Java Tips (/web/20170106040426/http://www.java-tips.org/)

Covariant return types

This Tech Tip reprinted with permission by java.sun.com (https://web.archive.org/web/20170106040426/http://java.sun.com/)

You cannot have two methods in the same class with signatures that only differ by return type. Until the J2SE 5.0 release, it was also true that a class could not override the return type of the methods it inherits from a superclass. In this tip you will learn about a new feature in J2SE 5.0 that allows covariant return types. What this means is that a method in a subclass may return an object whose type is a subclass of the type returned by the method with the same signature in the superclass. This feature removes the need for excessive type checking and casting.

Let's start with the following class, ConfusedClass. The class tries to declare two methods with the same signature. One of the methods returns a JTextField, and the other returns a JPasswordField.

```
 import javax.swing.JTextField;
import javax.swing.JPasswordField;

public class ConfusedClass {

  public JTextField getTextField(){
    return new JTextField();
  }

  public JPasswordField getTextField(){
    return new JPasswordField();
  }
}
```

If you try to compile ConfusedClass, you get the following compile error:

```
ConfusedClass.java:10: getTextField() is already defined in
 ConfusedClass
   public JPasswordField getTextField(){
                               ^
 1 error
```

Looking at this situation from the perspective of a class calling getTextField(), you can see the reason for the compile time error. How would you indicate which of the two methods you are targeting? Consider, for example, this snippet:

```
ConfusedClass cc = new ConfusedClass();
 JTextField field = cc.getTextField();
```

Because a JPasswordField extends JTextField, either version of the method could correctly be called.

Next, create two classes, each of which having a different version of the getTextField() methods. The two methods differ by implementation and return type. Start with the following base:

```
import javax.swing.JTextField;

  public class ConfusedSuperClass {

    public JTextField getTextField(){
      System.out.println("Called in " + this.getClass());
      return new JTextField();
    }

  }
```

Compile ConfusedSuperClass. You'll see that it compiles without error. Now create a derived class, one that extends ConfusedSuperClass. The derived class attempts to return an instance of JPasswordField instead of the

Main Menu

Home (/web/20170106040426/http://www.java-tips.org/)
java.lang (/web/20170106040426/http://www.java-tips.org/java-lang.html)

Old Menu

Java Tutorials (/web/20170106040426/http://www.java-tips.org/java-tutorials.html)
Book Reviews (/web/20170106040426/http://www.java-tips.org/book-reviews.html)
Java SE Tips (/web/20170106040426/http://www.java-tips.org/java-se-tips.html)
Java ME Tips (/web/20170106040426/http://www.java-tips.org/java-me-tips.html)
Java EE Tips (/web/20170106040426/http://www.java-tips.org/java-ee-tips.html)
Other API Tips (/web/20170106040426/http://www.java-tips.org/other-api-tips.html)
Java Applications (/web/20170106040426/http://www.java-tips.org/java-applications.html)
Java Libraries (/web/20170106040426/http://www.java-tips.org/java-libraries.html)

Java Network

Java Forums (https://web.archive.org/web/20170106040426/http://www.java-forums.org/forum.php)
Java Blog (https://web.archive.org/web/20170106040426/http://www.java-forums.org/blogs/all/)

JTextField returned by the getTextField() method in ConfusedSuperClass.

```
import javax.swing.JPasswordField;

 public class ConfusedSubClass extends ConfusedSuperClass {

   public JPasswordField getTextField(){
     System.out.println("Called in " + this.getClass());
     return new JPasswordField();
   }

 }
```

If you use a version of the JDK prior to J2SE 5.0, ConfusedSubClass will not compile. You will see an error like this.

```
ConfusedSubClass.java:5: getTextField() in ConfusedSubClass
   cannot override getTextField() in ConfusedSuperClass;
   attempting to use incompatible return type
   found   : javax.swing.JPasswordField
   required: javax.swing.JTextField
     public JPasswordField getTextField(){
                                ^
   1 error
```

The error reported is that you are attempting to use an incompatible return type. In fact, the JPasswordField you are attempting to return is a subtype of JTextField. This same code compiles correctly under J2SE 5.0. You are now allowed to override the return type of a method with a subtype of the original type. In the current example, the getTextField() method in ConfusedSuperClass returns an instance of type JTextField. The getTextField() method in the ConfusedSubClass returns an instance of type JPasswordField.

You can exercise these two classes with the following NotConfusedClient class. This class creates an instance of type ConfusedSuperClass and of type ConfusedSubClass. It then calls getTextField() on each instance, and displays the type corresponding to the object returned by the method.

```
import javax.swing.JTextField;

  public class NotConfusedClient {
    static JTextField jTextField;

    public static void main(String[] args) {
      System.out.println("===== Super Class =====");
      jTextField = new ConfusedSuperClass().getTextField();
      System.out.println("Got back an instance of "
        + jTextField.getClass());
      System.out.println("===== Sub Class =====");
      jTextField = new ConfusedSubClass().getTextField();
      System.out.println("Got back an instance of  "
        + jTextField.getClass());
    }
  }
```

Compile and run NotConfusedClient. When you run it, you should see the following output:

```
 ===== Super Class =====
   Called in class ConfusedSuperClass
   Got back an instance of class javax.swing.JTextField
   ===== Sub Class =====
   Called in class ConfusedSubClass
   Got back an instance of  class javax.swing.JPasswordField
```

In fact, you will get the same output if you change the return type of getTextField() to JTextField in ConfusedSubClass. The payoff comes when you use the object that is returned by the call to getTextField(). Before J2SE 5.0, you needed to downcast to take advantage of methods that are present in the derived class but not in the base class. As you've seen, in J2SE 5.0 ConfusedSubClass compiles with a different return type specified for getTextField() than is present in the superclass. You can now use your covariant return type to call a method that is only available in the subtype. First, recast the supertype like this:

```
public class SuperClass {

    public SuperClass getAnObject(){
      return this;
    }

  }
```

Add the exclusive method to the corresponding subclass, and change the return type of the getAnObject() method:

```
public class SubClass extends SuperClass {

    public SubClass getAnObject(){
      return this;
    }

    public void exclusiveMethod(){
      System.out.println("Exclusive in Subclass.");
    }

    public static void main(String[] args) {
        System.out.println("===== Call Exclusive method =====");
        new SubClass().getAnObject().exclusiveMethod();
    }

  }
```

Compile SuperClass and SubClass, then run SubClass. You should see the following output:

```
===== Call Exclusive method =====
  Exclusive in Subclass.
```

The main() method creates an instance of the subclass. It then calls getAnObject(). It follows this with a call to exclusiveMethod() on the returned object of type SubClass. If the return type of getAnObject() was SuperClass in SubClass.java, the code would not have compiled.

Parent Category: Java SE Tips (/web/20170106040426/http://www.java-tips.org/java-se-tips-100019.html)

Back to Top