Interfaces

- Advantages of abstract classes:

1. They provide a common parent class for similar but distinct classes.
2. They force the subclasses to implement essential methods.

Interfaces

- Here is another use for abstract methods. I have a bunch of classes with different properties. A superclass of them does not make sense. But I still want to be able to make a list of objects of these classes and do a common operation, such as Print, to each of these objects.
- A bad solution is to make the list be a list of Objects. To run the operation on an object from the list, cast the object into its native type, and run the operation on it.

- A better solution is to make an ***interface*** that contains an abstract declaration for the common method, and to force each class to ***implement*** the interface.

- Here is a simple interface declaration:

```
public interface Printable {
        void Print();
}
```

We change our class declarations to say that they implement the interface:

```
public class Person implements Printable {
        .....
```

The compiler will make sure that the class implements each method listed in the interface declaration.

Interfaces can serve as the base type for arrays and lists:

```
        Printable L = new Printable[ ]
```

Question: What is the difference between an Abstract class and an Interface?

A. You can't make an object of an Abstract class but you can of an Interface.
B. You can't make an object of an Interface but you can of an Abstract class.
C. Abstract classes are complete classes with some methods not filled in. Interfaces describe one or more properties of the class.

Answer C: You can’t construct an object of an abstract class because you don’t have code for all of its methods. An interface doesn’t even describe a whole class; you can’t construct an object of an interface, so answers (A) and (B) are wrong.

Answer (C) correctly describes the difference between abstract classes and interfaces.

Question: I have a class Person and want to make subclasses for CollegeStudent and ConStudent. How would you do that?

A. Make an abstract subclass of Person called Student and have CollegeStudent and ConStudent both extend that.
B. Make an interface called Student and have both College Student and ConStudent(which are subclasses of Person) implement that.