Overriding

MAY 29, 2015

Reimplementing the inherited method from the parent class in the child class is called **Overriding in Java**.

There are certain **rules for overriding**, the below code points out all of the rules:

```java
class Animal {

    private void drink() {
        System.out.println("Animal Drink");
    }

    public void eat() {
        System.out.println("Animal Eat");
    }

    protected void walk() {
        System.out.println("Animal Walk");
    }

    public void run() {
        System.out.println("Animal Run");
    }

    public void sleep() throws IOException {
        System.out.println("Animal Sleep");
    }

    public Animal getAnimal() {
        return new Animal();
    }
}

class Horse extends Animal {

    private void drink() {  // Not a method override as drink()
                            // wasn't inherited by Horse class
        System.out.println("Horse Drink");
    }

    private void eat() {    // You can't use a more
                            // restrictive access modifier
                            // (gives you a compiler error)
        System.out.println("Horse Eat");
    }

    public void walk() {    // Valid method override as you can use less restrictive
                            // access modifier in the overriding method
        System.out.println("Horse Walk");

    }

    public void run(int n) {    // Not a method override (argument list differs)
                                // but rather a method overload of run() in Animal
        System.out.println("Horse Run");
    }

    public Horse getAnimal() {  // Valid method override as return type must
                                // be the same as, or a subtype of, the return type
                                // declared in the original overridden method in the supercl
        return new Horse();
    }
```

```java

    public void eat() throws Exception {   // Invalid method override as the overridden meth
                                           // throw any checked exceptions while overriding
                                           // (gives a compiler error)
        System.out.println("Horse Eat");
    }

    public void sleep() throws FileSystemException {   // Valid method override as FileSyste
                                                       // is a subclass of IOException
        System.out.println("Horse Sleep");
    }

    public void sleep() {                        // Valid method override as it isn't mandat
                                                 // the overridden method to throw any excep
            System.out.println("Horse Sleep");
    }

    public void sleep() throws Exception {  // Invalid method override as Exception is neith
                                            // same as nor a subclass of IOException
                                            // (gives a compiler error)
        System.out.println("Horse Sleep");
    }
}

public class Overriding {

    public static void main (String [] args) {
        Animal a = new Animal();
        Animal b = new Horse();  // Animal ref, but a Horse object

        a.eat();   // Runs the Animal version of eat()
        b.eat();   // Runs the Horse version of eat()
                   // (Concept called Dynamic method invocation)

    }
}
```

Some more rules which may be obvious:

- You cannot override a method marked `final`.
- You cannot override a method marked `static`.
- If a method can't be inherited, you cannot override it. As said earlier, overriding implies that you're reimplementing a method you inherited.

Dynamic Method Invocation: Overridden instance methods are dynamically invoked based on the real object's type rather than the reference type. For example, `b.eat()` will actually run the Horse version of `eat()`.

Q&A

Q1. Will the below code compile?

```java
class Animal {
    public void eat() throws Exception {
        // throws an Exception
    }
}

class Dog extends Animal {
    public void eat() { /* no Exceptions */}

    public static void main(String[] args) {
        Animal a = new Dog();
        Dog d = new Dog();
        d.eat();
        a.eat();
    }
}
```

Object oriented design

« Previous

Overloading

Next »

0 Comments Java Notes 1 Login

Recommend Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON JAVA NOTES

Overloading | Java Concepts
1 comment • 3 years ago
Anagh Hegde — public class MyTest { public static void main(String[] args) { MyTest test = new MyTest(); int i = 9; test.TestOverLoad(i); } …

Nested Classes | Java Notes
2 comments • 3 years ago
Amit Satpathy — Ram, it's really great to find your short and precise writeups on tech. Keep up the passion.