

Quiz #1 Answers

1. Imagine the following line in a Java program. What is this really declaring and what must be done before this can be used?

```
MyScreen scrn;
```

This line of code declares a variable, scrn, that can reference an object of type MyScreen. The key point is that it is a reference. This references won't have a valid value to start with. If this line appears inside a method you can't do anything with this variable until you initialize it to point to a real object. Often we do this by calling new to get a reference to a new object. This might look like the following.

```
scrn=new MyScreen ();
```

2. What two things does inheritance provide in Java? Which one do you get when you inherit from an interface?

Inheritance provides code reuse and subtyping. The code reuse is where the name comes from. When one class inherits from another, it implicitly gets all of the methods and data members of the superclass. The inheriting class is also a subtype of the inherited class. This means that any place you use the supertype you can use the subtype. That is what gives us inclusion polymorphism in Java.

Interfaces say what methods they have, but don't provide implementations. So with this lack of code we can't possibly have code reuse. Inheriting from an interface provides us with subtyping only. This turns out to be much more powerful than one might at first think.

Extra Credit: What is an anonymous inner class? Where do they go in code, why do they have that name and how do you write them?

As the name implies, an anonymous inner class is an inner class that doesn't have a name. We write these directly into methods in Java and put the body of the class after a usage of new to instantiate the supertype.