6. Inheritance :: Accessibility Specifiers and Instance Methods

The meaning of the accessibility modifiers **public**, **protected**, and **private** regarding superclass methods and subclass objects is the same as for superclass attributes:

1. **public** methods may be called from subclass objects. In fact, **public** methods of objects are callable from the methods of objects of *any* class.
2. **protected** methods may be called from subclass objects, but they **are not** callable from within the methods of objects of other classes.
3. **private** methods are only callable in objects of the superclass; they **are not** callable from within the methods of subclass objects.

6. Inheritance :: Accessibility Specifiers and Instance Methods (continued)

```
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;
  private int mPrivate2;
  ...
  protected int getPrivate() { return mPrivate; }
  protected void setPrivate(int pNewPrivate) { mPrivate = pNewPrivate; }
  public int getPrivate2() { return mPrivate2; }
  public void setPrivate2(int mNewPrivate2) { mPrivate2 = pNewPrivate2; }
}

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 protected.
    int x = getPrivate();    // Legal because setPrivate() is protected.
    setPrivate2(0);          // Legal because setPrivate2() is public.
    int x = getPrivate2();   // Legal because setPrivate2() is public.
  }
}
```

6. Inheritance :: Accessibility Specifiers and Instance Methods (continued)

```
public class C {
  private Super super;  // This is a composition relationship.
  private Sub sub;      // This is also composition relationship.
  public C() {
    super = new Super();
    super.mPublic = 0;            // Legal: mPublic is public.
    super.mProtected = 0;         // Illegal: C is not a subclass of Super.
    super.mPrivate = 0;           // Illegal: mPrivate is private.
    super.setPrivate(0);          // Illegal: setPrivate() is protected.
    int y = super.getPrivate();   // Illegal: getPrivate() is protected.
    super.setPrivate2(0);         // Legal: setPrivate2() is public.
    int x = super.getPrivate2();  // Legal: getPrivate2() is public.
    sub = new Sub();
    sub.mPublic = 0;              // Legal: mPublic is public.
    sub.mProtected = 0;           // Illegal: mProtected is protected.
    sub.mPrivate = 0;             // Illegal: mPrivate is private.
    sub.setPrivate(0);            // Illegal: setPrivate() is protected.
    int y = sub.getPrivate();     // Illegal: getPrivate() is protected.
    sub.setPrivate2(0);           // Legal: setPrivate2() is public.
    int x = sub.getPrivate2();    // Legal: getPrivate2() is public.
  }
}
```