Quiz Yourself: Inner, Nested, and Anonymous Classes (Advanced)

Java’s mechanisms and syntax for defining classes inside other classes

by Simon Roberts and Mikalai Zaikin

November 21, 2019

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. The "intermediate" and "advanced" designations refer to the exams, rather than to the questions, although in almost all cases, "advanced" questions will be harder. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value, and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

Given the following class:

```java
public class TextEditor {
    static class Char { // line 1
        char c;
        public Char(char c) {
            this.c = c;
        }
        public void print() {
            System.out.print(c);
        }
    }
    public static void main(String[] args) {
        typeText(new Char('A')); // line 2
        typeText(new Char('B')) { // line 2
            public void print() {
                System.out.print("");
                super.print(); // line 3
                System.out.print("");
            }
        });
        private static void typeText(Char c) {
            c.print();
        }
    }
}
```

**What is the result?** Choose one.

A. Compilation fails at line 1.
B. Compilation fails at line 2.
C. Compilation fails at line 3.
D. A[A].
E. A[B].

**Answer:** This question investigates several aspects of Java’s mechanisms for defining classes inside other classes. These include
Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance Java allows types (that is, classes and interfaces) to be defined at the top level of a source file, and this is the most familiar form. However, you can also define types inside other types. One powerful reason for doing this is to grant controlled access to private members of one class to another without breaking the overall encapsulation.

When a field or method is defined in a class, it can optionally carry the modifier static. The effect of this modifier is to associate the element (that is, the field or method) with the class as a whole rather than with a particular instance of the class. The same choice exists for classes defined inside other classes. If a class is defined inside another class and is marked static, it is known as a nested class. If it is not marked static, it's called an inner class.

For a nested (that is, static) class, the special relationship is simply that the two classes have direct access to one another’s members, even if those members are private. However, this access is possible only through explicit object references; one object's this reference does not give any access to an object of the other class. Some additional access of this second kind is possible in the case of non-static inner classes, but that is not the topic of this question.

Now, let's look at the syntax for declaring a nested class: If you simply take a regular class, move it inside another class (but not inside a method), and add the static modifier, then you have a simple nested class. Based on this, you can determine that the syntax used to declare the class Char is valid and, therefore, option A is incorrect.

Inside the main method, two instances of the class Char are created. First an instance is created directly using the conventional syntax new Char('A'). This syntax is valid for this nested class, although an inner class that's not marked static could not be instantiated this way from a static context such as this main method. But in this case, it's completely correct.

The second instantiation defines and instantiates an anonymous inner class instance. In this case, the syntax is valid and the resulting object is an instance of a subclass of Char. Also, the argument 'B' is passed to the constructor of the base class Char, so the construction is also valid. Further, this anonymous class overrides the print method of the parent Char class. Because this code is valid, you can conclude that option B is incorrect.

Given that the definition of the anonymous subclass is valid, it should be no surprise that you can invoke methods of the superclass from the anonymous subclass using the super prefix. So, line 3 compiles correctly, and option C is also incorrect.

When executed, the code will first print A. Then the anonymous inner class will print [ and then call the print method of the superclass Char, which outputs B, and after the superclass print method is complete, the anonymous class completes its print method by printing ]. Therefore the output will be A[B].

So you can conclude that option E is correct, and option D is incorrect.

The correct answer is option E.
Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

---

**Share this Page**

Facebook  Twitter  LinkedIn  Email

---

Contact

US Sales: +1.800.633.0738
Global Contacts
Support Directory
Subscribe to Emails

About Us

Careers
Communities
Company Information
Social Responsibility Emails

Downloads and Trials

Java for Developers
Java Runtime Download
Software Downloads
Try Oracle Cloud

News and Events

Acquisitions
Blogs
Events
Newsroom

---

Oracle Integrated Cloud
Applications & Platform Services

© Oracle | Site Map | Terms of Use & Privacy | Cookie Preferences | Ad Choices