Supplement: Hiding Data Fields and Static Methods
For Introduction to Java Programming
By Y. Daniel Liang

You can override an instance method, but you cannot override a data field (instance or static) or a static method. If you declare a data field or a static method in a subclass with the same name as one in the superclass, the one in the superclass is hidden, but it still exists. The two data fields or static methods are independent. You can reference the hidden data field or static method using the keyword **super** in the subclass. The hidden field or method can also be accessed via a reference variable of the superclass's type.

When invoking an instance method from a reference variable, the *actual class of the object* referenced by the variable decides which implementation of the method is used *at runtime*. When accessing a data field or a static method, the *declared type* of the reference variable decides which field or static method is used *at compile time*. This is the key difference between invoking an instance method and accessing a data field or a static method.

Listing 1 demonstrates the effect of hiding data fields and static methods.

Listing 1 HidingDemo.java

```
public class HidingDemo {
  public static void main(String[] args) {
    A x = new B();

    // Access instance data field i
    System.out.println("(1) x.i is " + x.i);
    System.out.println("(2) (B)x.i is " + ((B)x).i);

    // Access static data field j
    System.out.println("(3) x.j is " + x.j);
    System.out.println("(4) ((B)x).j is " + ((B)x).j);

    // Invoke static method m1
    System.out.println("(5) x.m1() is " + x.m1());
    System.out.println("(6) ((B)x).m1() is " + ((B)x).m1());

    // Invoke instance method m2
    System.out.println("(7) x.m2() is " + x.m2());
    System.out.println("(8) x.m3() is " + x.m3());
  }
}
```


© Copyright Y. Daniel Liang, 2005

```
class A {
  public int i = 1;
  public static int j = 11;

  public static String m1() {
    return "A's static m1";
  }

  public String m2() {
    return "A's instance m2";
  }

  public String m3() {
    return "A's instance m3";
  }
}

class B extends A {
  public int i = 2;
  public static int j = 12;

  public static String m1() {
    return "B's static m1";
  }

  public String m2() {
    return "B's instance m2";
  }
}
```

Sample Output

```
(1) x.i is 1
(2) (B)x.i is 2
(3) x.j is 11
(4) ((B)x).j is 12
(5) x.m1() is A's static m1
(6) ((B)x).m1() is B's static m1
(7) x.m2() is B's instance m2
(8) x.m3() is A's instance m3
```

Here are the explanations:

(1) **x.i** is 1 because **x**'s declared type is the class **A**.

(2) To use **i** in the class **B**, you need to cast **x** to **B** using **((B)x).i**.

(3) **x.j** is 11 because **x**'s declared type is the class **A**. **x.j** is better written as **A.j**.

(4) To use **j** in the class **B**, you need to cast **x** to **B** using **((B)x).j**. **((B)x).j** is better written as **B.j**.

© Copyright Y. Daniel Liang, 2005

(5) **x.m1()** invokes the static **m1** method in **A** because **x**'s declared type is **A**. **x.m1()** is better written as **A.m1()**.

(6) **((B)x).m1()** invokes the static **m1** method in **B** because the type for **(B)x** is **B**. **((B)x).m1()**is better written as **B.m1()**.

(7) **x.m2()** invokes the **m2** method in **B** at runtime because **x** actually references to the object of the class **B**.

(8) **x.m3()** invokes the **m3** method in **A** at runtime because **m3** is implemented in **A**.

NOTE

A static method or a static field can always be accessed using its declared class name, regardless of whether it is hidden or not.

© Copyright Y. Daniel Liang, 2005