Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Function Overriding/5b1e57047becc0459deae136

TUTORIAL

Java : Function Overriding

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden. Consider the following:

Java

```
// Method overriding.
class A {
  int i, j;
  A(int a, int b) {
    i = a;
    j = b;
  }
  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}
class B extends A {
  int k;
  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }
  // display k - this overrides show() in A
  void show() {
    System.out.println("k: " + k);
  }
```

```
}
class Main {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);
    subOb.show(); // this calls show() in B
  }
}

```

The output produced by this program is shown here:

```
k: 3
```

When show( ) is invoked on an object of type B, the version of show( ) defined within B is used. That is, the version of show( ) inside B overrides the version declared in A.

If you wish to access the superclass version of an overridden function, you can do so by using super. For example, in this version of B, the superclass version of show( ) is invoked within the subclass' version. This allows all instance variables to be displayed.

```
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
super.show(); // this calls A's show()
System.out.println("k: " + k);
}
}
```

If you substitute this version of A into the previous program, you will see the following output:

```
i and j: 1 2
k: 3
```

Here, super.show( ) calls the superclass version of show( ).

Method overriding occurs only when the names and the type signatures of the two methods are identical. If they are not, then the two methods are simply overloaded. For example, consider this modified version of the preceding example:

Java

```
// Methods with differing type signatures are overloaded - not overridden.
class A {
  int i, j;
  A(int a, int b) {
    i = a;
    j = b;
  }
  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}
// Create a subclass by extending class A.
class B extends A {
  int k;
  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }
  // overload show()
  void show(String msg) {
    System.out.println(msg + k);
  }
}
class Main {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);
    subOb.show("This is k: "); // this calls show() in B
    subOb.show(); // this calls show() in A
  }
}

```

The output produced by this program is shown here:

```
This is k: 3
i and j: 1 2
```

The version of show( ) in B takes a string parameter. This makes its type signature different from the one in A, which takes no parameters. Therefore, no overriding (or name hiding) takes place.