

1.2 Variables and Data Types

Self-Study Guide · About 50 minutes · With answer key

By the end you will be able to

- Explain what a data type is, in terms of its values and the operations allowed on them.
- Sort data types into the two categories, primitive and reference.
- Name the three primitive types used in this course, int, double, and boolean, and tell which values belong to each.
- Declare a variable with a type and a name, and explain what it means for a variable to hold a value.
- Explain why a variable can hold only values of its own type, and read the compiler message when it does not.
- Choose the most appropriate data type for a piece of information a program must store.

What to remember from earlier

- The compiler checks the whole program before it runs, and nothing runs if it finds an error. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.5 The Compiler: Checking Before Running](#))
- Statements in main run one at a time, from top to bottom. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.3 Sequencing: Order Matters](#))
- Java is case-sensitive, so System and system are different words. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.6 Syntax Errors](#))
- System.out.println displays something and then moves to a new line. ([Topic 1.1: Introduction to Algorithms, Programming, and Compilers](#))

1.2.1 Data Types: Values and Operations

Picture a video game. While you play, the game has to keep track of how many coins you have collected (1250), how fast your character is moving (4.75 squares per second), and whether the game is paused (yes or no). These are three very different kinds of information. A coin count is always a whole number. A speed can have a fractional part. "Paused" is not a number at all; it is just yes or no.

The program has to know which kind each piece of information is, for two reasons. First, it has to know which values make sense: a coin count of 2.5 is nonsense, and so is a "paused" of 17. Second, it has to know what can be done with each one: adding 50 coins to your coin count makes sense, but "adding" two yes-or-no answers does not.

Think of an online order form. The "Number of tickets" box accepts only whole numbers, and the "I agree to the terms" checkbox can only be checked or unchecked. The form also knows what to do with each field: it can multiply the number of tickets by a price, but it never tries to do math with the checkbox. Java works the same way, with one difference: the form checks your input while you type, but Java checks the code in your program before it runs, using the compiler you met in Topic 1.1.

In Java, each kind of information is described by a **data type**: a set of values, together with the set of operations that can be done on those values. Both halves matter. Knowing a value's data type tells you what it could possibly be and what you are allowed to do with it.

KEY IDEA

A data type is a set of values plus the operations that work on those values. Every value in a Java program has a data type.

WORKED EXAMPLE Describing the game's information as data types

For each piece of information in the game, describe both halves of its data type: the values it can take, and some operations that make sense for it.

Information	Set of values	Operations that make sense
Coin count	Whole numbers: ..., -2, -1, 0, 1, 2, ...	Add, subtract, multiply, divide, compare
Speed	Numbers that may have a fraction: 4.75, 0.5	Add, subtract, multiply, divide, compare
Paused?	Only two values: yes or no	Flip it (yes becomes no); combine it with another yes/no

Read across each row. The coin count and the speed share the same operations, because both are numbers, but their sets of values differ: 4.75 belongs to the speed's set, not the coin count's. The "paused" information has a tiny set of values (just two) and none of the arithmetic operations. Adding "yes" to "no" has no meaning, so it is not in the set of operations.

Now change one thing: a second program keeps track of a student's ID number, such as 40213. The values look exactly like a coin count: whole numbers. But would you ever add two ID numbers together, or find the average ID in a class? No. The answer would be meaningless. Two pieces of information can have similar-looking values and still be used in completely different ways. That is why a data type is defined by its operations as well as its values.

WATCH OUT

"A data type is just a list of allowed values." Students often remember only the values half. But the operations half is what makes types useful: it is how Java knows that adding two coin counts is fine and adding two yes-or-no answers is a mistake. When you describe a data type, always say what values it has **and** what can be done with them.

PAUSE AND TRY**Question 1** Multiple choice

Which description best fits a data type?

- A. The name a programmer gives to a stored value
- B. A set of values together with the operations that can be done on them
- C. Any number that appears in a program
- D. The order in which a program's statements run

Question 2 Short answer

A smart lamp app stores whether the lamp is on. Describe the set of values for this information, and explain whether "add 10" is an operation that belongs with it.

1.2.2 Primitive Types and Reference Types

Java has a lot of data types. A few are built right into the language for the simplest kinds of information: numbers and yes-or-no answers. Many more exist for bigger things, such as a piece of text, a picture, or the keyboard. Java sorts all of these into two categories, and knowing the category tells you a lot about how a type behaves.

In plain words: some types hold one simple value, like a single number. Everything else is for more complex things that bundle information together.

A **primitive type** is a data type built into the Java language that holds a simple value: a number or a true/false answer. A **reference type** is any data type that is not primitive. Reference types are used to define **objects**, which are bundles of related information that can also do things. You will learn exactly what objects are, and why these types have the name "reference," in Topic 1.12. For now, you need to know that the category exists and how to recognize it.

The most common reference type is `String`, the type for text. The message "Good morning!" that you displayed in Topic 1.1 is a `String`. Text is not a number or a yes/no answer, so `String` is not primitive.

KEY IDEA

Every data type is either primitive or reference. Primitive types hold simple values (numbers and true/false). Reference types, such as `String`, are used for objects.

WORKED EXAMPLE Sorting types into the two categories

Here are four type names you will use often. For each, decide the category by asking one question: does it hold a simple number or true/false value?

Type	Holds	Category	Why
<code>int</code>	Whole numbers	Primitive	A simple number
<code>double</code>	Numbers that may have a fraction	Primitive	A simple number
<code>boolean</code>	true or false	Primitive	A simple true/false value
<code>String</code>	Text, such as "Hi"	Reference	Text is not a number or true/false

There is also a quick visual clue. In this course, the three primitive types are lowercase Java words (`int`, `double`, `boolean`). Reference types are named after classes, and class names start with a capital letter (`String`; later you will meet `Scanner` and `Math`). Because Java is case-sensitive (Topic 1.1), the capital S in `String` is part of its name.

Now change one thing: imagine a program that stores a student's full name, "Maya Ortiz". You might think "it's just one name, so it's simple." But simple to a person is not the test. The test is

whether the value is a number or true/false, and text is neither, so a name is stored with the reference type `String`.

WATCH OUT

"Short text is simple, so it must be primitive." The size of the value has nothing to do with the category. A one-letter `String` such as `"A"` is still a `String`, and `String` is a reference type. Only number and true/false types are primitive in this course.

PAUSE AND TRY**Question 3** Multiple choice

Which of these is a reference type?

- A. `int`
- B. `boolean`
- C. `String`
- D. `double`

Question 4 Short answer

A classmate says, "`String` must be primitive because every program uses it." Explain what actually decides whether a type is primitive or reference, and give the correct category for `String`.

1.2.3 The Three Primitive Types: `int`, `double`, and `boolean`

You have seen the three primitive type names. Now you need to know exactly which values belong to each, because Java is strict: a value belongs to exactly one type, and Java decides which one by how the value is written.

Each primitive type matches one of the kinds of information from the game in 1.2.1:

- An `int` value is an integer: a whole number with no decimal point, such as `1250`, `0`, or `-7`.
- A `double` value is a real number: a number that can have a fractional part. In code it is written with a decimal point, such as `4.75`, `-0.5`, or `3.0`.
- A `boolean` value is either `true` or `false`. Those are the only two values, and they are written in lowercase with no quotation marks.

The names come from math and history. "Integer" is the math word for a whole number, and `int` is short for it. `double` is short for "double precision," a way of storing real numbers. `boolean` is named after George Boole, a mathematician who studied true/false logic.

Java actually has eight primitive types. The other five (`long`, `short`, `byte`, `float`, and `char`) are not used in this course or on the AP exam, so you will only work with `int`, `double`, and `boolean`.

KEY IDEA

The three primitive types in this course are `int` (integers), `double` (real numbers, written with a decimal point), and `boolean` (true or false).

WORKED EXAMPLE One value of each type

This code displays one value of each primitive type. (`System.out.println` displays a value and moves to a new line, as in Topic 1.1; output is taught fully in Topic 1.3. From here on, most examples show only the statements that go inside `main`.)

```
1 System.out.println(42);
2 System.out.println(3.14);
3 System.out.println(true);
```

Here is how Java classifies each value, and what is displayed:

Value	Type	How you can tell	Displayed
42	int	Whole number, no decimal point	42
3.14	double	Has a decimal point	3.14
true	boolean	One of the two true/false values	true

```
42
3.14
true
```

Now change one thing: write the same amount two ways.

```
1 System.out.println(7);
2 System.out.println(7.0);
```

```
7
7.0
```

7 and 7.0 stand for the same amount, but they are different values to Java. 7 has no decimal point, so it is an `int`. 7.0 has a decimal point, so it is a `double`, even though its fractional part is zero. Java keeps track of the difference and displays each one in its own form. The decimal point, not the size of the fraction, decides the type.

Remember from 1.2.1 that a type also comes with operations. The two number types share the arithmetic operations: add, subtract, multiply, divide, and remainder (you will use them in Topic 1.3). `boolean` values have different operations, which you will use in Unit 2 to make decisions. Arithmetic is not one of them.

WATCH OUT

Doing arithmetic with `boolean` values. Because `true` and `false` feel like "1 and 0," some students expect them to add like numbers. They do not, and the compiler checks this before the program runs:

```
1 System.out.println(true + false);
```

```
error: bad operand types for binary operator '+'
    first type:  boolean
    second type: boolean
```

The `+` symbol means "add" (Topic 1.3). An **operand** is a value an operator works on, so the message says that `+` cannot work on two `boolean` values: addition is not in the `boolean` set of operations. The exact wording can vary between Java versions, but it will name the operator and the `boolean` types. The fix is to use number values for arithmetic, and keep `true/false` for yes-or-no information.

PAUSE AND TRY

Question 5 Multiple choice

Which value is a `double`?

- A. `12`
- B. `-12`
- C. `12.0`
- D. `true`

Question 6 Tracing

What does this code display? Name the type of each value as you go.

```
1 System.out.println(-15);
2 System.out.println(0.5);
3 System.out.println(15.0);
4 System.out.println(false);
```

1.2.4 Declaring a Variable

So far, every value has been written straight into a `println` statement and then forgotten. A real game cannot work that way. It has to remember the coin count, use it later, and update it when you collect more coins. A program needs a place to keep a value.

In plain words: picture a labeled box. The label is a name, so you can find the box again. The box is made for one kind of thing: a box for whole numbers, a box for `true/false`, and so on. Whatever value is inside is what the program "remembers." The analogy has a limit: a real box can hold many things at once, but this box always holds exactly one value.

A **variable** is a storage location that holds a value. Every variable has a **name** and a data type. A variable of a primitive type holds one value from that type: an `int` variable holds an `int` value, and so on.

Before you can use a variable, you must **declare** it: write a statement that tells Java the variable's type and name, so Java can set aside the storage. A **variable declaration** looks like this:

```
1 int coins;
```

Piece	What it is
<code>int</code>	The data type. It fixes which values this variable can ever hold.
<code>coins</code>	The variable's name. The program uses this name to reach the value.
<code>;</code>	Ends the statement, like every Java statement (Topic 1.1).

Most of the time you also give the new variable a starting value in the same statement:

```
1 int coins = 1250;
```

The `=` stores the value on its right, `1250`, in the new variable on its left. The `=` symbol is called the assignment operator; you will learn all of its rules in Topic 1.4. A variable declared with no starting value, like `int coins;`, cannot be used until it is given a value, which is also part of Topic 1.4.

Naming a variable

Java has rules for names, and programmers follow a few conventions on top of them:

- A name has no spaces. In this course, build names from letters and digits, starting with a letter. (Java allows a couple of other symbols too, but you will not need them.)
- A name cannot be a word Java already uses, such as `int`, `double`, or `boolean`.
- Names are case-sensitive: `coins` and `Coins` are two different names.
- By convention, a variable's name starts with a lowercase letter, and each new word inside it starts with a capital: `highScore`, `milesWalked`. This style is called **camel case**, because the capitals stick up like a camel's humps.
- Choose a name that says what the value means. `coins` tells a reader much more than `c` or `x`.

KEY IDEA

A variable is a named storage location that holds one value of its data type. You declare it by writing the type, then the name, as in `int coins = 1250;`

WORKED EXAMPLE The game's three variables

This complete program declares one variable of each primitive type for the game, then displays the value in each one. Look at the type on the left of each declaration and the kind of value on the right.

GameStats.java

```
1 public class GameStats
2 {
3     public static void main(String[] args)
4     {
5         int coins = 1250;
6         double speed = 4.75;
7         boolean paused = false;
8         System.out.println(coins);
9         System.out.println(speed);
10        System.out.println(paused);
11    }
12 }
```

Lines 5 to 7 each declare a variable and store a starting value. After line 7 runs, memory looks like this:

Name	Type	Value it holds
coins	int	1250
speed	double	4.75
paused	boolean	false

Each value matches its variable's type: 1250 is an integer, 4.75 has a decimal point, and false is a boolean value. Lines 8 to 10 display the values. Notice there are no quotation marks around coins in line 8: the name means "the value stored in coins," so Java looks up the box and displays what is inside.

```
1250
4.75
false
```

Now change one thing: store the same amount in an int variable and in a double variable.

```
1 int wholeLaps = 5;
2 double exactLaps = 5.0;
3 System.out.println(wholeLaps);
4 System.out.println(exactLaps);
```

```
5
5.0
```

Both variables hold "five," but wholeLaps is an int variable holding the int value 5, and exactLaps is a double variable holding the double value 5.0. The variable's type decides what kind of value lives inside, and the value is displayed in that type's form. (In this chapter, a double variable always gets a value written with a decimal point. What Java does when you store a whole number in a double variable is part of Topic 1.5.)

WATCH OUT

Using a variable without declaring it. Students sometimes start storing a value in a name without first telling Java the type:

```
1 coins = 1250;
2 System.out.println(coins);
```

```
error: cannot find symbol
symbol:   variable coins
```

Java has never heard of coins, because no statement declared it, so the compiler cannot find any variable with that name. (You saw cannot find symbol in Topic 1.1 for a misspelled word; it means the same thing here.) Nothing runs. The fix is to declare the variable by putting its type

in front: `int coins = 1250;`. The same message appears if you declare `lapCount` and later type `lapcount`, because case matters in names.

PAUSE AND TRY

Question 7 Debugging

This code should store and display a high score, but it does not compile. Find the error and write the corrected code.

```
1 int high score = 980;  
2 System.out.println(high score);
```

Question 8 Tracing

Fill in a table with each variable's name, type, and value, then write the exact output.

```
1 int books = 4;  
2 double weight = 7.5;  
3 boolean zipped = true;  
4 System.out.println(weight);  
5 System.out.println(books);  
6 System.out.println(zipped);
```

1.2.5 A Variable's Value Can Change

The word "variable" means "able to vary." In the game, the coin count is not stuck at 1250 forever; every coin you pick up changes it. That is the whole point of storing it in a variable rather than writing `1250` everywhere.

A variable's value can change while the program is running. The box keeps its name and its type for the whole program; only the value inside changes. And because the box holds exactly one value, putting a new value in replaces the old one. The old value is gone.

To change a variable's value, write its name, `=`, and the new value. There is no type in front this time, because the variable already exists. (This kind of statement is an assignment statement, taught fully in Topic 1.4.)

KEY IDEA

A variable's value can change while the program runs, but it holds only one value at a time. Its name and type never change.

WORKED EXAMPLE Watching the coin count change

Follow `coins` one statement at a time. A trace table shows the value after each line.

```

1 int coins = 1250;
2 System.out.println(coins);
3 coins = 1300;
4 System.out.println(coins);
5 coins = 40;
6 System.out.println(coins);

```

Line	Statement	coins afterward	Displayed
1	int coins = 1250;	1250	
2	System.out.println(coins);	1250	1250
3	coins = 1300;	1300	
4	System.out.println(coins);	1300	1300
5	coins = 40;	40	
6	System.out.println(coins);	40	40

```

1250
1300
40

```

Line 1 is the only declaration, so it is the only line with `int`. Each `println` displays whatever is in the box **at that moment**. Line 3 did not add 1300 to 1250; it replaced 1250 with 1300. Line 5 replaced 1300 with 40, and Java does not remember that `coins` was ever 1250 or 1300.

Now change one thing: use a `boolean` variable instead of a number.

```

1 boolean paused = false;
2 System.out.println(paused);
3 paused = true;
4 System.out.println(paused);

```

```

false
true

```

The rule does not depend on the type. `paused` starts as `false`, then line 3 replaces it with `true`. It is still a `boolean` variable, and it still holds one value at a time.

WATCH OUT

Writing the type again when changing a value. Students often copy the declaration line and edit the number:

```

1 int coins = 1250;
2 int coins = 1300;
3 System.out.println(coins);

```

```
error: variable coins is already defined in method main(String[])
```

Writing `int` in front of `coins` a second time asks Java to make a **new** variable named `coins`, but one already exists, and two variables in the same place cannot share a name. The compiler stops, so nothing runs (not even the first line). The fix is to drop the type on the second line: `coins = 1300;`. Write the type exactly once, when you declare the variable.

PAUSE AND TRY

Question 9 Tracing

Trace `lives` with a table, then write the exact output.

```
1 int lives = 3;
2 System.out.println(lives);
3 lives = 2;
4 lives = 5;
5 System.out.println(lives);
6 lives = 0;
7 System.out.println(lives);
```

Question 10 Multiple choice

An `int` variable `score` holds 70. Then the statement `score = 85;` runs. What is true afterward?

- A. `score` holds 155.
- B. `score` holds both 70 and 85.
- C. `score` holds 85, and 70 is gone.
- D. `score` is now a new variable, separate from the old one.

1.2.6 A Variable Holds Only Values of Its Type

You know that a variable's value can change. Can it change to anything at all? What if a program tries to put 3.5 into an `int` variable?

Go back to 1.2.1: a data type is a set of values. An `int` variable promises that the value inside is always an integer, so the rest of the program can rely on that. If a decimal value could sneak in, the promise would be broken. So Java enforces a rule: a variable of a primitive type can hold only a value from its own type. Because the variable's type is written in the code, the compiler can check every value you try to store before the program runs, just as it checks for missing semicolons.

KEY IDEA

A variable can hold only values of its own data type. The compiler checks this, so a mismatched value is a compiler error and the program does not run.

WORKED EXAMPLE A decimal value in an int variable

A pizza tracker tries to store three and a half slices in an `int` variable:

```
1 int pizzaSlices = 3.5;
2 System.out.println(pizzaSlices);
```

error: incompatible types: possible lossy conversion from double to int

Read the message in pieces:

1. **incompatible types:** the value and the variable have types that do not match. `3.5` is a `double` value, and `pizzaSlices` is an `int` variable.
2. **possible lossy conversion from double to int:** to fit `3.5` into an `int` box, Java would have to throw away the `.5`, losing part of the value. "Lossy" means "something would be lost." Java refuses to do that silently. (Converting between types on purpose is Topic 1.5.)

The right fix depends on what the information means. If the program really needs to store half slices, the variable should be a `double`:

Fixed

```
1 double pizzaEaten = 3.5;
2 System.out.println(pizzaEaten);
```

3.5

Now change one thing: put a number into a `boolean` variable.

```
1 boolean lightsOn = 1;
2 System.out.println(lightsOn);
```

error: incompatible types: int cannot be converted to boolean

`1` is an `int`, and the only values in the `boolean` set are `true` and `false`. In Java, `1` does not mean "true." The fix is `boolean lightsOn = true;`. Notice that the message is a little different from the pizza one (there is no "lossy" this time), but the first part, `incompatible types`, is the same: a value from one type cannot go into a variable of another.

WATCH OUT

Capitalizing `true` or `false`. The `boolean` values are written in lowercase. Because Java is case-sensitive, `True` is not a value at all:

```
1 boolean lightsOn = True;
2 System.out.println(lightsOn);
```

error: cannot find symbol
symbol: variable True

Java sees an unknown name and goes looking for a variable called `True`. There is none, so the compiler reports `cannot find symbol`. Write `true`.

PAUSE AND TRY**Question 11** Multiple choice

Which declaration compiles?

- A. `int pages = 120.5;`
- B. `boolean done = 0;`
- C. `double height = 1.62;`
- D. `boolean ready = True;`

Question 12 Debugging

A classroom has exactly 30 seats. This code does not compile. Explain why, and write a corrected first line that keeps `seats` as an `int`.

```
1 int seats = 30.0;
2 System.out.println(seats);
```

1.2.7 Choosing the Most Appropriate Type

Every time you declare a variable, you must pick its type, and the compiler will hold you to that choice from then on. How do you decide? The type should fit **every value the information could ever have**, and no values it could never have.

Use the three primitive types and `String` from 1.2.2 to 1.2.3, and ask these questions in order:

1. Is the information a yes-or-no answer? Use `boolean`.
2. Is it a number that can only ever be whole, such as a count of things? Use `int`.
3. Is it a number that can have a fractional part, such as a measurement or a price? Use `double`.
4. Is it text, or anything else that is not a number or true/false? It needs a reference type, such as `String` for text.

Why not just use `double` for every number, since a `double` can hold 5.0 too? Because the type's set of values is a safety net. If the number of students in a class is an `int`, then 24.5 students cannot be stored; the compiler would stop it, as you saw in 1.2.6. A well-chosen type makes impossible values impossible, and it tells anyone reading the code what kind of information the variable holds.

KEY IDEA

Choose the data type whose set of values matches every value the information could have: `boolean` for yes/no, `int` for whole-number counts, `double` for numbers that may have a fraction, and a reference type such as `String` for text.

WORKED EXAMPLE Choosing types for a school app

For each piece of information, apply the questions in order:

Information	Best type	Reasoning
-------------	-----------	-----------

Number of students in a class	<code>int</code>	A count; there is never half a student
Price of a hoodie, such as \$24.99	<code>double</code>	Prices can have cents, a fractional part
Whether homework has been turned in	<code>boolean</code>	Yes or no
Outside temperature, such as 68.5°	<code>double</code>	A measurement that can have a fraction
Goals scored in a soccer game	<code>int</code>	A count of whole goals
A student's name	<code>String</code>	Text, so not primitive: a reference type

Each row is decided by the possible values, not by the topic. "Number of students" and "goals scored" have nothing to do with each other, but both are whole-number counts, so both are `int`.

Now change one thing: the temperature. If the program is for a thermostat whose screen shows only whole degrees, and the setting can only be changed one degree at a time, then every possible value is a whole number and `int` fits. If the program is a weather station that measures 68.5°, it needs `double`. The same kind of information can need a different type, because what decides the type is the set of values the program must store.

WATCH OUT

Choosing the type from one example value. A student sees that today's hoodie costs \$25 and declares `int hoodiePrice = 25;`. It works today, but tomorrow the price is \$24.99, and that value cannot go into an `int` variable. Do not ask "what is the value right now?" Ask "what values could this ever have?"

PAUSE AND TRY

Question 13 Multiple choice

A program for a cafe needs to store whether the cafe is currently open. Which declaration is most appropriate?

- A. `int open = 1;`
- B. `double open = 1.0;`
- C. `boolean open = true;`
- D. `boolean open = 1;`

Question 14 Short answer

A pizza party app stores two pieces of information: the number of pizzas ordered, and the amount of pizza eaten (for example, 2.75 pizzas). Choose a type for each and justify both choices.

Putting it together

A fitness app tracks a walker's day. It needs three pieces of information: the number of steps taken, the miles walked, and whether the daily goal has been reached. Here is a program that stores them, displays some of them, updates them in the afternoon, and displays them again.

Choosing the types (1.2.3, 1.2.7). Steps are counted one at a time, so they are always whole: `int`. Miles can be fractional, like 1.9: `double`. "Goal reached?" is yes or no: `boolean`. All three are primitive types (1.2.2).

StepTracker.java

```
1 public class StepTracker
2 {
3     public static void main(String[] args)
4     {
5         int steps = 4200;
6         double milesWalked = 1.9;
7         boolean goalReached = false;
8         System.out.println(steps);
9         System.out.println(goalReached);
10        steps = 10350;
11        milesWalked = 4.7;
12        goalReached = true;
13        System.out.println(steps);
14        System.out.println(milesWalked);
15        System.out.println(goalReached);
16    }
17 }
```

Declaring (1.2.4). Lines 5 to 7 declare the three variables. Each declaration has the type, the name, and a starting value from that type: 4200 is an integer, 1.9 has a decimal point, and false is a `boolean` value. All three names use camel case.

Changing values (1.2.5). Lines 10 to 12 store new values. They have no type in front, because the variables already exist. Each new value matches its variable's type (1.2.6), so the program compiles.

Line	steps	milesWalked	goalReached	Displayed
5-7	4200	1.9	false	
8	4200	1.9	false	4200
9	4200	1.9	false	false
10-12	10350	4.7	true	
13	10350	4.7	true	10350
14	10350	4.7	true	4.7
15	10350	4.7	true	true

The complete output:

```
4200
false
10350
4.7
true
```

`milesWalked` held 1.9 at first, but it was never displayed before line 11 replaced it, so 1.9 never appears in the output. And if line 10 had been `steps = 10350.5;`, the compiler would have reported

incompatible types: possible lossy conversion from double to int, just like the pizza example in 1.2.6, and the program would not have run at all.

Vocabulary

Term	Definition	Example
data type	A set of values together with the operations that can be done on them	<code>int</code>
primitive type	A data type built into Java that holds a simple number or true/false value	<code>int</code> , <code>double</code> , <code>boolean</code>
reference type	Any data type that is not primitive; used to define objects	<code>String</code>
object	A bundle of related information that can also do things (Topic 1.12)	A piece of text
int	The primitive type for integers (whole numbers)	<code>1250</code> , <code>-7</code>
double	The primitive type for real numbers, written with a decimal point	<code>4.75</code> , <code>3.0</code>
boolean	The primitive type whose only values are <code>true</code> and <code>false</code>	<code>false</code>
operand	A value that an operator, such as <code>+</code> , works on	In <code>true + false</code> , both values
variable	A storage location with a name and a data type that holds one value	<code>coins</code>
declare / variable declaration	Create a variable by stating its type and name	<code>int coins;</code>
camel case	Naming style: first word lowercase, each later word capitalized	<code>milesWalked</code>

Chapter summary

- A **data type** is a set of values plus the operations that work on them. Two kinds of information can look alike and still need different types because of what you do with them.
- Every type is either **primitive** (built in; simple number or true/false values) or **reference** (everything else, used for objects, such as `String`).
- This course uses three primitive types: `int` for integers, `double` for real numbers written with a decimal point, and `boolean` for `true` or `false`. Java's other five primitive types are not part of the course.
- A **variable** is a named storage location with a type. You **declare** it once, with its type and name. Its value can change while the program runs, but it holds one value at a time.
- A variable holds only values of its own type, and the compiler enforces this. Choose the type whose values match every value the information could ever have.

Review questions

Question 15 Multiple choice

Which statement is true?

- A. Java has exactly three primitive types: `int`, `double`, and `boolean`.
- B. `String` is a primitive type because it is built into every Java program.
- C. A reference type is any type that is not primitive, and reference types are used for objects.
- D. A `boolean` is a reference type, because it holds a word like `true` rather than a number.

Question 16 Tracing

Trace all three variables, then write the exact output.

```
1 double roomTemp = 68.5;
2 int fanSpeed = 2;
3 boolean heatOn = false;
4 System.out.println(roomTemp);
5 roomTemp = 70.0;
6 heatOn = true;
7 fanSpeed = 3;
8 System.out.println(fanSpeed);
9 System.out.println(roomTemp);
10 System.out.println(heatOn);
```

Question 17 Debugging

This program should display `250` and then `false`, but it has two compiler errors. Find both and write the corrected lines.

Concert.java

```
1 public class Concert
2 {
3     public static void main(String[] args)
4     {
5         int ticketsSold = 212;
6         double ticketPrice = 15.5;
7         boolean soldOut = 0;
8         int ticketsSold = 250;
9         System.out.println(ticketsSold);
10        System.out.println(soldOut);
11    }
12 }
```

Question 18 Multiple choice

A parking garage program must store the number of cars inside, the hourly parking rate (such as \$3.50), and whether the garage is full. Which set of declarations is most appropriate?

- A. `int cars = 0; int rate = 3; boolean full = false;`
- B. `int cars = 0; double rate = 3.50; boolean full = false;`
- C. `double cars = 0.0; double rate = 3.50; double full = 0.0;`
- D. `int cars = 0; double rate = 3.50; int full = 0;`

Question 19 Short answer

A student says, "I'll declare every number as a `double`, since a `double` can hold 5.0 and 5.25. That way I never get a compiler error." Using the idea that a data type is a set of values, explain why `int` is a better choice for the number of tickets sold.

Question 20 Multiple choice

The statement `int total = 9.0;` does not compile, even though 9.0 equals 9. Which explanation is correct?

- A. Variable names cannot start with the letter `t`.
- B. `9.0` is a `double` value because it is written with a decimal point, and an `int` variable can hold only `int` values.
- C. An `int` variable cannot be given a starting value when it is declared.
- D. Java rounds `9.0` up to `10`, which is too large for the variable.

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can explain that a data type is a set of values plus the operations on them. Reread: 1.2.1 Data Types: Values and Operations
- ☐ I can tell whether a type is primitive or reference, and say why `String` is a reference type. Reread: 1.2.2 Primitive Types and Reference Types
- ☐ I can name the three primitive types in this course and tell which type a value like 7, 7.0, or `true` has. Reread: 1.2.3 The Three Primitive Types: `int`, `double`, and `boolean`
- ☐ I can declare a variable with a type and a meaningful camel-case name. Reread: 1.2.4 Declaring a Variable
- ☐ I can trace a variable whose value changes, knowing it holds one value at a time. Reread: 1.2.5 A Variable's Value Can Change
- ☐ I can explain why a value of the wrong type cannot be stored in a variable, and read the compiler message. Reread: 1.2.6 A Variable Holds Only Values of Its Type
- ☐ I can choose the most appropriate data type for a piece of information. Reread: 1.2.7 Choosing the Most Appropriate Type

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

B. A data type has two halves: the values that belong to it and the operations that work on those values.

- **A** is wrong: a name for a stored value belongs to a variable, which you will meet in 1.2.4. A data type describes a kind of value, not one particular value.
- **C** is wrong: a number is a value. Values belong to a data type, but a single value is not a type. Also, not every type is about numbers (think of yes/no).
- **D** is wrong: that describes sequencing from Topic 1.1, not data types.

Reread 1.2.1 *Data Types: Values and Operations* if you chose differently.

Question 2

Model answer: The set of values has only two members: on or off (yes or no). "Add 10" does not belong, because adding a number to "on" has no meaning. Operations that do make sense are things like switching it to the opposite value.

A complete answer must:

- name the two values (on/off, yes/no, or true/false), and
- explain that arithmetic such as adding is not one of its operations because it would have no meaning.

Reread 1.2.1 *Data Types: Values and Operations* if your answer differed.

Question 3

C. `String` holds text, which is not a number or true/false value, so it is a reference type, used for objects. Its capital letter is a clue: it is named after a class.

- **A** is wrong: `int` holds whole numbers, so it is primitive.
- **B** is wrong: `boolean` holds true or false, so it is primitive.
- **D** is wrong: `double` holds numbers, so it is primitive.

Reread 1.2.2 *Primitive Types and Reference Types* if you chose differently.

Question 4

Model answer: How often a type is used does not matter. A primitive type is built into Java and holds a simple value, a number or true/false. Any type that is not primitive is a reference type, used for objects. `String` holds text, so it is a reference type.

A complete answer must:

- say that primitive types hold simple number or true/false values (and reference types are everything else, used for objects), and
- correctly name `String` as a reference type.

Reread 1.2.2 *Primitive Types and Reference Types* if your explanation differed.

Question 5

C. A `double` value is written with a decimal point. `12.0` has one, so it is a `double`, even though its fractional part is zero.

- **A** is wrong: `12` has no decimal point, so it is an `int`.
- **B** is wrong: a minus sign does not change the type. `-12` is a negative integer, an `int`.
- **D** is wrong: `true` is one of the two `boolean` values.

Reread 1.2.3 *The Three Primitive Types: int, double, and boolean* if you chose differently.

Question 6

Value	Type	Displayed
-15	int	-15
0.5	double	0.5
15.0	double	15.0
false	boolean	false

```
-15
0.5
15.0
false
```

`15.0` is displayed with its `.0` because it is a `double`; Java does not drop the decimal point. `-15` is still an `int`, because a negative whole number is still an integer.

Reread 1.2.3 *The Three Primitive Types: int, double, and boolean* if your output or types differed.

Question 7

Error: the name `high score` contains a space, and a variable name cannot contain spaces. Java reads `int high` as a complete declaration and then does not know what to do with `score`. The compiler reports `';' expected at the space`, and nothing runs.

Corrected code:

```
1 int highScore = 980;
2 System.out.println(highScore);
```

Joining the words in camel case, `highScore`, makes one legal name. The name must be fixed in **both** lines, because the second line must use exactly the same name. The corrected code displays `980`.

Reread the naming rules in 1.2.4 *Declaring a Variable* if you missed the error.

Question 8

Name	Type	Value
books	int	4
weight	double	7.5
zipped	boolean	true

The `println` statements run in the order written, top to bottom, not in the order the variables were declared. `weight` is displayed first:

```
7.5
4
true
```

Reread 1.2.4 *Declaring a Variable* if your table or output differed.

Question 9

Line	Lives afterward	Displayed
1	3	
2	3	3
3	2	
4	5	
5	5	5
6	0	
7	0	0

```
3
5
0
```

The value 2 is never displayed: line 4 replaced it before any `println` ran. A variable holds one value at a time, and `println` shows only the current one.

Reread 1.2.5 *A Variable's Value Can Change* if your output differed.

Question 10

C. A variable holds one value at a time, so storing 85 replaces 70.

- **A** is wrong: storing a value replaces; it does not add.
- **B** is wrong: a variable holds exactly one value, never a list of old ones.
- **D** is wrong: there is no type in front of `score`, so this statement changes the existing variable. Only a declaration (with a type) creates a variable.

Reread 1.2.5 *A Variable's Value Can Change* if you chose differently.

Question 11

C. 1.62 is a `double` value going into a `double` variable, so the types match.

- **A** is wrong: 120.5 is a `double` value. Storing it in an `int` variable would lose the .5, so the compiler reports incompatible types: possible lossy conversion from double to int.
- **B** is wrong: 0 is an `int`, not a boolean value, so the compiler reports `int cannot be converted to boolean`. 0 does not mean false in Java.
- **D** is wrong: `True` with a capital T is not a value; Java reports `cannot find symbol`. The value is `true`.

Reread 1.2.6 *A Variable Holds Only Values of Its Type* if you chose differently.

Question 12

Error: `30.0` has a decimal point, so it is a `double` value, even though its fractional part is zero. A `double` value cannot be stored in an `int` variable, so the compiler reports incompatible types: possible lossy conversion from `double` to `int`.

Corrected line:

```
1 int seats = 30;
```

`30` with no decimal point is an `int` value, so it matches the variable, and the code displays `30`. Keeping `int` is the right choice here because a number of seats is always a whole number.

Reread 1.2.6 *A Variable Holds Only Values of Its Type* and the `7` versus `7.0` example in 1.2.3 if you missed it.

Question 13

C. "Is the cafe open?" is a yes-or-no question, and `boolean` is the type whose only values are `true` and `false`.

- **A** is wrong: it compiles, but an `int` allows values like `7` or `-3`, which mean nothing for "open," and arithmetic on it would make no sense. The type's set of values does not match the information.
- **B** is wrong: for the same reason, plus it allows fractions like `0.5`.
- **D** is wrong: `boolean` is the right type, but `1` is an `int` value, not `true`, so this declaration does not even compile (incompatible types: `int` cannot be converted to `boolean`).

Reread 1.2.7 *Choosing the Most Appropriate Type* if you chose differently.

Question 14

Model answer: The number of pizzas ordered should be an `int`, because you can only order whole pizzas, so every possible value is a whole number. The amount eaten should be a `double`, because people eat parts of pizzas, so the value can have a fractional part, like `2.75`.

A complete answer must:

- choose `int` for pizzas ordered, because it is a whole-number count, and
- choose `double` for the amount eaten, because it can include a fraction.

Both pieces of information are about pizzas, but their sets of possible values differ, and the set of values is what decides the type. Reread 1.2.7 *Choosing the Most Appropriate Type* if your reasoning differed.

Question 15

C. Every data type is either primitive or reference. Reference types are all the non-primitive types, and they are used to define objects.

- **A** is wrong: Java has eight primitive types. This **course** uses three of them; the other five (`long`, `short`, `byte`, `float`, `char`) are not part of it. "The course uses three" and "Java has three" are different claims.
- **B** is wrong: being used often does not make a type primitive. `String` holds text, which is not a simple number or true/false value, so it is a reference type.
- **D** is wrong: `true` is not a word stored as text; it is one of the two `boolean` values, and `boolean` is primitive.

Reread 1.2.2 *Primitive Types and Reference Types* and 1.2.3 *The Three Primitive Types: int, double, and boolean* if you chose differently.

Question 16

Line	roomTemp	fanSpeed	heatOn	Displayed
1-3	68.5	2	false	
4	68.5	2	false	68.5
5	70.0	2	false	
6	70.0	2	true	
7	70.0	3	true	
8	70.0	3	true	3
9	70.0	3	true	70.0
10	70.0	3	true	true

```
68.5
3
70.0
true
```

Two details to check: 70.0 is displayed with its .0 because roomTemp is a double, and fanSpeed's old value 2 is never displayed because line 7 replaced it before line 8 ran.

Reread 1.2.5 *A Variable's Value Can Change* and the 7 versus 7.0 example in 1.2.3 if your output differed.

Question 17

Error 1 (line 7): 0 is an int value, and soldOut is a boolean variable. The compiler reports incompatible types: int cannot be converted to boolean. In Java, 0 does not mean false.

Error 2 (line 8): int ticketsSold declares the variable a second time. The compiler reports variable ticketsSold is already defined. To change the value of an existing variable, leave off the type.

Corrected lines:

```
1 boolean soldOut = false;
2 ticketsSold = 250;
```

With both fixes, line 8 replaces 212 with 250, and the program displays 250 and then false. Both errors are compiler errors, so the original program displays nothing at all.

Reread 1.2.6 *A Variable Holds Only Values of Its Type* and 1.2.5 *A Variable's Value Can Change* if you missed either error.

Question 18

B. Cars are counted in whole numbers (int), a rate like \$3.50 can have cents (double), and "full?" is yes or no (boolean).

- **A** is wrong: an int rate cannot hold \$3.50. Choosing the type from a whole-dollar example value breaks as soon as the rate has cents.

- **C** is wrong: `double` allows 12.5 cars, which is impossible, and a `double` for "full?" allows meaningless values like 0.7. It compiles, but the types do not match the information.
- **D** is wrong: an `int` for "full?" allows values like 5 that mean nothing; `boolean` matches a yes/no answer exactly.

Reread *1.2.7 Choosing the Most Appropriate Type* if you chose differently.

Question 19

Model answer: A data type's set of values decides what the variable is allowed to hold. Tickets are sold whole, so the number sold is always a whole number. If it is an `int`, a value like 212.5 cannot be stored: the compiler rejects it, which catches a mistake before the program runs. If it is a `double`, 212.5 would be accepted silently, even though it is impossible. The `int` type also tells anyone reading the code that the value is a count.

A complete answer must:

- say that the number of tickets can only be a whole number, so `int`'s set of values matches it, and
- explain that a `double` would allow impossible fractional values that the compiler would otherwise catch.

Reread *1.2.7 Choosing the Most Appropriate Type* and *1.2.6 A Variable Holds Only Values of Its Type* if your explanation differed.

Question 20

B. Java decides a value's type from how it is written. The decimal point makes 9.0 a `double`, and a variable holds only values of its own type, so the compiler reports incompatible types: possible lossy conversion from `double` to `int`.

- **A** is wrong: names may start with any letter. `total` is a fine, meaningful name.
- **C** is wrong: giving a starting value in the declaration is normal, as in `int coins = 1250;`. The problem is the type of the value, not the fact that there is one.
- **D** is wrong: Java does not round or change the value; it refuses to compile. Nothing about 9 or 10 is "too large" for an `int`.

Reread *1.2.6 A Variable Holds Only Values of Its Type* and *1.2.3 The Three Primitive Types: int, double, and boolean* if you chose differently.