Computer Programming II Instructor: Greg Shaw

COP 3337

# Introduction to Interfaces

1. **Interfaces vs. Classes**

An *interface* is similar to a class, but there are several important differences:

1. All methods in an interface are *abstract*. That is, they have a name and a return type, and they may have parameters, but they have no *implementation* (i.e., no "body", not even an empty set of {}'s)

* As of Java 8, interfaces may also have **static** and **default** methods. We will consider these later

1. All methods in an interface are *automatically* public. The keyword **public** does not appear.
2. Interfaces have no instance variables, because *no objects of the interface type can be created!*

1. Any variables declared in an interface are *automatically* **public,** **static,** and **final** (i.e. *constants*)**.** For that reason, these keywords should not appear in the declaration.
2. Interfaces cannot have a constructor. Again, this is because no objects of the interface type can be created.
3. **Why Use Interfaces?**

* Interfaces promote software “*extensibility*.” Once you have defined classes that *implement* an interface, you treat objects of those classes as objects of the interface type. You can then add new classes to the system with no modification of existing software, because that software depends only on the interface type and not on any of the classes that implement it
* Interfaces are commonly used for classes that are essentially similar, but differ in the details. For example, a screen manager needs to be able to draw and erase a variety of different shapes. We could define an interface called *Shape* with abstract methods *draw* and *erase*. We could then have classes called *Circle*, *Rectangle*, *Triangle*, *etc*. that implement the *Shape* interface. These classes would each have their own implementations of the *draw* and *erase* methods.

1. **Declaring an Interface**

An interface declaration begins just like a class declaration except the keyword **interface** is used instead of **class**. E.g.,

**public interface** Measurable

{

/\*\*

Computes the measure of the object.

@return the measure

\*/

**double** getMeasure() ;

}

(Note that the method *getMeasure* has no body and keyword **public** does not appear. All interface methods are *automatically* public)

1. **Implementing (aka: “Realizing”) an Interface**
2. Any class may *implement* an interface as long as
3. an *implements* clause follows the class name in the declaration, and

1. the class provides an implementation for all the abstract methods declared in the interface.

**public class** Coin **implements** Measurable

{

**private double** value ;

**private String** name ;

.

.

.

**public double** getMeasure()

{

**return** value ;

}

}

1. A class may implement *multiple* interfaces. In that case, the names of the interfaces implemented appear after the keyword **implements**, separated by commas. Naturally, the class would then have to implement all the methods of all the interfaces listed.

**public class** A **implements** B, C, D