5. Inheritance :: Implementing Inheritance in Java

To specify that a subclass *Sub* inherits from a superclass *Super* we write:

```
public class Sub extends Super {
   ...
}
```

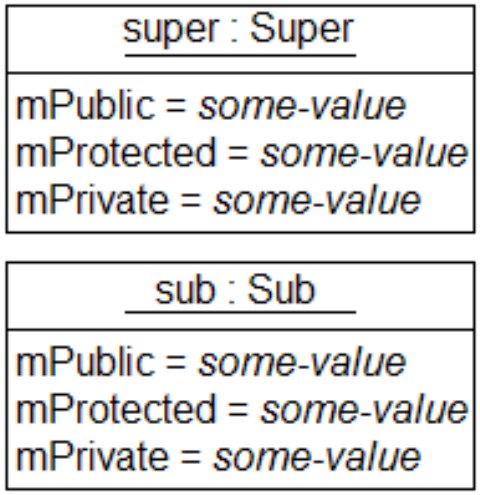
Sub inherits *all* instance variables—whether **public**, **protected**, or **private**—from *Super*. For example:

```
public class Super {
   // mPublic is public for illustration purposes. Do not ever declare a public
   // instance variable.
   public int mPublic;
   protected int mProtected;
   private int mPrivate;
   ...
}

public class Sub extends Super {
   // mPublic is inherited from Super.
   // mProtected is inherited from Super.
   // mPrivate is inherited from Super.
   ...
}
```

5. Inheritance :: Inheritance of Instance Variables

Consider the instantiation of *Super* and *Sub* objects named *super* and *sub*, respectively:



When I was learning OO programming, I used to get confused about which instance variables *Sub* objects inherit from *Super*; in particular, it made sense that *sub* would inherit *mPublic* and *mProtected* from *Super* but inheriting *mPrivate* seemed contradictory since *mPrivate* is **private** within *Super*. However, the rule for inheriting instance variables is this: **a subclass object inherits all instance variables from the superclass declaration.**

5. Inheritance :: Inheritance of Instance Variables (continued)

What the accessibility modifiers **public**, **protected**, and **private** specify regarding attributes declared in a superclass, is whether or not those attributes are **directly accessible** in a subclass. In particular, the rules are:

1. **public** attributes are directly accessible in subclass objects. In fact, **public** attributes of objects are directly accessible from the methods of objects of *any* class—which is why we **never** declare public attributes.
2. **protected** attributes are directly accessible in subclass objects, but they **are not** directly accessible from within the methods of objects of other classes.
3. **private** attributes are only accessible in objects of the superclass; they **are not** directly accessible from within the methods of subclass objects.

5. Inheritance :: Inheritance of Instance Variables (continued)

Consequently, within the *sub* object *mPublic* and *mProtected* are directly accessible, but *mPrivate* is not. Example:

```
public class Sub extends Super {
  public Sub() {
    mPublic = 0;      // Legal because mPublic is public.
    mProtected = 0;   // Legal because mProtected is protected.
    mPrivate = 0;     // Illegal because mPrivate is private.
  }
}
```

The last case is why I used to get confused about **private** instance variables being inherited. They **are inherited**, but they are not directly accessible. So how do we access a **private** inherited instance variables?

5. Inheritance :: Accessing Private Instance Variables in a Subclass

How do we access a **private** inherited instance variables? Through superclass-declared accessor and mutator methods. Example:

```
public class Super {
  // mPublic is public for illustration purposes. Do not ever declare a public
  // instance variable.
  public int mPublic;
  protected int mProtected;
  private int mPrivate;
  ...
  public int getPrivate() { return mPrivate; }
  public void setPrivate(int pNewPrivate) { mPrivate = pNewPrivate; }
}

public class Sub extends Super {
  public Sub() {
    mPublic = 0;           // Legal because mPublic is public.
    mProtected = 0;        // Legal because mProtected is protected.
    setPrivate(0);         // Legal because setPrivate() is public.
    int x = getPrivate(); // Legal because getPrivate() is public.
  }
}
```