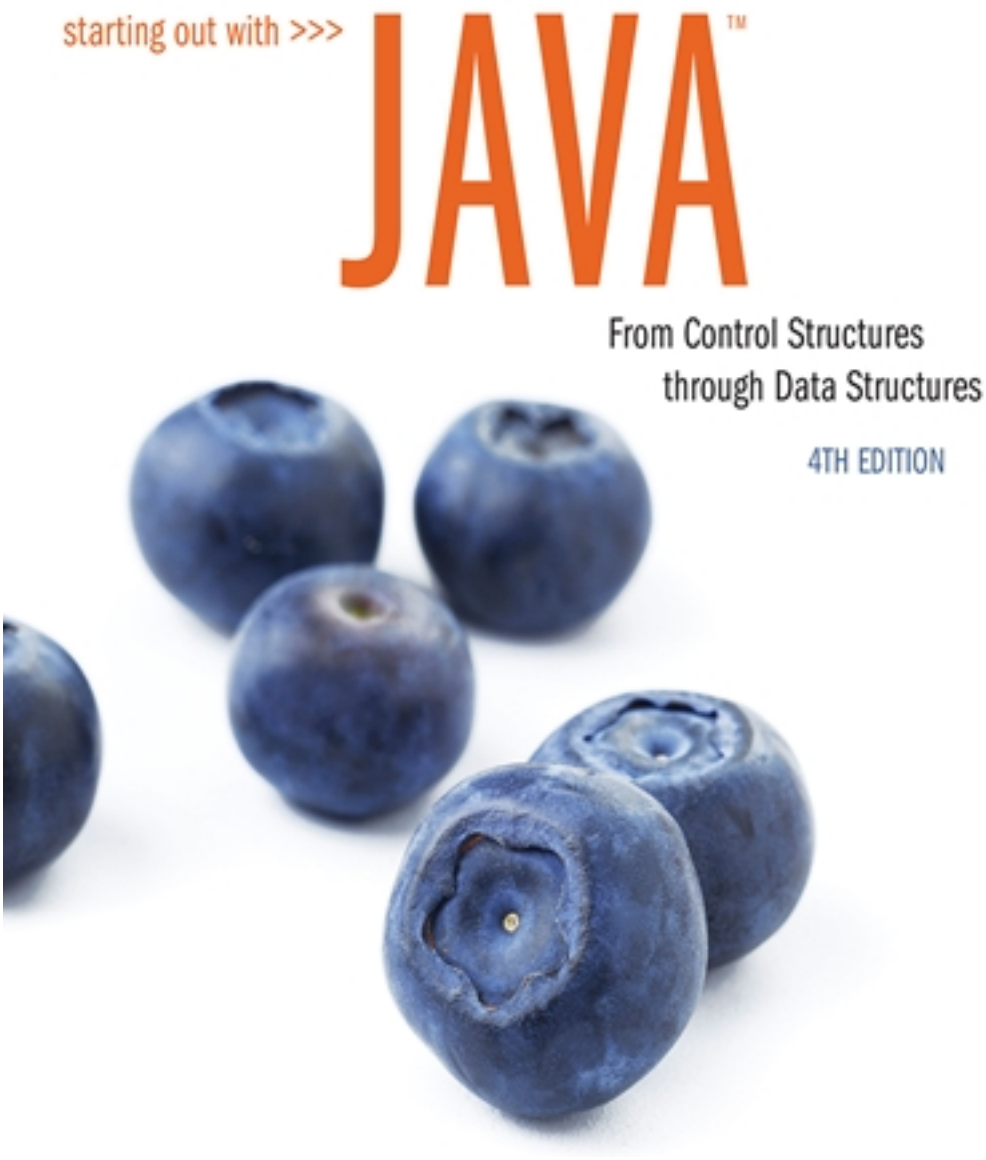
Solutions for Starting Out with Java From Control Structures through Data Structures 4th Edition by Gaddis

CLICK HERE TO ACCESS COMPLETE Solutions



Solutions

Starting Out with Java: From Control Structures through Data Structures
Answers to Review Questions

Chapter 2

Multiple Choice and True/False

1. c
2. b
3. a
4. b and c
5. a, c, and d
6. a
7. c
8. b
9. a
10. d
11. b
12. a
13. a
14. c
15. a
16. True
17. True
18. False
19. True
20. False
21. False

Predict the Output

1.
```
0
100
```
2.
```
8
2
```
3.
```
I am the incrediblecomputing
machine
and I will
amaze
you.
```
4.
```
Be careful
This might/n be a trick question.
```
5.
```
23
1
```

Find the Error

- The comment symbols in the first line are reversed. They should be `/*` and `*/`.

- The word `class` is missing in the second line. It should read `public class MyProgram`.
- The `main` header should not be terminated with a semicolon.
- The fifth line should have a left brace, not a right brace.
- The first four lines inside the `main` method are missing their semicolons.
- The comment in the first line inside the `main` method should begin with forward slashes (`//`), not backward slashes.
- The last line inside the `main` method, a call to `println`, uses a string literal, but the literal is enclosed in single quotes. It should be enclosed in double quotes, like this: `"The value of c is"`.
- The last line inside the `main` method passes `C` to `println`, but it should pass `c` (lowercase).
- The class is missing its closing brace.

Algorithm Workbench

1.
```
double temp, weight, age;
```
2.
```
int months = 2, days, years = 3;
```
3.
   a)
```
b = a + 2;
```
   b)
```
a = b * 4;
```
   c)
```
b = a / 3.14;
```
   d)
```
a = b – 8;
```
   e)
```
c = 'K';
```
   f)
```
c = 66;
```
4.
   a) 12
   b) 4
   c) 4
   d) 6
   e) 1
5.
   a) 3.287E6
   b) -9.7865E12
   c) 7.65491E-3
6.
```
System.out.print("Hearing in the distance\n\n\n");
System.out.print("Two mandolins like creatures in the\n\n\n");
System.out.print("dark\n\n\n");
System.out.print("Creating the agony of ecstasy.\n\n\n");
System.out.println(" - George Barker");
```
7.
```
10 20 1
```
8.
```
12
```
9.
```
a
```
10.
```
HAVE A GREAT DAY!
Have a great day!
```

11.
```
int speed, time, distance;
speed = 20;
time = 10;
distanct = speed * time;
System.out.println(distance);
```
12.
```
double force, area, pressure;
force = 172.5;
area = 27.5;
pressure = area / force;
System.out.println(pressure);
```

13.
```
double income;
// Create a Scanner object for keyboard input.
Scanner keyboard = new Scanner(System.in);
// Ask the user to enter his or her desired income
System.out.print("Enter your desired annual income: ");
income = keyboard.nextDouble();
```
14.
```
String str;
double income;
str = JOptionPane.showInputDialog("Enter your desired " +
                          "annual income.");
income = Double.parseDouble(str);
```

15. `total = (float)number;`

Short Answer

1. Multi-line style
2. Single line style
3. A self-documenting program is written in such a way that you get an understanding of what the program is doing just by reading its code.
4. Java is a case sensitive language, which means that it regards uppercase letters as being entirely different characters than their lowercase counterparts. This is important to know because some words in a Java program must be entirely in lowercase.
5. The `print` and `println` methods are members of the `out` object. The `out` object is a member of the `System` class. The `System` class is part of the Java API.
6. A variable declaration tells the compiler the variable's name and the type of data it will hold.
7. You should always choose names for your variables that give an indication of what they are used for. The rather nondescript name, `x`, gives no clue as to what the variable's purpose is.
8. It is important to select a data type that is appropriate for the type of data that your program will work with. Among the things to consider are the largest and smallest

possible values that might be stored in the variable, and whether the values will be whole numbers or fractional numbers.

9. In both cases you are storing a value in a variable. An assignment statement can appear anywhere in a program. An initialization, however, is part of a variable declaration.
10. Comments that start with `//` are single-line style comments. Everything appearing after the `//` characters, to the end of the line, is considered a comment. Comments that start with `/*` are multi-line style comments. Everything between these characters and the next set of `*/` characters is considered a comment. The comment can span multiple lines.
11. Programming style refers the way a programmer uses spaces, indentations, blank lines, and punctuation characters to visually arrange a program's source code. An inconsistent programming style can create confusion for a person reading the code.
12. One reason is that the name `PI` is more meaningful to a human reader than the number 3.14. Another reason is that any time the value that the constant represents needs to be changed, we merely have to change the constant's initialization value. We do not have to search through the program for each statement that uses the value.
13. `javadoc SalesAverage.java`
14. The result will be an `int`.

APPENDIX B

Operator Precedence and Associativity

This table shows the precedence and associativity of all the Java operators. The table is divided into groups, and each operator in a group has the same precedence. The groups of operators are arranged from the highest precedence at the top of the table to the lowest precedence at the bottom of the table. For example, the first group of operators shown is:

```
. [] () ++ --
```

This group of operators has the highest precedence of all the operators; and each of these operators has the same precedence.

Operator	Description	Associativity
`.`	membership	left-to-right
`[]`	array subscript	left-to-right
`()`	method argument list	left-to-right
`++`	postfix increment	left-to-right
`--`	postfix decrement	left-to-right
`++`	prefix increment	right-to-left
`--`	prefix decrement	right-to-left
`+`	unary plus	right-to-left
`-`	unary minus	right-to-left
`~`	bitwise complement	right-to-left
`!`	logical NOT	right-to-left
`new`	object creation	right-to-left
`(type)`	cast	right-to-left
`*`	multiplication	left-to-right
`/`	division	left-to-right
`%`	remainder	left-to-right

Operator	Description	Associativity
+	addition	left-to-right
+	string concatenation	left-to-right
-	subtraction	left-to-right
<<	left shift	left-to-right
>>	signed right shift	left-to-right
>>>	unsigned right shift	left-to-right
<	less than	left-to-right
>	greater than	left-to-right
<=	less than or equal to	left-to-right
>=	greater than or equal to	left-to-right
instanceof	type comparison	left-to-right
==	equal to	left-to-right
!=	not equal to	left-to-right
&	bitwise AND	left-to-right
^	bitwise XOR	left-to-right
|	bitwise OR	left-to-right
&&	logical AND	left-to-right
||	logical OR	left-to-right
?:	conditional	right-to-left
=	assignment	right-to-left
+=	combined assignment	right-to-left
-=	combined assignment	right-to-left
*=	combined assignment	right-to-left
/=	combined assignment	right-to-left
<<=	combined assignment	right-to-left
>>=	combined assignment	right-to-left
>>>=	combined assignment	right-to-left
&=	combined assignment	right-to-left
^=	combined assignment	right-to-left
|=	combined assignment	right-to-left