1. Examine Buyable.java. It’s a Java interface that I wrote. It specifies one abstract method.
2. Examine Song.java,its instance variables and methods. Get a sense of what Song objects are. Change Song.java so that it implements the Buyable interface. You should get an error saying that you have not supplied the getPrice() method yet. The Buyable interface requires a getPrice() method. Write the .getPrice() method. The price for a song is stored in each Song object (it’s an instance variable). The getPrice() method needs to return that price. There should be no errors.
3. Change SmileyFace.java so that it also implements the Buyable interface. It already implements **Comparable**. Add **Buyable** as a second interface for SmileyFace by putting a comma after the word **Comparable** and then add **Buyable** after the comma.

Now you wil have to write the .getPrice() method: the price for a SmileyFace is more complicated than for a Song. It’s dependent on its size. They’re pretty expensive: 35 cents per 1,000 square pixels of area (don’t calculate the area yourself, there’s already a method for it!). If it’s frowning, it should get a 20% discount (if it’s smiling, no discount).

1. Write a client called **Shopper.java** with a main method as follows:
   1. Declare four Buyable object references. Call them b1, b2, b3, b4. Do not declare them as Song or SmileyFace object references, but as Buyable object references.
   2. Make b1 and b2 reference two new Song objects that you instantiate. Use whatever you want for the song data.
   3. Make b3 and b4 reference two new SmileyFace objects that you instantiate. For the radii, use 50 pixels for one and 75 for the other. The other parameters can be anything; that last parameter needs to be **null**.
   4. Make b4 frown. Note: You will not be able to do b4.frown() since b4 is declared as a *Buyable* and *Buyable* doesn’t have a frown() method. To make it frown, you need to **cast** b4 to a SmileyFace object, and then call frown(). Like this:

((SmileyFace)b4).frown()

* 1. Print the price for all four objects as follows:

The price for b1 is: …

The price for b2 is …

The price for b3 is …

The price for b4 is …

**\*\*\*\* IMPORTANT, BIG IDEA\*\*\*\*** The benefit of interfaces is that the client refers to these objects using generic *Buyable* references. All the client knows when using these object references is that they area *Buyable* objects. The only method *Buyable* has is .getPrice(). When the client calls .getPrice(), the actual referred-to object’s method gets executed (i.e. SmileyFace or Song). This is called **Polymorphism** (from ancient-Greek meaning many-forms). Through one reference (b1), price is calculated one way, and through another reference (b3), price is calculated another way. So, the .getPrice() method takes on many forms, and is polymorphic. The client code doesn’t distinguish between the forms; it’s generic in the client; however, when the client runs, different versions of .getPrice() get called depending on what objects are referenced.