defined above. Run it a second time after uncommenting the line that invokes speak.

import java.util.\*;  
  
public class AnimalRunner  
{  
 public static void main(String[] args)  
 {  
 ArrayList dogcatList = new ArrayList();  
 dogcatList.add(new Dog("Fred"));  
 // dogList.add(new Cat("Wanda"));  
 for (Object obj : dogcatList)  
 {  
 // obj.speak();  
 }  
 }  
}

The experiment shows that we are now able to add Dog and Cat objects to the ArrayList, but there is a compile error on the line obj.speak because obj is an Object reference variable and the class Object doesn’t contain a speak method. We need a reference variable that can refer to Dog and Cat objects and which also allows us to invoke speak. The solution to the problem uses interfaces.  
  
First create an interface called Speakable that specifies a void speak() method. Be sure to modify the Dog and Cat classes to indicate that they implement the Speakable interface. For example, in the case of theDog class, we will code public class Dog implements Speakable. Be sure to make a similar change in the declaration of the Cat class.  
  
The term Speakable can be used to create Speakable references. Using generics, create an ArrayList of Speakable objects in the main method. Modify the for loop so that it iterates over Speakable objects. Try adding the Dog and Cat objects and invoking the speak method on each object. Does this work?