

3.7 Class Variables and Methods

Self-Study Guide · About 55 minutes · With answer key

By the end you will be able to

- Declare a class variable with static and explain why every object of the class shares one copy of it.
- Use a public class variable from outside its class with the class name and a dot.
- Use final to make a variable whose value cannot be changed, and write class constants.
- Write class methods that read and change class variables and call other class methods.
- Explain why a class method cannot use instance variables or instance methods unless an object is passed to it, and fix code that tries.

What to remember from earlier

- A class method is called with the class name, a dot, and the method name. ([Topic 1.10: Calling Class Methods](#), [1.10.2 Calling with the Class Name and a Dot](#))
- The keyword static marks a method that belongs to the class, not to any one object. ([Topic 1.10: Calling Class Methods](#), [1.10.1 Class Methods Belong to the Class](#))
- Each object gets its own copy of every instance variable, usually private. ([Topic 3.3: Anatomy of a Class](#), [3.3.2 Each Object Has Its Own Instance Variables](#))
- Accessors return a copy of a value and mutators change one. ([Topic 3.5: Methods: How to Write Them](#))
- A parameter that holds a reference points at the same object as the argument. ([Topic 3.6: Methods: Passing and Returning References of an Object](#), [3.6.1 Passing an Object Reference](#))
- A method may use the private data of a parameter of its own class. ([Topic 3.6: Methods: Passing and Returning References of an Object](#), [3.6.5 Private Data of a Parameter](#))

3.7.1 Class Variables Are Shared

A club wants its `Member` class to keep track of how many members have joined. Where should that number live? It cannot be an ordinary instance variable. Recall from Topic 3.3 that every object gets its **own** copy of each instance variable. If every `Member` object had its own count, each one would only know about itself. The count is not a fact about Ana or about Ben; it is a fact about the club as a whole.

Java's answer is a variable that belongs to the class instead of to any one object. There is exactly one copy of it, no matter how many objects exist, and every object of the class uses that same copy. Think of a classroom whiteboard: each student has their own notebook (an instance variable), but there is only one whiteboard, and when one student writes on it, everyone sees the change. The comparison has a limit: a whiteboard sits in one room, while a Java class variable can be reached by any code that is allowed to see it, as you will learn in 3.7.2.

A **class variable** is a variable that belongs to the class itself, so all objects of the class share a single copy of it. You make one by writing the keyword `static` before the variable's type in the

declaration. You met `static` in Topic 1.10 as the word that marks a class method; on a variable it means the same thing: "this belongs to the class."

```
1 private static int memberCount = 0;
```

Part	What it means
<code>private</code>	Only code inside the class can use it, the same rule as for instance variables.
<code>static</code>	One shared copy that belongs to the class. Leave it out and every object gets its own copy.
<code>int</code>	The type, as with any variable.
<code>memberCount</code>	The name.
<code>= 0</code>	The starting value, given right in the declaration.

Why is the starting value written in the declaration instead of in the constructor? A constructor runs once **for each new object**. If the constructor set `memberCount = 0`, every new member would wipe out the count. A class variable is set up once, when the class is first used, so its starting value goes on the declaration line.

KEY IDEA

A class variable is declared with `static` before its type. It belongs to the class, so there is exactly one copy, shared by every object of the class.

WORKED EXAMPLE Counting club members

The constructor below adds 1 to the shared count each time a `Member` is made. Watch what `ana.getMemberCount()` returns after each new member joins.

Every example in this chapter fits in one file. A Java file may hold only one `public` class, so the helper class (here `Member`) is written without `public` in front of `class`. In your own projects, each class would normally be `public` and live in its own file.

MemberCount.java

```

1  class Member
2  {
3      private String name;
4      private static int memberCount = 0;
5
6      public Member(String memberName)
7      {
8          name = memberName;
9          memberCount++;
10     }
11
12     public int getMemberCount()
13     {
14         return memberCount;
15     }
16 }
17
18 public class MemberCount
19 {
20     public static void main(String[] args)
21     {
22         Member ana = new Member("Ana");
23         System.out.println(ana.getMemberCount());
24         Member ben = new Member("Ben");
25         System.out.println(ana.getMemberCount());
26         Member cy = new Member("Cy");
27         System.out.println(ana.getMemberCount());
28         System.out.println(cy.getMemberCount());
29     }
30 }

```

- Line 3 declares an instance variable: each `Member` has its own `name`.
- Line 4 declares a class variable: one `memberCount` for the whole class, starting at 0.
- Line 9, inside the constructor, adds 1 to that one shared copy. Constructors and instance methods are allowed to use class variables.
- Lines 12 to 15 are an accessor (Topic 3.5) that returns the shared count.

Trace `main`. The memory picture after line 26 shows three objects, each with its own `name`, and one `memberCount` that belongs to the class:

Member class (one shared copy): `memberCount = 3`

```

ana --> [ name = "Ana" ]
ben --> [ name = "Ben" ]
cy  --> [ name = "Cy"  ]

```

Line	What happens	memberCount	Printed
22	Ana's object is made; constructor adds 1	1	
23	<code>ana.getMemberCount()</code>	1	1
24	Ben's object is made; constructor adds 1	2	

25	<code>ana.getMemberCount()</code> sees Ben's change	2	2
26	Cy's object is made; constructor adds 1	3	
27	<code>ana.getMemberCount()</code>	3	3
28	<code>cy.getMemberCount()</code> reads the same copy	3	3

```
1
2
3
3
```

Line 25 is the key moment. Nothing was done to Ana's object, yet asking it for the count gives a new answer. That is because there is no count stored inside Ana's object; every member reads the one shared copy.

Now vary the example so an instance variable and a class variable change side by side. Each Player keeps their own `points`, and the team keeps one shared `teamPoints`:

TeamScore.java

```
1 class Player
2 {
3     private int points;
4     private static int teamPoints = 0;
5
6     public Player()
7     {
8         points = 0;
9     }
10
11    public void scorePoints(int amount)
12    {
13        points += amount;
14        teamPoints += amount;
15    }
16
17    public int getPoints()
18    {
19        return points;
20    }
21
22    public int getTeamPoints()
23    {
24        return teamPoints;
25    }
26 }
27
28 public class TeamScore
29 {
30     public static void main(String[] args)
31     {
32         Player maya = new Player();
33         Player leo = new Player();
34         maya.scorePoints(3);
```

```

35         leo.scorePoints(2);
36         maya.scorePoints(4);
37         System.out.println(maya.getPoints());
38         System.out.println(leo.getPoints());
39         System.out.println(leo.getTeamPoints());
40     }
41 }

```

scorePoints (lines 11 to 15) adds the same amount to both variables, but they behave differently:

Line and call	maya.points	leo.points	teamPoints (shared)
32–33 two players made	0	0	0
34 maya.scorePoints(3)	3	0	3
35 leo.scorePoints(2)	3	2	5
36 maya.scorePoints(4)	7	2	9

```

7
2
9

```

Each call changes only the `points` of the object it was called on, but every call changes the single `teamPoints`. That is why `leo.getTeamPoints()` reports 9, including the 7 points Maya scored.

WATCH OUT

Forgetting static on a counter. Here is the member program with one word removed: line 4 says `private int memberCount = 0;` without `static`.

MemberNoStatic.java

```
1 class Member
2 {
3     private String name;
4     private int memberCount = 0;
5
6     public Member(String memberName)
7     {
8         name = memberName;
9         memberCount++;
10    }
11
12    public int getMemberCount()
13    {
14        return memberCount;
15    }
16 }
17
18 public class MemberNoStatic
19 {
20     public static void main(String[] args)
21     {
22         Member ana = new Member("Ana");
23         System.out.println(ana.getMemberCount());
24         Member ben = new Member("Ben");
25         System.out.println(ana.getMemberCount());
26         Member cy = new Member("Cy");
27         System.out.println(ana.getMemberCount());
28         System.out.println(cy.getMemberCount());
29     }
30 }
```

It compiles and runs, but prints the wrong thing:

```
1
1
1
1
```

Without `static`, `memberCount` is an instance variable, so every new object gets its own copy starting at 0, and its constructor raises only that copy to 1. No object ever sees another object's increment. Students make this mistake because the declaration looks almost the same either way. The fix is the one word: `private static int memberCount = 0;`. Ask yourself, "Is this value about one object, or about all of them?" If it is about all of them, it needs `static`.

Where in a class each kind of variable may be used is the subject of Topic 3.8.

PAUSE AND TRY**Question 1** Multiple choice

A `Ticket` class needs a variable that every `Ticket` object shares, to hold the total number of tickets printed. Which declaration is correct?

- A. `private int totalPrinted = 0;`
- B. `private static int totalPrinted = 0;`
- C. `private int static totalPrinted = 0;`
- D. `static private totalPrinted = 0;`

Question 2 Tracing

Write the exact output.

`BikeRides.java`

```
1  class Bike
2  {
3      private int rides;
4      private static int totalRides = 0;
5
6      public Bike(int startingRides)
7      {
8          rides = startingRides;
9          totalRides += startingRides;
10     }
11
12     public void ride()
13     {
14         rides++;
15         totalRides++;
16     }
17
18     public int getRides()
19     {
20         return rides;
21     }
22
23     public int getTotalRides()
24     {
25         return totalRides;
26     }
27 }
28
29 public class BikeRides
30 {
31     public static void main(String[] args)
32     {
33         Bike red = new Bike(5);
34         Bike blue = new Bike(0);
35         blue.ride();
36         red.ride();
37         blue.ride();
38         System.out.println(red.getRides());
```

```
39         System.out.println(blue.getRides());
40         System.out.println(red.getTotalRides());
41     }
42 }
```

3.7.2 Public Class Variables and the Class Name

So far the shared count was `private`, so code outside the class needed an object to ask for it. But a class variable does not belong to any object. How should code in another class, such as `main`, refer to it directly?

Use the same pattern you used for class methods in Topic 1.10: the class name, the dot operator, then the variable name. If a class variable is declared `public`, any class can reach it as `ClassName.variableName`. The class name is the natural thing to write because the variable is attached to the class, not to an object. You have already done this once: in Topic 1.5 you used `Integer.MAX_VALUE`, which is a public class variable of Java's `Integer` class.

KEY IDEA

A `public` class variable is used from outside its class by writing the class name, a dot, and the variable name, as in `Raffle.ticketsSold`.

WORKED EXAMPLE Reading a public class variable from main

`ticketsSold` is `public static`, so `main`, which is in a different class, can use it directly.

RaffleSales.java

```

1  class Raffle
2  {
3      public static int ticketsSold = 0;
4      private String buyer;
5
6      public Raffle(String buyerName)
7      {
8          buyer = buyerName;
9          ticketsSold++;
10     }
11 }
12
13 public class RaffleSales
14 {
15     public static void main(String[] args)
16     {
17         System.out.println(Raffle.ticketsSold);
18         Raffle first = new Raffle("Jo");
19         Raffle second = new Raffle("Sam");
20         System.out.println(Raffle.ticketsSold);
21         Raffle.ticketsSold = 0;
22         System.out.println(Raffle.ticketsSold);
23     }
24 }

```

Line	What happens	Raffle.ticketsSold	Printed
17	Read through the class name; no objects exist yet	0	0
18	Jo's ticket is made; constructor adds 1	1	
19	Sam's ticket is made; constructor adds 1	2	
20	Read through the class name	2	2
21	main assigns 0 to the shared copy	0	
22	Read again	0	0

```

0
2
0

```

Two things to notice:

- **Line 17 works before any Raffle object exists.** A class variable is not stored inside objects, so it is there as soon as the class is used, even with zero objects.
- **Line 21 shows the danger of public.** Any code anywhere can change a public variable, so the count now says 0 even though two tickets exist. This is why Topic 3.3 recommends `private` for data: the class should control its own data. A public class variable is used when the class's specification calls for it; in 3.7.3 you will see the most common safe case, a value that can never be changed.

Now vary the access. Here `ticketsSold` is `private` and `main` tries the same read:

RafflePrivate.java

```
1 class Raffle
2 {
3     private static int ticketsSold = 0;
4     private String buyer;
5
6     public Raffle(String buyerName)
7     {
8         buyer = buyerName;
9         ticketsSold++;
10    }
11 }
12
13 public class RafflePrivate
14 {
15     public static void main(String[] args)
16     {
17         System.out.println(Raffle.ticketsSold);
18         Raffle first = new Raffle("Jo");
19         Raffle second = new Raffle("Sam");
20         System.out.println(Raffle.ticketsSold);
21         System.out.println(Raffle.ticketsSold);
22     }
23 }
```

The compiler stops at line 17:

```
RafflePrivate.java:17: error: ticketsSold has private access in Raffle
```

The class name and dot are the right *form*, but `private` still means only code inside `Raffle` may touch the variable. Writing the class name never gets around `private`. To share a private class variable safely, the class provides a method that returns it. You will write that kind of method, one that needs no object at all, in 3.7.4.

WATCH OUT

Leaving off the class name outside the class. Inside `Raffle`, the constructor can write just `ticketsSold`, because it is in the class that declares the variable. Code in another class cannot. If `main` writes `System.out.println(ticketsSold);`, Java looks for a variable named `ticketsSold` in `main` and in `main`'s own class, finds none, and reports `error: cannot find symbol`. This is the same rule you learned for class methods in Topic 1.10: inside the defining class, the class name is optional; from any other class, it is required. Write `Raffle.ticketsSold`.

PAUSE AND TRY**Question 3** Debugging

This program should print 1, but it does not compile. Find the error and write the corrected line.

RaffleNoClassName.java

```
1  class Raffle
2  {
3      public static int ticketsSold = 0;
4      private String buyer;
5
6      public Raffle(String buyerName)
7      {
8          buyer = buyerName;
9          ticketsSold++;
10     }
11 }
12
13 public class RaffleNoClassName
14 {
15     public static void main(String[] args)
16     {
17         Raffle first = new Raffle("Jo");
18         System.out.println(ticketsSold);
19     }
20 }
```

Question 4 Multiple choice

What does this program print?

VisitorCount.java

```
1 class Museum
2 {
3     public static int visitors = 0;
4     private String exhibit;
5
6     public Museum(String exhibitName)
7     {
8         exhibit = exhibitName;
9     }
10
11    public void enter()
12    {
13        visitors++;
14    }
15 }
16
17 public class VisitorCount
18 {
19     public static void main(String[] args)
20     {
21         Museum dinosaurs = new Museum("Dinosaurs");
22         Museum space = new Museum("Space");
23         dinosaurs.enter();
24         space.enter();
25         space.enter();
26         System.out.println(Museum.visitors);
27     }
28 }
```

- A. 1
- B. 2
- C. 3
- D. Nothing; Museum.visitors does not compile because visitors must be reached through an object.

3.7.3 Final Variables Cannot Change

Some values should never change while a program runs: the number of slices in a pizza, the passing score on a test, a student's ID number. If such a value is stored in an ordinary variable, a careless line like the one in 3.7.2 could change it, and every calculation after that would be wrong. You want Java itself to refuse the change.

That is what the keyword **final** does. A **final variable** is a variable whose value cannot be modified once it has been given. Any statement that tries to assign it a new value is a compile error, so the mistake is caught before the program ever runs.

final combines naturally with **static**. A value like "8 slices per pizza" is the same for every pizza order, so there is no reason for each object to store its own copy; one shared copy is enough. And because it can never change, it is safe to make **public**: nobody can damage it. A

`public static final` variable is called a **class constant**. By custom, Java programmers write constant names in all capital letters with underscores between words, such as `SLICES_PER_PIZZA`, so readers can spot them. `Integer.MAX_VALUE` from Topic 1.5 is a class constant.

```
1 public static final int SLICES_PER_PIZZA = 8;
```

Read it as: anyone may use it (`public`), there is one copy for the class (`static`), it cannot change (`final`), it is an `int` named `SLICES_PER_PIZZA`, and its value is 8.

KEY IDEA

A variable declared `final` cannot be changed after it gets its value. A `public static final` variable is a class constant: one shared, unchangeable value, used as `ClassName.CONSTANT_NAME`.

WORKED EXAMPLE A class constant for pizza slices

PizzaParty.java

```
1 class PizzaOrder
2 {
3     public static final int SLICES_PER_PIZZA = 8;
4     private int pizzas;
5
6     public PizzaOrder(int pizzaCount)
7     {
8         pizzas = pizzaCount;
9     }
10
11     public int getSlices()
12     {
13         return pizzas * SLICES_PER_PIZZA;
14     }
15 }
16
17 public class PizzaParty
18 {
19     public static void main(String[] args)
20     {
21         System.out.println(PizzaOrder.SLICES_PER_PIZZA);
22         PizzaOrder party = new PizzaOrder(4);
23         System.out.println(party.getSlices());
24     }
25 }
```

- Line 3 declares the class constant.
- Line 13 is inside `PizzaOrder`, so it uses the constant by its name alone:
`pizzas * SLICES_PER_PIZZA`.
- Line 21 is in another class, so it uses the class name: `PizzaOrder.SLICES_PER_PIZZA`. It prints 8, and no `PizzaOrder` object exists yet.
- Line 22 makes an order of 4 pizzas, and line 23 prints `4 * 8`, which is 32.

8
32

`final` is not only for class variables; it can go on any variable. Vary the example: here `final` is on an **instance** variable. Each student gets their own ID, set once in the constructor, and it should never change, while a nickname may change any time.

StudentId.java

```
1  class Student
2  {
3      private final int idNumber;
4      private String nickname;
5
6      public Student(int id, String startingNickname)
7      {
8          idNumber = id;
9          nickname = startingNickname;
10     }
11
12     public void setNickname(String newNickname)
13     {
14         nickname = newNickname;
15     }
16
17     public int getIdNumber()
18     {
19         return idNumber;
20     }
21
22     public String getNickname()
23     {
24         return nickname;
25     }
26 }
27
28 public class StudentId
29 {
30     public static void main(String[] args)
31     {
32         Student kim = new Student(4417, "Kimmy");
33         Student raj = new Student(5280, "Raj");
34         kim.setNickname("K");
35         System.out.println(kim.getNickname());
36         System.out.println(kim.getIdNumber());
37         System.out.println(raj.getIdNumber());
38     }
39 }
```

Line 3 declares `idNumber` as `private final int` with no value. The constructor gives it one on line 8, once per object. Kim's nickname changes on line 34, but her ID stays 4417, and Raj has his own ID, 5280, because `idNumber` is not static:

```
K
4417
5280
```

If the class had a method such as `setIdNumber` that tried `idNumber = newId;`, the class would not compile: Java reports `cannot assign a value to final variable idNumber`. The same goes for a `final` local variable. In this code the third line fails to compile with `cannot assign a value to final variable minutesPerHour`:

```
1 final int minutesPerHour = 60;
2 int hours = 3;
3 minutesPerHour = 90;
4 System.out.println(hours * minutesPerHour);
```

Remove that line and the code prints 180.

WATCH OUT

Trying to change a constant from outside. Because a class constant is `public`, a student may think they are allowed to adjust it, for example to model a bigger pizza:

FinalChange.java

```
1 class PizzaOrder
2 {
3     public static final int SLICES_PER_PIZZA = 8;
4     private int pizzas;
5
6     public PizzaOrder(int pizzaCount)
7     {
8         pizzas = pizzaCount;
9     }
10
11     public int getSlices()
12     {
13         return pizzas * SLICES_PER_PIZZA;
14     }
15 }
16
17 public class FinalChange
18 {
19     public static void main(String[] args)
20     {
21         PizzaOrder.SLICES_PER_PIZZA = 10;
22         PizzaOrder party = new PizzaOrder(4);
23         System.out.println(party.getSlices());
24     }
25 }
```

Line 21 does not compile. JDK 26 reports:

```
FinalChange.java:21: error: cannot assign a value to static final variable
SLICES_PER_PIZZA
```

(The exact wording can vary between JDK versions.) `public` controls who may **read** the constant; `final` stops **everyone** from changing it, including code inside its own class. If a value really needs to vary, it should not be `final`; for example, the number of slices could be a constructor parameter stored in an instance variable.

PAUSE AND TRY

Question 5 Multiple choice

This program does not compile. Which line causes the error?

StudentIdChange.java

```
1  class Student
2  {
3      private final int idNumber;
4      private String nickname;
5
6      public Student(int id, String startingNickname)
7      {
8          idNumber = id;
9          nickname = startingNickname;
10     }
11
12     public void setNickname(String newNickname)
13     {
14         nickname = newNickname;
15     }
16
17     public void setIdNumber(int newId)
18     {
19         idNumber = newId;
20     }
21 }
22
23 public class StudentIdChange
24 {
25     public static void main(String[] args)
26     {
27         Student kim = new Student(4417, "Kimmy");
28         kim.setNickname("K");
29         kim.setIdNumber(1000);
30     }
31 }
```

- A. Line 8, `idNumber = id;`
- B. Line 14, `nickname = newNickname;`
- C. Line 19, `idNumber = newId;`
- D. Line 29, `kim.setIdNumber(1000);`

Question 6 Short answer

A `GradeBook` class needs the value `100`, the most points any test can be worth, and the value is the same for every gradebook. Explain why it makes sense to declare it as `public static final int MAX_POINTS = 100;`, giving a reason for each of the three keywords.

3.7.4 Writing Class Methods

In 3.7.2, making the ticket count `private` meant `main` could not read it. An instance accessor would work, but only if you already had a ticket object to call it on, and the count is not about any one ticket. You need a method that, like the variable, belongs to the class.

In Topic 1.10 you called class methods such as `Math.sqrt` and wrote a few in the class that holds `main`. Now you will write them in your own object classes. A class method is written like any method from Topic 3.5, with `static` added before the return type:

```
1 public static int getTicketsSold()
```

A class method may use the class's **class variables**, reading them or changing them, and may call the class's other **class methods**. Everything it uses belongs to the class, so no object is needed to run it. From another class you call it as `ClassName.methodName(arguments)`; inside its own class the class name may be left off, exactly as in Topic 1.10.

KEY IDEA

A class method has `static` in its header. It can read and change class variables and call other class methods, and it is called through the class name, with no object needed.

WORKED EXAMPLE A private count with class methods

`ticketsSold` is `private` again, and two class methods give controlled access: one reads it, one resets it.

RaffleReport.java

```
1 class Raffle
2 {
3     private static int ticketsSold = 0;
4     private String buyer;
5
6     public Raffle(String buyerName)
7     {
8         buyer = buyerName;
9         ticketsSold++;
10    }
11
12    public static int getTicketsSold()
13    {
14        return ticketsSold;
15    }
```

```

16
17     public static void resetSales()
18     {
19         ticketsSold = 0;
20     }
21 }
22
23 public class RaffleReport
24 {
25     public static void main(String[] args)
26     {
27         System.out.println(Raffle.getTicketsSold());
28         Raffle first = new Raffle("Jo");
29         Raffle second = new Raffle("Sam");
30         Raffle third = new Raffle("Lee");
31         System.out.println(Raffle.getTicketsSold());
32         Raffle.resetSales();
33         System.out.println(Raffle.getTicketsSold());
34     }
35 }

```

- Lines 12 to 15: `getTicketsSold` is a class method that **reads** a class variable. It is an accessor, as in Topic 3.5, but because it is `static` it needs no object.
- Lines 17 to 20: `resetSales` is a void class method that **changes** a class variable. It is a mutator for the class's data.

Line	What happens	ticketsSold	Printed
27	<code>Raffle.getTicketsSold()</code> , no objects yet	0	0
28–30	three tickets made; each constructor adds 1	3	
31	<code>Raffle.getTicketsSold()</code>	3	3
32	<code>Raffle.resetSales()</code> sets the count to 0	0	
33	<code>Raffle.getTicketsSold()</code>	0	0

```

0
3
0

```

Compare this with `RaffleSales` in 3.7.2: the count can still be reset, but only through a method the `Raffle` class chose to provide, not by any code writing to the variable.

Now vary it: a class with **no** instance variables at all, where one class method calls another. `GradeScale` works like `Math`: it is a toolbox of class methods.

GradeCheck.java

```
1 class GradeScale
2 {
3     public static final int PASSING = 70;
4
5     public static double percent(int earned, int possible)
6     {
7         return 100.0 * earned / possible;
8     }
9
10    public static boolean passed(int earned, int possible)
11    {
12        return percent(earned, possible) >= PASSING;
13    }
14 }
15
16 public class GradeCheck
17 {
18     public static void main(String[] args)
19     {
20         System.out.println(GradeScale.percent(45, 50));
21         System.out.println(GradeScale.passed(45, 50));
22         System.out.println(GradeScale.passed(30, 50));
23     }
24 }
```

- percent (lines 5 to 8) computes $100.0 * \text{earned} / \text{possible}$. Starting with 100.0 makes the arithmetic double, so no integer division happens.
- passed (lines 10 to 13) **calls another class method**, percent, without a class name because both are in GradeScale. It compares the result with the class constant PASSING.

Line 20 prints $100.0 * 45 / 50$, which is $4500.0 / 50$, or 90.0. Line 21 calls `passed(45, 50)`: percent returns 90.0, and $90.0 \geq 70$ is true. Line 22: `percent(30, 50)` is 60.0, and $60.0 \geq 70$ is false.

```
90.0
true
false
```

WATCH OUT

Forgetting static on the method. Suppose `getTicketsSold` is written

`public int getTicketsSold()`, without static, but main still calls `Raffle.getTicketsSold()`:

RaffleMissingStatic.java

```
1 class Raffle
2 {
3     private static int ticketsSold = 0;
4     private String buyer;
5
6     public Raffle(String buyerName)
7     {
```

```
8         buyer = buyerName;
9         ticketsSold++;
10    }
11
12    public int getTicketsSold()
13    {
14        return ticketsSold;
15    }
16
17    public static void resetSales()
18    {
19        ticketsSold = 0;
20    }
21 }
22
23 public class RaffleMissingStatic
24 {
25     public static void main(String[] args)
26     {
27         System.out.println(Raffle.getTicketsSold());
28         Raffle first = new Raffle("Jo");
29         Raffle second = new Raffle("Sam");
30         Raffle third = new Raffle("Lee");
31         System.out.println(Raffle.getTicketsSold());
32         Raffle.resetSales();
33         System.out.println(Raffle.getTicketsSold());
34     }
35 }
```

The compiler reports, at line 27 (and again at lines 31 and 33):

```
RaffleMissingStatic.java:27: error: non-static method getTicketsSold() cannot be
referenced from a static context
```

Without `static`, `getTicketsSold` is an instance method, so it must be called on an object. `Raffle.getTicketsSold()` names only the class, not an object. Topic 1.10 showed the same message when `main` called a non-static method. The fix is to add `static` to the header, since the method uses only a class variable.

PAUSE AND TRY

Question 7 Tracing

Write the exact output.

ParkingLot.java

```
1 class Garage
2 {
3     public static final int TOTAL_SPOTS = 5;
4     private static int spotsTaken = 0;
5 }
```

```
6     public static void park()
7     {
8         spotsTaken++;
9     }
10
11    public static void leave()
12    {
13        spotsTaken--;
14    }
15
16    public static int spotsOpen()
17    {
18        return TOTAL_SPOTS - spotsTaken;
19    }
20
21    public static boolean isFull()
22    {
23        return spotsOpen() == 0;
24    }
25 }
26
27 public class ParkingLot
28 {
29     public static void main(String[] args)
30     {
31         Garage.park();
32         Garage.park();
33         Garage.park();
34         Garage.leave();
35         System.out.println(Garage.spotsOpen());
36         Garage.park();
37         Garage.park();
38         Garage.park();
39         System.out.println(Garage.isFull());
40     }
41 }
```

Question 8 Multiple choice

A `Library` class has `private static int booksOut = 0;`. Which method, added to `Library`, compiles and can be called as `Library.checkOut()` with no `Library` object?

- A. `public void checkOut() { booksOut++; }`
- B. `public static void checkOut() { booksOut++; }`
- C. `public static checkOut() { booksOut++; }`
- D. `static public int checkOut() { booksOut++; }`

3.7.5 What Class Methods Cannot Reach

You can call a class method with no objects in existence. That raises a question: what happens if a class method tries to use an **instance** variable, such as a member's `name`? Every `Member` object has

its own `name`. Whose name should the method use? There might be a hundred members, or none at all.

Java does not guess. Inside a class method there is no particular object, so the method cannot use an instance variable or call an instance method by name alone. The only way to work with an object's data is to be **given** an object, through a parameter. Then the method names the object it means: `someone.getName()` or `someone.name`. Recall from Topic 3.6 that a method inside a class may even use the private data of a parameter of that same class.

KEY IDEA

A class method cannot read or change instance variables, or call instance methods, unless it is passed an object of the class through a parameter and uses that object.

WORKED EXAMPLE Giving a class method an object to work on

First, the attempt that fails. `describe` is `static`, and it tries to use `name`:

`StaticReadsInstance.java`

```
1  class Member
2  {
3      private String name;
4
5      public Member(String memberName)
6      {
7          name = memberName;
8      }
9
10     public String getName()
11     {
12         return name;
13     }
14
15     public static String describe()
16     {
17         return "Member: " + name;
18     }
19 }
20
21 public class StaticReadsInstance
22 {
23     public static void main(String[] args)
24     {
25         Member ana = new Member("Ana");
26         System.out.println(Member.describe());
27     }
28 }
```

`StaticReadsInstance.java:17: error: non-static variable name cannot be referenced from a static context`

"Static context" means "inside a class method, where there is no current object." Line 26 calls `Member.describe()` with no object in sight, so the method has no `name` it could use. Calling the

instance method instead, `return "Member: " + getName();`, fails the same way: non-static method `getName()` cannot be referenced from a static context.

The fix is a parameter that tells the method which member to describe:

MemberDescribe.java

```
1  class Member
2  {
3      private String name;
4
5      public Member(String memberName)
6      {
7          name = memberName;
8      }
9
10     public String getName()
11     {
12         return name;
13     }
14
15     public static String describe(Member someone)
16     {
17         return "Member: " + someone.getName();
18     }
19
20     public static void rename(Member someone, String newName)
21     {
22         someone.name = newName;
23     }
24 }
25
26 public class MemberDescribe
27 {
28     public static void main(String[] args)
29     {
30         Member ana = new Member("Ana");
31         Member ben = new Member("Ben");
32         System.out.println(Member.describe(ana));
33         Member.rename(ben, "Benji");
34         System.out.println(Member.describe(ben));
35     }
36 }
```

- Line 17: `describe` calls the instance method `getName` **on the parameter**, `someone.getName()`. Now Java knows whose name to get.
- Line 22: `rename` **changes** an instance variable through its parameter. `someone.name` is private, but `rename` is inside `Member` and `someone` is a `Member`, so it is allowed (Topic 3.6).
- Line 33 passes `ben`, so `someone` refers to Ben's object and his name becomes "Benji".

```
Member: Ana
Member: Benji
```

Now vary the number of objects. A class method can receive **two** objects of its class and compare them:

HigherScorer.java

```
1 class Player
2 {
3     private String name;
4     private int points;
5
6     public Player(String playerName, int startingPoints)
7     {
8         name = playerName;
9         points = startingPoints;
10    }
11
12    public String getName()
13    {
14        return name;
15    }
16
17    public static Player higherScorer(Player first, Player second)
18    {
19        if (first.points >= second.points)
20        {
21            return first;
22        }
23        return second;
24    }
25 }
26
27 public class HigherScorer
28 {
29     public static void main(String[] args)
30     {
31         Player maya = new Player("Maya", 12);
32         Player leo = new Player("Leo", 15);
33         Player winner = Player.higherScorer(maya, leo);
34         System.out.println(winner.getName());
35     }
36 }
```

higherScorer reads `first.points` and `second.points` (both allowed, since the parameters are `Players` and the method is in `Player`). On line 33, `first` refers to Maya (12) and `second` to Leo (15). `12 >= 15` is `false`, so line 23 returns the reference in `second`, Leo's object, and `main` prints:

Leo

This also explains something you have done since Unit 1. `main` is a class method, so it has no object of its own; that is why it always makes an object first and then calls instance methods on it, like `ana.getMemberCount()`. Topic 3.9 adds a keyword for referring to the current object inside instance methods and constructors; class methods have no current object for it to refer to.

Code inside a class method	Allowed?
Read or change a class variable	Yes
Call another class method	Yes

Use an instance variable by name alone	No: non-static variable error
Call an instance method by name alone	No: non-static method error
Use an instance variable or method through a parameter	Yes: someone.name, someone.getName()

The reverse direction has no such limit: instance methods and constructors may use class variables freely, as `Member`'s constructor did with `memberCount`.

WATCH OUT

Making a method static just to call it without an object. A student sees that `Member.describe()` is easy to call and adds `static` to a method that needs one member's data. The error above is the result. Before writing `static`, ask: "Does this method need one object's instance variables?" If yes, either leave it an instance method (called as `ana.describe()`) or give it a parameter that receives the object.

PAUSE AND TRY

Question 9 Debugging

This class does not compile. Explain the error and fix `isTooHot` so that it can still be called through the class name (as `Thermostat.isTooHot(...)`). Also give the corrected call in `main`.

`ThermostatBug.java`

```
1 class Thermostat
2 {
3     public static final int MAX_COMFORT = 76;
4     private int temperature;
5
6     public Thermostat(int startingTemp)
7     {
8         temperature = startingTemp;
9     }
10
11     public static boolean isTooHot()
12     {
13         return temperature > MAX_COMFORT;
14     }
15 }
16
17 public class ThermostatBug
18 {
19     public static void main(String[] args)
20     {
21         Thermostat kitchen = new Thermostat(80);
22         System.out.println(Thermostat.isTooHot());
23     }
24 }
```

Question 10 Multiple choice

A `Lamp` class has an instance variable `isOn`, a class variable `lampsOn`, an instance method `turnOn()`, and a class method `getLampsOn()`. A new class method with **no parameters** is added to `Lamp`. Which of these can its body do?

- A. Read `isOn`
- B. Call `turnOn()`
- C. Call `getLampsOn()`
- D. Set `isOn` to `false`

Putting it together

A food truck numbers its orders starting at 101. Each `Order` remembers its own number and cost, the class keeps the next number to hand out and the number of orders placed, and every order pays the same delivery fee.

`OrderNumbers.java`

```
1 class Order
2 {
3     public static final double DELIVERY_FEE = 2.5;
4     private static int nextNumber = 101;
5     private static int ordersPlaced = 0;
6
7     private final int number;
8     private double foodCost;
9
10    public Order(double cost)
11    {
12        foodCost = cost;
13        number = nextNumber;
14        nextNumber++;
15        ordersPlaced++;
16    }
17
18    public int getNumber()
19    {
20        return number;
21    }
22
23    public double getTotal()
24    {
25        return foodCost + DELIVERY_FEE;
26    }
27
28    public static int getOrdersPlaced()
29    {
30        return ordersPlaced;
31    }
32
33    public static Order larger(Order first, Order second)
```

```

34     {
35         if (first.getTotal() >= second.getTotal())
36         {
37             return first;
38         }
39         return second;
40     }
41 }
42
43 public class OrderNumbers
44 {
45     public static void main(String[] args)
46     {
47         System.out.println(Order.getOrdersPlaced());
48         Order tacos = new Order(12.0);
49         Order salad = new Order(8.5);
50         Order pasta = new Order(15.25);
51         System.out.println(Order.getOrdersPlaced());
52         System.out.println(salad.getNumber());
53         System.out.println(salad.getTotal());
54         Order biggest = Order.larger(tacos, pasta);
55         System.out.println(biggest.getNumber());
56         System.out.println(Order.DELIVERY_FEE);
57     }
58 }

```

Sort the variables first:

Line	Variable	Kind	Why
3	DELIVERY_FEE	class constant	same for every order and never changes
4	nextNumber	class variable	one counter for the whole truck
5	ordersPlaced	class variable	a total about all orders
7	number	final instance variable	each order's own number, set once
8	foodCost	instance variable	each order's own cost

The constructor (lines 10 to 16) copies the shared `nextNumber` into this order's `number`, then moves the shared counter forward, so the next order gets the next number. Trace `main`:

Line	What happens	Shared afterward	Printed
47	class method call, no orders yet	nextNumber 101, ordersPlaced 0	0
48	tacos gets number 101, cost 12.0	nextNumber 102, ordersPlaced 1	
49	salad gets number 102, cost 8.5	nextNumber 103, ordersPlaced 2	
50	pasta gets number 103, cost 15.25	nextNumber 104, ordersPlaced 3	
51	<code>Order.getOrdersPlaced()</code>	unchanged	3
52	<code>salad.getNumber()</code>	unchanged	102
53	<code>salad.getTotal(): 8.5 + 2.5</code>	unchanged	11.0
55	number of the larger order	unchanged	103
56	<code>Order.DELIVERY_FEE</code>	unchanged	2.5

On line 54, the class method `larger` receives two orders. It cannot use `getTotal()` by name alone, so it calls it on each parameter: `tacos` totals 14.5 and `pasta` totals 17.75. `14.5 >= 17.75` is false, so it returns `second`, the `pasta` order, whose number is 103.

```
0
3
102
11.0
103
2.5
```

Every idea from the chapter appears: shared class variables (`nextNumber`, `ordersPlaced`) changed by the constructor, a public class constant read with the class name, a `final` instance variable set once, a class method that reads a class variable, and a class method that works on objects only through its parameters.

Common mistakes

The Watch Out boxes explain each mistake. This table is a lookup for the compiler messages you will see in this topic (wording can vary slightly between JDK versions).

Message contains	Usual cause	Fix
non-static variable ... cannot be referenced from a static context	a class method uses an instance variable by name alone	pass an object and use <code>param.variable</code> , or remove <code>static</code>
non-static method ... cannot be referenced from a static context	a class method (or <code>ClassName.method()</code>) calls an instance method with no object	call it on an object, or add <code>static</code> if it uses only class data
... has private access in ...	another class uses a private class variable	use a class method that returns it
cannot assign a value to ... final variable ...	a statement assigns to a <code>final</code> variable after it has its value	remove the assignment, or do not make the variable <code>final</code>
cannot find symbol	another class uses a class variable without the class name	write <code>ClassName.variableName</code>

A counter declared without `static` gives no message at all: it compiles and counts only to 1.

Vocabulary

Term	Definition	Example
class variable	A variable that belongs to the class, with one copy shared by all objects of the class	<code>private static int memberCount = 0;</code>
static	The keyword before a variable's type or a method's return type that makes it belong to the class	<code>public static int getTicketsSold()</code>

final	The keyword that makes a variable unchangeable once it has its value	<code>private final int idNumber;</code>
final variable	A variable declared <code>final</code> ; any later assignment to it is a compile error	<code>final int minutesPerHour = 60;</code>
class constant	A public static final variable: one shared, unchangeable value	<code>PizzaOrder.SLICES_PER_PIZZA</code>

Chapter summary

- A **class variable** is declared with `static` before its type. It belongs to the class, so every object shares one copy, and it exists even before any object is made. Give it its starting value in the declaration, not in the constructor.
- A public class variable is used from another class as `ClassName.variableName`. Inside its own class, its name alone is enough. Data is still usually kept `private`.
- A `final` variable cannot be changed after it gets its value. A public static final variable is a class constant, written in capital letters, such as `Integer.MAX_VALUE`.
- A class method has `static` in its header, is called through the class name with no object, and can read and change class variables and call other class methods.
- A class method cannot use instance variables or instance methods by name alone, because there is no current object. Pass it an object through a parameter and work through that object. Instance methods and constructors may always use class variables.

Review questions

Question 11 Multiple choice

Which statement is true?

- A. Each object of a class has its own copy of every class variable.
- B. A class variable can be used from another class as `ClassName.variableName` only if it is `public`.
- C. A `final` class variable can be changed by the class's own methods, but not by other classes.
- D. A class method can use any instance variable of its class by name alone.

Question 12 Tracing

Write the exact output. The loop is from Topic 2.8.

GroupVisits.java

```

1 class Visit
2 {
3     private static int visitCount = 0;
4     private static int totalPeople = 0;
5     private int groupSize;
6
7     public Visit(int people)
8     {
9         groupSize = people;
10        visitCount++;
11        totalPeople += people;

```

```
12     }
13
14     public static double averageGroup()
15     {
16         return (double) totalPeople / visitCount;
17     }
18
19     public static int getVisitCount()
20     {
21         return visitCount;
22     }
23 }
24
25 public class GroupVisits
26 {
27     public static void main(String[] args)
28     {
29         for (int i = 1; i <= 4; i++)
30         {
31             Visit v = new Visit(i * 2);
32         }
33         System.out.println(Visit.getVisitCount());
34         System.out.println(Visit.averageGroup());
35     }
36 }
```

Question 13 Short answer

A classmate is tired of "non-static" errors and suggests: "In the `Student` class, just make every variable `static`, including each student's `gpa`. Then every method can use every variable." Explain what would go wrong with the program's data.

Question 14 Debugging

Each badge should get the next ID number: 1, 2, 3. The program compiles but prints 1, 1, 1. Find the error and write the corrected line.

BadgeBug.java

```
1 class Badge
2 {
3     private int nextId = 1;
4     private int id;
5
6     public Badge()
7     {
8         id = nextId;
9         nextId++;
10    }
11
12    public int getId()
13    {
14        return id;
15    }
16 }
17
18 public class BadgeBug
19 {
20     public static void main(String[] args)
21     {
22         Badge first = new Badge();
23         Badge second = new Badge();
24         Badge third = new Badge();
25         System.out.println(first.getId());
26         System.out.println(second.getId());
27         System.out.println(third.getId());
28     }
29 }
```

Question 15 Multiple choice

A Lamp class has private static int lampsOn and private boolean isOn. Which method, written inside Lamp, does **not** compile?

- A. public void turnOn() { isOn = true; lampsOn++; }
- B. public static int getLampsOn() { return lampsOn; }
- C. public static boolean check(Lamp someLamp) { return someLamp.isOn; }
- D. public static void turnOn() { isOn = true; lampsOn++; }

Question 16 Tracing

Write the exact output. Pay attention to line 35.

ScoreBoost.java

```
1 class Quiz
2 {
3     public static final int BONUS = 5;
4     private static int quizzesGraded = 0;
5     private int score;
6
7     public Quiz(int startingScore)
8     {
9         score = startingScore;
```

```
10         quizzesGraded++;
11     }
12
13     public int getScore()
14     {
15         return score;
16     }
17
18     public static void addBonus(Quiz target)
19     {
20         target.score += BONUS;
21     }
22
23     public static int getQuizzesGraded()
24     {
25         return quizzesGraded;
26     }
27 }
28
29 public class ScoreBoost
30 {
31     public static void main(String[] args)
32     {
33         Quiz first = new Quiz(80);
34         Quiz second = new Quiz(91);
35         Quiz same = first;
36         Quiz.addBonus(first);
37         Quiz.addBonus(same);
38         Quiz.addBonus(second);
39         System.out.println(first.getScore());
40         System.out.println(second.getScore());
41         System.out.println(Quiz.getQuizzesGraded());
42     }
43 }
```

Reflection

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

- ☐ I can declare a class variable with static and explain why all objects share one copy. Reread: [3.7.1 Class Variables Are Shared](#)
- ☐ I can use a public class variable from another class with the class name and a dot. Reread: [3.7.2 Public Class Variables and the Class Name](#)
- ☐ I can explain what final does and write a class constant. Reread: [3.7.3 Final Variables Cannot Change](#)
- ☐ I can write a class method that reads or changes class variables and calls other class methods. Reread: [3.7.4 Writing Class Methods](#)
- ☐ I can explain why a class method needs an object parameter to use instance data, and fix code that breaks this rule. Reread: [3.7.5 What Class Methods Cannot Reach](#)

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

B. A class variable has `static` written before its type, and `static` is what makes the copy shared.

- **A** is wrong: with no `static`, each `Ticket` gets its own `totalPrinted`, so the objects would not share a total.
- **C** is wrong: `static` must come before the type `int`, not between the type and the name. This does not compile.
- **D** is wrong: it is missing the type. Every variable declaration needs a type such as `int`.

Reread 3.7.1 *Class Variables Are Shared* if you chose differently.

Question 2

```
6
2
8
```

`rides` is an instance variable (each bike has its own), and `totalRides` is a class variable (one shared copy).

Line and statement	<code>red.rides</code>	<code>blue.rides</code>	<code>totalRides</code>
33 <code>new Bike(5)</code>	5		5
34 <code>new Bike(0)</code>	5	0	5
35 <code>blue.ride()</code>	5	1	6
36 <code>red.ride()</code>	6	1	7
37 <code>blue.ride()</code>	6	2	8

`red.getTotalRides()` returns the shared 8, which includes blue's rides. A common wrong answer is 6 for the last line, which treats `totalRides` as red's own.

Reread 3.7.1 *Class Variables Are Shared* if your output differed.

Question 3

Line 18 should be:

```
1      System.out.println(Raffle.ticketsSold);
```

What was wrong: `ticketsSold` is declared in `Raffle`, but line 18 is in `RaffleNoClassName`, a different class. From outside the declaring class, a public class variable must be reached through the class name and a dot. Without it, the compiler reports error: `cannot find symbol` because there is no variable named `ticketsSold` in `main` or in `RaffleNoClassName`. With the fix, one `Raffle` has been made, so the program prints 1.

Reread 3.7.2 *Public Class Variables and the Class Name* if you found a different error.

Question 4

C. `visitors` is a class variable, so all `Museum` objects share one copy. `dinosaurs.enter()` raises it to 1, and `space.enter()` twice raises it to 3. `Museum.visitors` reads that one copy.

- **A** is wrong: 1 is how many times `dinosaurs` called `enter`, as if each object had its own count.
- **B** is wrong: 2 is how many times `space` called `enter`; it ignores the call made through `dinosaurs`.
- **D** is wrong: a public class variable is reached through the class name, which is exactly what `Museum.visitors` does.

Reread 3.7.2 *Public Class Variables and the Class Name* if you chose differently.

Question 5

C. `idNumber` is `final`, so after the constructor gives it its value, no statement may assign to it again. Line 19 tries, so the compiler reports cannot assign a value to final variable `idNumber`.

- **A** is wrong: a `final` instance variable with no value in its declaration must get its value in the constructor. Line 8 is that one allowed assignment.
- **B** is wrong: `nickname` is not `final`, so a mutator may change it.
- **D** is wrong: calling a method is fine. The problem is inside the method, on line 19, and the compiler reports it there. Removing the `setIdNumber` method (and this call) fixes the program.

Reread 3.7.3 *Final Variables Cannot Change* if you chose differently.

Question 6

Model answer: `static` because the value is the same for every `GradeBook`, so one shared copy that belongs to the class is enough; storing a copy in each object would be wasteful. `final` because the value should never change, and Java will then refuse any statement that tries to change it. `public` because other classes may need to read it as `GradeBook.MAX_POINTS`, and making it public is safe: since it is `final`, no one outside can damage it.

A complete answer must:

- connect `static` to "one value shared by all objects, belonging to the class,"
- connect `final` to "cannot be changed," and
- explain that `public` is safe here because `final` prevents changes.

Reread 3.7.3 *Final Variables Cannot Change* if your answer differed.

Question 7

3
true

Every method is a class method that changes or reads the one class variable `spotsTaken`.

Lines	Calls	spotsTaken
31-33	<code>park()</code> three times	3
34	<code>leave()</code>	2
35	<code>spotsOpen()</code> returns <code>5 - 2</code>	2, prints 3
36-38	<code>park()</code> three times	5

Line 39 calls `isFull()`, which calls the class method `spotsOpen()`. That returns `5 - 5`, which is `0`, and `0 == 0` is `true`.

Reread 3.7.4 *Writing Class Methods* if your output differed.

Question 8

B. It is `static`, so it can be called through the class name, and a class method may change a class variable.

- **A** is wrong: it compiles, but without `static` it is an instance method, so `Library.checkOut()` gives non-static method `checkOut()` cannot be referenced from a static context.
- **C** is wrong: there is no return type. Every method header needs one (`void` here).
- **D** is wrong: the return type is `int`, but the body never returns a value, so it does not compile.

Reread 3.7.4 *Writing Class Methods* if you chose differently.

Question 9

The compiler reports

non-static variable `temperature` cannot be referenced from a static context at line 13.

`isTooHot` is a class method, so it has no particular thermostat; `temperature` belongs to each object, and Java cannot know whose temperature to use. Give the method a `Thermostat` parameter and use its temperature:

```
1    public static boolean isTooHot(Thermostat room)
2    {
3        return room.temperature > MAX_COMFORT;
4    }
```

and in `main`:

```
1    System.out.println(Thermostat.isTooHot(kitchen));
```

`room.temperature` is allowed even though it is private, because the method is inside `Thermostat` and `room` is a `Thermostat`. Using `MAX_COMFORT` is fine either way: it is a class variable. With the fix, the kitchen is at 80, `80 > 76` is `true`, and the program prints `true`.

Reread 3.7.5 *What Class Methods Cannot Reach* if you found a different error.

Question 10

C. A class method may call another class method; neither one needs an object.

- **A** is wrong: `isOn` is an instance variable. With no parameter, the method has no lamp whose `isOn` it could read, so this is a non-static variable error.
- **B** is wrong: `turnOn` is an instance method and must be called on a particular lamp. With no object, this is a non-static method error.
- **D** is wrong: changing an instance variable needs an object just as reading one does.

Reread 3.7.5 *What Class Methods Cannot Reach* if you chose differently.

Question 11

B. A public class variable is reached from outside through the class name; a private one cannot be reached from outside at all (has private access).

- **A** is wrong: that describes instance variables. A class variable has exactly one copy, shared by all objects.
- **C** is wrong: `final` stops everyone from changing the variable, including code inside its own class.
- **D** is wrong: a class method has no current object, so it can use an instance variable only through an object passed as a parameter.

Reread 3.7.2 *Public Class Variables and the Class Name* if you chose differently.

Question 12

```
4
5.0
```

Each pass of the loop makes a new `visit` with `i * 2` people. Only one copy of each class variable exists, so every constructor adds to the same totals:

i	people	visitCount	totalPeople
1	2	1	2
2	4	2	6
3	6	3	12
4	8	4	20

The variable `v` is replaced on each pass, but that does not matter: the totals live in the class, not in any object. `averageGroup` casts `totalPeople` to `double` before dividing, so the result is `20.0 / 4`, which is `5.0`.

Reread 3.7.1 *Class Variables Are Shared* and 3.7.4 *Writing Class Methods* if your output differed.

Question 13

Model answer: If `gpa` is `static`, there is only one `gpa` for the whole `Student` class, shared by every student object. Setting one student's GPA would change it for all students, and every student would report the same value, whichever was stored last. The error messages are not the real problem; they point out that a method needs a particular object's data. The right fix is to keep `gpa` an instance variable and either call the method on a student object or pass the student to a class method as a parameter.

A complete answer must:

- say that a `static` variable has only one shared copy, so every student would have the same GPA, and
- explain that values that differ from object to object must stay instance variables.

Reread 3.7.1 *Class Variables Are Shared* and 3.7.5 *What Class Methods Cannot Reach* if your answer differed.

Question 14

Line 3 should be:

```
1    private static int nextId = 1;
```

What was wrong: `nextId` is meant to be one counter for the whole class, but without `static` it is an instance variable. Each new `Badge` gets its own `nextId` starting at 1, copies that 1 into `id`, and then raises only its own `nextId` to 2, which no other badge ever sees. With `static`, all badges share one `nextId`, so the first badge takes 1 and leaves 2, the second takes 2 and leaves 3, and the third takes 3. The program then prints 1, 2, 3.

`id` must stay an instance variable, because each badge needs its own.

Reread 3.7.1 *Class Variables Are Shared* if you found a different error.

Question 15

D. It is a class method, and it assigns to the instance variable `isOn` by name alone. With no object, Java reports non-static variable `isOn` cannot be referenced from a static context.

- **A** compiles: an instance method may use both its own object's instance variables and the class variables.
- **B** compiles: a class method may read a class variable.
- **C** compiles: a class method may use an instance variable through a parameter, and `someLamp.isOn` is allowed even though it is private because `check` is inside `Lamp`.

Reread 3.7.5 *What Class Methods Cannot Reach* if you chose differently.

Question 16

```
90
96
2
```

- Lines 33 and 34 make two `Quiz` objects, so `quizzesGraded` becomes 2.
- Line 35 makes **no** new object. As in Topic 3.6, `same` gets a copy of the reference in `first`, so both variables refer to the same quiz. The constructor does not run, and `quizzesGraded` stays 2.
- `addBonus` is a class method, so it changes `score` only through its parameter. Line 36 adds 5 to the first quiz (85). Line 37 passes `same`, which is that same quiz, so it becomes 90. Line 38 adds 5 to the second quiz (96).

A common wrong answer is 85 and 3, which treats `same` as a separate quiz.

Reread 3.7.5 *What Class Methods Cannot Reach* and 3.7.1 *Class Variables Are Shared* if your output differed.