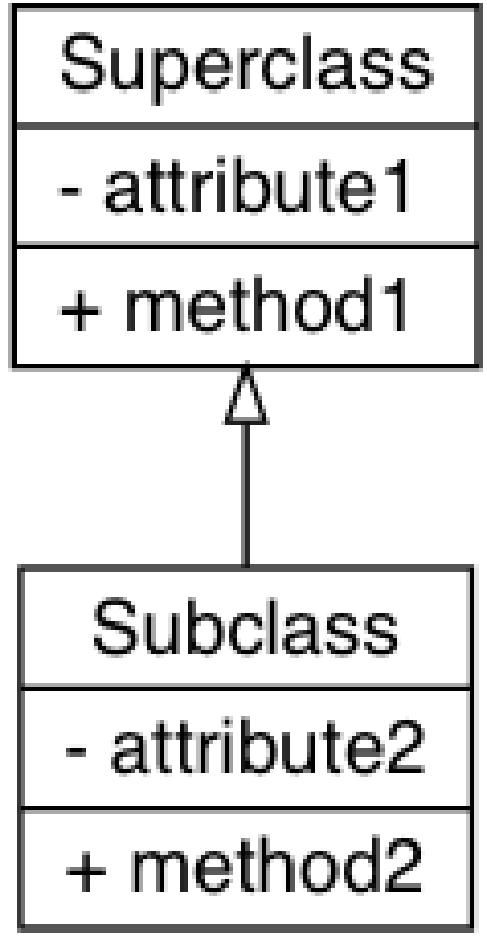
CSC 243 - Java Programming

Inheritance and Subtype Polymorphism

Inheritance

- Inheritance in Java is when one class is *based* on another class
- The base class is called the superclass
- The class inheriting from the superclass is called the subclass
- The subclass inherits all accessible attributes and methods from the superclass and may add new attributes and methods



Java Protected Accessibility

- A `private` attribute or method can not be directly accessed by a subclass
- A `protected` attribute or method can be accessed by any subclass or any class in the same package

Java Inheritance Syntax

- In Java, modeling the class inheritance relationship is done by using the `extends` keyword

```
public class Subclass extends Superclass
```

The `this` and `super` keywords

- The `this` keyword is a reference to the calling object
- The `super` keyword refers to the superclass
- The `super` keyword can be used in two ways
  - To call a superclass constructor
  - To call a superclass method

Overloading and Overriding Methods

- Overloading is ability to define multiple methods with the same name but different signatures
- Overriding is the ability to provide a different implementation of a method in a subclass
- An overridden method has the same name and signature as the method in the superclass
- Java provides an annotation for overriding methods

```
public class C2 extends C1 {
  @Override
  public String toString() {
    return super.toString() + "C2";
  }
}
```

Preventing Extending and Overriding

- The `final` keyword can prevent a class from being extended

```
public final class C
```

- The `final` keyword can also prevent a method from being overridden

```
public class C {
  public final method m() {}
}
```

Subtype Polymorphism

- In Java, a class defines a type
- A type defined by a subclass is called a subtype
- A type defined by a superclass is called a supertype
- Subtype polymorphism allows a variable of a supertype to refer to a subtype object

Declared and Actual Types

- The *declared* type of a variable is type that declares a variable
- The *actual* type of a variable is the is the type that it is constructed as
- When a method is invoked by an object, the actual type is used to determine the appropriate method to call
- Example

```
// the declared type for o is Object
// the actual type of o is String
Object o = new String("Hi");
System.out.println(o.toString());
```

Object Casting

- Implicit casting occurs when an object's declared type is a superclass of the actual type

```
// the String object is implicitly
// casted to type Object
Object o = new String("Hi");
```

- Explicit casting must be performed to convert a superclass to a subclass

```
// the Object o must be converted
// to a String type
String s = (String)o;
```

The `instanceof` operator

- When casting objects to a subclass, if the subclass object is not an instance of the superclass object a `ClassCastException` is thrown
- The `instanceof` operator returns the actual type of the variable

```
if (o instanceof String) {
  String s = (String)o;
}
```