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 WeirdLoop {
    private static final int i = 99;
    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) { // line n1
            System.out.print(i);
            i++; // line n2
            break;
        }
        System.out.print(i);
    }
}
```

What is the result? Choose one.

A. Compilation fails at line n1.
B. Compilation fails at line n2.
C. 099
D. 9999
E. 99100

Answer: This question addresses issues of the naming and scope of variables and also the meaning of the keyword `final` when it is applied to primitive variables.

The first observation is that there are two distinct variables called `i` that are potentially relevant at line n1. One is the `static final` field declared just above the `main` method. The other is the method local variable declared in the `for` loop that starts on line n1.

In Java, an unqualified variable name will be resolved by first looking for an in-scope local variable of that name. If no such variable exists, then the compiler proceeds by looking for an element of that name either as a static or instance member. Order is not relevant for this, because only one or the other can succeed in the current class. If nothing is found in the current class, the parent classes are searched in order by working up the hierarchy for either static or instance variables.
The scope of a local variable starts with its declaration and generally ends at the end of the curly-brace-bounded block that contains it. If the variable is declared in a for loop (as in line n1 in this question), it remains in scope until the end of the code that iterates under the control of the for loop (and remember that, from a syntax perspective, a for loop doesn’t always have to have a block following it). Similar variations apply to method, constructor, and lambda argument lists as well as to catch blocks.

In this case, the method local version of i that is declared in the for loop is in scope for all the unqualified uses of the identifier i that are present in the for loop, because those uses all fall in the same scope and come after the initial declaration of i in the loop. After the for loop ends, that local i is out of scope and the field is once again the target of the unqualified identifier. Because this is all valid, it causes no compilation errors. Therefore option A is incorrect.

Notice however, that a local variable may not shadow another local variable in the same method, so the following code, which attempts to define two variables called i in the same method, would not compile:

```java
public static void main(String[] args) {
    int i = 99;
    for (int i = 0; i < 100; i++) { // FAILS !
        ...
    }
}
```

This is true even if the second variable were declared in a nested block, like this:

```java
public static void main(String[] args) {
    int i = 99;
    {
        int i = 0; // Also fails!
    }
}
```

Given the preceding discussion, you know that the meaning of the variable name i inside the loop block is the local variable i. This variable is not final, unlike the static one declared at the class level. Given that i++ modifies a nonfinal variable, the increment at line n2 is entirely correct and compiles successfully. This tells you that option B is also incorrect.

Of course, if you were to try to refer to the static final directly, it would be a compile time error to attempt the increment. So, if you modified the loop body like the following, it would not compile:

```java
for (int i = 0; i < 100; i++) { // line 1
    System.out.print(i);
    WeirdLoop.i++; // FAILS!
 ...
}
```

Now that you know the code is valid and compiles without error, turn your attention to what the output is.

Inside the loop that starts on line n1, the unqualified variable i refers to the value that’s declared and initialized—to zero—by the loop itself. Because of this, the first print statement will produce 0, and that variable will increment to 1. However, immediately after the increment occurs, the break statement exits the loop structure. Now, the scope of the loop-declared variable i starts at its declaration but ends at the end of the loop’s body; hence, exiting the loop structure also exits the scope of that variable. Therefore, that cannot be the value printed after the loop. This doesn’t break the code however, because the static final i class member is in scope, and now that it’s not shadowed, it provides the
value that is printed. Because the value 99 was assigned when the class
was loaded and initialized, and the value cannot change because it is
final, this second print statement will output 99.

Therefore, the program prints 099, and that tells you that option C is
correct.

Option D cannot be correct as the question is written, but it would be
correct if the code explicitly referred to the static class member in the first
print statement instead, like this:

```java
for (int i = 0; i < 100; i++) {
    System.out.print(WeirdLoop.i);
    i++;
    break;
} System.out.print(i);
```

Option E never would be possible as the final primitive variable may
not change value, so the second print always produces 99.

The correct answer is option C.

**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
educator who publishes recorded and live video
training through Pearson InformIT (available direct
and through the O’Reilly Safari Books Online
service). He remains involved with Oracle’s Java
certification projects.

**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.