Chapter1

Monday, October 6, 2025 11:33 AM

```
System.out.println(Integer.valueOf("5", 16)); // 5
System.out.println(Integer.valueOf("a", 16)); // 10
System.out.println(Integer.valueOf("F", 16)); // 15
System.out.println(Integer.valueOf("G", 16)); // NumberFormatExce
```

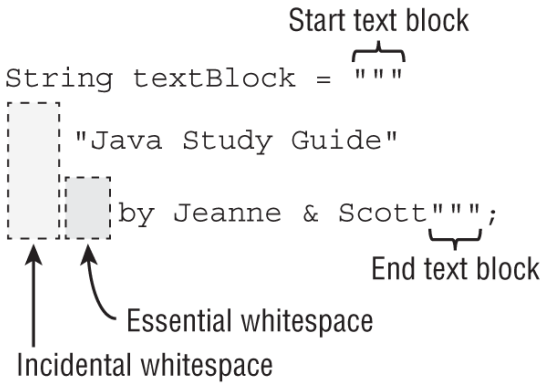

FIGURE 1.3 Text block

OPTIONAL MODIFIERS IN *MAIN()* METHODS

While most modifiers, such as `public` and `static`, are required for `main()` methods, there are some optional modifiers allowed.

```
public final static void main(final String[] args) {}
```

In this example, both `final` modifiers are optional, and the `main()` method is a valid entry point with or without them. We cover the meaning of `final` methods and parameters in Chapter 6.

Keyword	Type	Min value	Max value	Default value	Example
`boolean`	`true` or `false`	n/a	n/a	`false`	`true`
`byte`	8-bit integral value	-128	127	`0`	`123`
`short`	16-bit integral value	-32,768	32,767	`0`	`123`
`int`	32-bit integral value	-2,147,483,648	2,147,483,647	`0`	`123`
`long`	64-bit integral value	-2^{63}	$2^{63}-1$	`0L`	`123L`
`float`	32-bit floating-point value	n/a	n/a	`0.0f`	`123.45f`
`double`	64-bit floating-point value	n/a	n/a	`0.0`	`123.456`
`char`	16-bit Unicode value	0	65,535	`\u0000`	`'a'`

```
String block = """
   "doe\"\"\"
   \"deer\"""
  """;
System.out.println("*"+ block  + "*");
```

The answer is:

```
* "doe"""
 "deer"""
*
```

All of the `\"` escape the `"`. There is one space of essential whitespace on the `doe` and `deer` lines. All the other leading whitespace is incidental whitespace.

TRICKY things

This is where it gets tricky. Pay attention to tricky things! The exam will attempt to trick you. Again, how many variables do you think are declared and initialized in the following code?

```
void paintFence() {
   int i1, i2, i3 = 0;
}
```

As you should expect, three variables were declared: `i1`, `i2`, and `i3`. However, only one of those values was initialized: `i3`. The other two remain declared but not yet initialized. That's the trick. Each snippet separated by a comma is a little declaration of its own. The initialization of `i3` only applies to `i3`. It doesn't have anything to do with `i1` or `i2` despite being in the same statement. As you will see in the next section, you can't actually use `i1` or `i2` until they have been initialized.

Another way the exam could try to trick you is to show you code like this line:

```
int num, String value; // DOES NOT COMPILE
```

```
7:  public void breakingDeclaration() {
8:     var silly
9:        = 1;
10: }
```

This example is valid and does compile, but we consider the declaration and initialization of `silly` to be happening on the same line.

Let's try another example. Do you see why this does not compile?

```
public int addition(var a, var b) {  // DOES NOT COMPILE
   return a + b;
}
```

In this example, `a` and `b` are method parameters. These are not local variables. Be on the lookout for `var` used with constructor parameters, method parameters, or instance variables. Using `var` in one of these places is a good exam trick to see if you are paying attention. Remember that `var` is used only for local variable type inference!

There's one last rule you should be aware of: `var` is not a reserved word and allowed to be used as an identifier. It is considered a reserved type name. A *reserved type name* means it cannot be used to define a type, such as a class, interface, or `enum`. Do you think this is legal?

```
package var;

public class Var {
   public void var() {
      var var = "var";
   }
   public void Var() {
      Var var = new Var();
   }
}
```

Believe it or not, this code does compile. Java is case sensitive, so `Var` doesn't introduce any conflicts as a class name. Naming a local variable `var` is legal. Please don't write code that looks like this at your job! But understanding why it works will help get you ready for any tricky exam questions the exam creators could throw at you.

```
4: boolean b1, b2;
5: String s1 = "1", s2;
6: double d1, double d2;
7: int i1; int i2;
8: int i3; i4;
```

VAR

```
4: public void twoTypes() {
5:    int a, var b = 3;  // DOES NOT COMPILE
6:    var a, b = 3;      // DOES NOT COMPILE
7:    var n = null;      // DOES NOT COMPILE
8: }
```

Line 5 wouldn't work even if you replaced `var` with a real type. All the types declared on a single line must be the same type and share the same declaration. We couldn't write `int a, int b = 3;` either. Line 6 shows that you can't use `var` to define two variables on the same line.

Line 7 is a single line. The compiler is being asked to infer the type of `null`. This could be any reference type. The only choice the compiler could make is `Object`. However, that is almost certainly not what the author of the code intended. The designers of Java decided it would be better not to allow `var` for `null` than to have to guess at intent.

While a `var` cannot be initialized with a `null` value without a type, it can be reassigned a `null` value after it is declared, provided that the underlying data type is a reference type.

```
public class Zoo {
   public void whatTypeAmI() {
      var name = "Hello";
      var size = 7;
   }
}
```

The formal name of this feature is *local variable type inference*. Let's take that apart. First comes *local variable*. This means just what it sounds like. You can use this feature only for local variables. The exam may try to trick you with code like this:

```
public class VarKeyword {
   var tricky = "Hello"; // DOES NOT COMPILE
}
```

Wait a minute! We just learned the difference between instance and local variables. The variable `tricky` is an instance variable. Local variable type inference works with local variables and not instance variables.

Scopes

Scopes

Reviewing Scope

Got all that? Let's review the rules on scope.

- *Local variables*: In scope from declaration to the end of the block
- *Method parameters*: In scope for the duration of the method
- *Instance variables*: In scope from declaration until the object is eligible for garbage collection
- *Class variables*: In scope from declaration until the program ends

GC

Java includes a built-in method to help support garbage collection where you can suggest that garbage collection run.

```
System.gc();
```

Just like the post office, Java is free to ignore you. This method is not *guaranteed* to do anything.

White spaces doesn't count on Java Syntax

compiles when dropped into valid surrounding code. Finally, remember that extra whitespace doesn't matter in Java syntax. The exam may use varying amounts of whitespace to trick you.

EXAM

Responses:

1. E
2. C
3. A,E
4. C,F
5. A,E
6. G
7. D,E
8. B,D,G,H
9. A,E
10. A,E,F
11. D
12. A
13. ?
14. A,D,E
15. B,C,F
16. D
17. B,D,F,H
18. B,D,F,G
19. A,D,F
20. E
21. C
22. A,C,F,G
23. A,D

1. -> D :
   E:

1. Which of the following are legal entry point methods that can be run from the command line? (Choose all that apply.)
   A. `private static void main(String[] args)`
   B. `public static final main(String[] args)`
   C. `public void main(String[] args)`
   D. `public static final void main(String[] args)`
   E. `public static void main(String[] args)`
   F. `public static main(String[] args)`

1. D, E. Option E is the canonical `main()` method signature. You need to memorize it. Option D is an alternate form with the redundant `final`. Option A is incorrect because the `main()` method must be public. Options B and F are incorrect because the `main()` method must have a `void` return type. Option C is incorrect because the `main()` method must be `static`.

2. -> C:
   D:
   E:

2. Which answer options represent the order in which the following statements can be assembled into a program that will compile successfully? (Choose all that apply.)

```
X: class Rabbit {}
Y: import java.util.*;
Z: package animals;
```

   A. `X`, `Y`, `Z`
   B. `Y`, `Z`, `X`
   C. `Z`, `Y`, `X`
   D. `Y`, `X`
   E. `Z`, `X`
   F. `X`, `Z`
   G. None of the above

2. C, D, E. The `package` and `import` statements are both optional. If both are present, the order must be `package`, then `import`, and then `class`. Option A is incorrect because `class` is before `package` and `import`. Option B is incorrect because `import` is before `package`. Option F is incorrect because `class` is before `package`.

3. A, E
4. B, E, G.

4. Which of the following are valid Java identifiers? (Choose all that apply.)
   A. `_`
   B. `_helloWorld$`
   C. `true`
   D. `java.lang`
   E. `Public`
   F. `1980_s`
   G. `_Q2_`

4. B, E, G. Option A is invalid because a single underscore is not allowed. Option C is not a valid identifier because `true` is a Java reserved word. Option D is not valid because a period (`.`) is not allowed in identifiers. Option F is not valid because the first character is not a letter, dollar sign (`$`), or underscore (`_`). Options B, E, and G are valid because they contain only valid characters.

5. A, D, F.

5. Which statements about the following program are correct? (Choose all that apply.)

```
2:  public class Bear {
3:     private Bear pandaBear;
4:     private void roar(Bear b) {
5:        System.out.println("Roar!");
6:        pandaBear = b;
7:     }
8:     public static void main(String[] args) {
9:        Bear brownBear = new Bear();
10:       Bear polarBear = new Bear();
11:       brownBear.roar(polarBear);
12:       polarBear = null;
13:       brownBear = null;
14:       System.gc(); } }
```

   A. The object created on line 9 is first eligible for garbage collection after line 13.
   B. The object created on line 9 is first eligible for garbage collection after line 14.
   C. The object created on line 10 is first eligible for garbage collection after line 12.
   D. The object created on line 10 is first eligible for garbage collection after line 13.
   E. Garbage collection is guaranteed to run.
   F. Garbage collection might or might not run.
   G. The code does not compile.

5. A, D, F. Garbage collection is never guaranteed to run, making option F correct and option E incorrect. Next, the class compiles and runs without issue, so option G is incorrect. The `Bear` object created on line 9 is accessible until line 13 via the `brownBear` reference variable, which is option A. The `Bear` object created on line 10 is accessible via both the `polarBear` reference and the `brownBear.pandaBear` reference. After line 12, the object is still accessible via `brownBear.pandaBear`. After line 13, though, it is no longer accessible since `brownBear` is no longer accessible, which makes option D the final correct answer.

6. F.

6. Assuming the following class compiles, how many variables defined in the class or method are in scope on the line marked on line 14?

```
1:  public class Camel {
2:     { int hairs = 3_000_0; }
3:     long water, air=2;
4:     boolean twoHumps = true;
5:     public void spit(float distance) {
6:        var path = "";
7:        { double teeth = 32 + distance++; }
8:        while(water> 0) {
9:           int age = twoHumps ? 1 : 2;
10:          short i=-1;
11:          for(i=0; i<10; i++) {
12:             var Private = 2;
13:          }
14:          // SCOPE
15:       }
16:    }
17: }
```

A. 2
B. 3
C. 4
D. 5
E. 6
F. 7
G. None of the above

7. C,E
8. !!!

B, D, E, H

is correct as this is a valid numeric expression. You might know that dividing by zero produces a runtime exception, but the question was only about whether the code compiled. Finally, options F and G are incorrect as `var` cannot be used in a multiple-variable assignment.

9. E

default values. Option A is incorrect because `float` should have a decimal point. Option B is incorrect because primitives do not de-

11. E

because `Tank` and `Water` are in the same package, making the correct option E. If `Tank` and `Water` were in different packages,

12. A,C,D

Summary

Context	Value	Type	Notes
Default float value	`0.0`	`float`	JVM assigns this internally to float vars
Float literal in code	`0f`	`float`	Must use `f` to avoid being treated as double
Double literal in code	`0.0`	`double`	Default type for decimal literals

So yes — you're absolutely right that `0.0` is a `double` literal, and `0f` is a `float` literal. But the JVM uses `0.0` as the default value for both `float` and `double`, depending on the variable type.