

Unit 5: Writing Classes

Unit 5 is where you stop using classes written by other people and start writing your own. It covers the anatomy of a class, constructors, accessor and mutator methods, static members, scope, and the `this` keyword. These are the tools behind FRQ 2, the class-design question, so this unit carries weight far beyond its share of the multiple-choice section.

AP Computer Science A

Writing Classes

About 16 minutes to read

How to use this guide

Read it in order the first time because the topics build on each other. Fields come before constructors, constructors come before the methods that use the objects they build, and scope makes sense only after you have seen fields, parameters, and local variables side by side. Exam questions usually hand you a class and ask you to trace what it does, so read every code sample slowly and predict the output before you read on.

After the first read, use the trap boxes and the confusions table to review the distinctions that exam questions test most often. Finish with the practice questions, then complete the recall check on the last page out loud and note any items you cannot explain yet.

What this unit is worth. Unit 5 is about 5 to 7.5 percent of the AP CSA exam as tested directly, but it underpins FRQ 2 (Class Design) every year, which is 9 points. You cannot write the class-design FRQ without constructors, accessors, mutators, and correct public/private choices.

5.1 Anatomy of a Class

A class is a blueprint for objects. It describes what data each object holds and what each object can do. The data is stored in instance variables, also called fields. The behavior is defined by methods. A constructor is the special method that runs when an object is created and sets the fields to their starting values.

Classes in this course use two access modifiers. Fields are declared **private** so that only code inside the class can touch them directly. Methods that other classes should use are declared **public**. This split is called encapsulation. The private fields are the hidden implementation, and the public methods are the interface the rest of the program is allowed to use.

```
public class Dog {  
    private String name;    // field: data each Dog holds  
    private int age;        // field  
  
    public Dog(String name, int age) { // constructor  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() { // method: behavior  
        return name;  
    }  
}
```

Notice the pattern. The fields sit at the top and are private. The constructor has the same name as the class and no return type. The method `getName` is public and returns the field. It is completely fine for methods inside the class to use its private fields directly. The rule is only that code outside the class cannot reach them.

Trap. Private does not mean invisible to the class itself. Methods inside the class can read and write its private fields freely. Private means invisible to other classes. An accessor like `getName` is needed only when outside code wants the value.

5.2 Constructors

A constructor creates and initializes a new object. Three rules define it. It has the same name as the class. It has no return type, not even void. And it usually assigns starting values to the fields, often from its parameters.

If you write no constructor at all, Java provides a **default constructor**. It takes no arguments and sets numeric fields to 0, boolean fields to false, and object references to null. The moment you write any constructor of your own, the default constructor is gone. Code that calls `new Dog()` will no longer compile unless you wrote a no-argument constructor yourself.

```
public class Book {
    private String title;
    private int pages;

    public Book(String title, int pages) {
        this.title = title;
        this.pages = pages;
    }

    public Book(String title) {    // overloaded constructor
        this.title = title;
        this.pages = 100;
    }
}
```

The two constructors above are overloaded. Overloaded constructors share the class name but take different parameter lists, and Java picks the one whose arguments match the call. `new Book("Dune")` uses the second constructor and pages becomes 100. `new Book("Dune", 412)` uses the first.

Trap. A constructor with void is not a constructor. `public void Dog()` is a regular method that happens to share the class name. It compiles, it runs like any method, and it never runs when you write `new Dog()`. If you see a return type on something shaped like a constructor, it is a method.

5.3 Documentation with Comments

Comments explain code to humans and are ignored by the compiler. Single-line comments start with `//` and multi-line comments sit between `/*` and `*/`. For classes and public methods, Java uses documentation comments that start with `/**`. These can include tags like `@param` to describe a parameter and `@return` to describe the returned value, and tools can turn them into formatted documentation.

On the exam, documentation is tested lightly. What matters is that you can read a doc comment and that any public method you write on an FRQ has a clear purpose. A one-line `//` comment above a tricky section of your own FRQ code is good practice.

5.4 Accessor Methods

An **accessor method**, usually called a getter, gives outside code controlled read access to a private field. It takes no parameters and returns the field's value, so its return type matches the field's type. The naming pattern is `get` followed by the capitalized field name, as in `getName` for the field `name`.

```
public String getName() {  
    return name;  
}  
  
public int getAge() {  
    return age;  
}
```

An accessor can also return a computed value rather than a raw field. A method `getArea` that returns `length * width` is still an accessor in spirit, because it reads state without changing it.

5.5 Mutator Methods

A **mutator method**, usually called a setter, gives outside code controlled write access to a private field. Its return type is void, because its job is to change state rather than report it. It takes one parameter holding the new value. The naming pattern is `set` followed by the capitalized field name.

```
public void setAge(int newAge) {  
    if (newAge >= 0) {  
        age = newAge;  
    }  
}
```

Mutators are where validation lives. The `setAge` above refuses negative ages instead of letting bad data into the object. This is the payoff of private fields. Because outside code cannot assign `age` directly, every change goes through the setter, and the setter can enforce the rules.

Trap. A mutator that returns a value is a red flag. `public int setAge(int a)` compiles only if it returns an int somewhere, and an exam question may use it to check whether you know the setter pattern is void. Getters return a value. Setters take a parameter and return void.

5.6 Writing Methods

A method signature has three parts. The return type, which is void when nothing is returned. The method name. And the parameter list, which names the type and name of each input. The body then does the work, and any method with a non-void return type must return a value of that type on every path through the method.

```
public double average(int a, int b) {  
    return (a + b) / 2.0;  
}
```

Parameters are input variables that exist only while the method runs. The call `average(4, 6)` copies 4 into `a` and 6 into `b`, and the method returns 5.0. Arguments in the call must match the parameters in type and order.

Trap. Every path must return. If an if-else returns a value in only one branch, the method does not compile. When you write or check a method with a return type, look at the branch you almost forgot.

5.7 Static Variables and Methods

A **static variable** belongs to the class, not to any one object, and every object shares the same single copy. A **static method** also belongs to the class. It is called on the class name, like `Math.sqrt(16)`, and it cannot touch instance fields or use `this`, because there is no particular object for those to refer to.

```
public class Player {
    private String name;
    private static int playerCount = 0;

    public Player(String name) {
        this.name = name;
        playerCount++;
    }

    public static int getPlayerCount() {
        return playerCount;
    }
}
```

Each new `Player` increments the shared `playerCount`. After `Player p1 = new Player("Ava");` and `Player p2 = new Player("Ben");`, the call `Player.getPlayerCount()` returns 2. The call uses the class name. Inside `getPlayerCount`, writing `name` would be a compile error, because a static method has no object whose `name` it could read.

Trap. Static methods cannot use instance variables or `this`. If a method needs a field that differs per object, it must be an instance method. When an exam question asks which method fails to compile, look for a static method reaching for an instance field.

5.8 Scope and Access

Scope is the region of code where a variable can be used. There are three kinds to keep straight. Instance variables are declared in the class outside any method and are visible to every method of the class. Local variables are declared inside a method and exist only while that method runs. Parameters behave like local variables that are declared in the method header.

VARIABLE KIND	DECLARED	VISIBLE
Instance variable	In the class, outside any method	Every method of the class
Local variable	Inside a method	Only inside that method, after its declaration
Parameter	In the method header	Only inside that method

When a local variable or parameter has the same name as an instance variable, the local one **shadows** the field. Inside that method, the bare name refers to the local variable, and the field is hidden. This is legal, it is common in constructors and setters, and it is the source of some of the exam's favorite bugs.

5.9 The this Keyword

The keyword `this` refers to the current object, the one whose method or constructor is running. Its main job is to disambiguate. In `public Dog(String name, int age)`, the parameters shadow the fields, so `this.name = name` means assign the parameter `name` into the field `name` of this object.

```
public Dog(String name, int age) {  
    this.name = name; // field = parameter  
    this.age = age;  
}
```

Without `this`, `name = name` would assign the parameter to itself and leave the field at its default, which is null for a String. The code compiles, it runs, and the object is silently wrong. `this` is also how one constructor can call another, with `this(...)` as the first line, though the exam tests the disambiguation use far more often.

Trap. Shadowing bugs compile cleanly. `name = name` in a constructor looks right at a glance and leaves the field null. When a traced object has a null or 0 field you did not expect, check whether a parameter is shadowing the field and whether `this` is missing.

5.10 Ethical and Social Implications

Writing classes is a design activity, and design choices have consequences for real people. Deciding what data a class stores, who is allowed to see it, and what a program does with it raises questions about privacy, fairness, and access. A class that collects personal information creates a responsibility to protect it, and a method that makes decisions about people can carry bias from its programmer or its data. These questions appear lightly on the exam, usually as one multiple-choice question about the responsibilities that come with writing software.

Confusions That Cost Points

PAIR	HOW TO KEEP THEM STRAIGHT
Constructor vs a method with the class's name	A constructor has no return type at all. If you see void or any type before the class name, it is a method, not a constructor.
Default constructor present vs gone	Java provides a no-argument default constructor only when you write no constructors. Write any constructor and the default disappears.
Static method vs instance method	A static method runs on the class and cannot use instance fields or <code>this</code> . An instance method runs on an object and can use both.
Parameter shadowing a field vs assigning the field	Inside a constructor or setter, the bare name means the parameter. Write <code>this.name</code> to reach the field.
Private field inside the class vs outside it	Methods of the class can use its private fields directly. Code in other classes must go through public accessors and mutators.
Accessor vs mutator	An accessor returns a value and usually takes no parameters. A mutator returns void and takes the new value as a parameter.

Practice Questions

Original questions written for this guide in the style of the AP exam. Answers and explanations are on the next page, so complete the questions before checking them.

1. Consider the following class.

```
public class Ticket {  
    private int price;  
  
    public void Ticket(int p) {  
        price = p;  
    }  
  
    public int getPrice() {  
        return price;  
    }  
}
```

After the code segment `Ticket t = new Ticket();` executes, what does `t.getPrice()` return?

1. 50
2. 0
3. The code does not compile.
4. A runtime error occurs.

2. Consider the following class.

```
public class Circle {  
    private double radius;  
  
    public Circle(double r) {  
        radius = r;  
    }  
  
    public double getRadius() {  
        return radius;  
    }  
}
```

Which of the following code segments causes a compile-time error?

1. `Circle c = new Circle(2.5);`
2. `Circle c = new Circle(2.5); double r = c.getRadius();`
3. `Circle c = new Circle();`
4. `Circle c = new Circle(2);`

3. Consider the following class.

```
public class Counter {  
    private int count;  
    private static int total = 0;  
  
    public Counter() {  
        count = 0;  
        total++;  
    }  
  
    public static int getTotal() {  
        return total;  
    }  
  
    public static void resetAll() {  
        count = 0;  
        total = 0;  
    }  
}
```

Which method will NOT compile?

1. The constructor `Counter()`
2. `getTotal`
3. `resetAll`
4. All of the methods compile.

4. Consider the following class.

```
public class Student {  
    private String name;  
  
    public Student(String name) {  
        name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

After `Student s = new Student("Mia");` executes, what does `s.getName()` return?

1. "Mia"
2. null
3. ""
4. The code does not compile.

5. Consider the following class.

```
public class Bank {  
    private static int numAccounts = 0;  
    private int balance;  
  
    public Bank(int b) {  
        balance = b;  
        numAccounts++;  
    }  
  
    public static int getNumAccounts() {  
        return numAccounts;  
    }  
}
```

After the code segment below executes, what is the value of `Bank.getNumAccounts()` ?

```
Bank a = new Bank(100);  
Bank b = new Bank(200);
```

1. 0
2. 1
3. 2
4. 300

6. A class has the private field `private double gpa;`. Which of the following is a correct mutator method for this field, intended to be called from another class?

1.

```
public double setGpa(double g) {  
    gpa = g;  
}
```
2.

```
public void setGpa(double gpa) {  
    gpa = gpa;  
}
```
3.

```
public void setGpa(double g) {  
    this.gpa = g;  
}
```
4.

```
private void setGpa(double g) {  
    gpa = g;  
}
```

7. Consider the following class.

```
public class Wallet {  
    private int dollars;  
  
    public Wallet(int d) {  
        dollars = d;  
    }  
  
    public int getDollars() {  
        return dollars;  
    }  
  
    public void addDollars(int d) {  
        dollars += d;  
    }  
}
```

Consider the following code segment, which appears in a different class.

```
Wallet w = new Wallet(20); // line 1  
w.addDollars(10);          // line 2  
int x = w.dollars;          // line 3  
int y = w.getDollars();     // line 4
```

Which line causes a compile-time error?

1. line 1
2. line 2
3. line 3
4. line 4

8. Consider the following class.

```
public class Rectangle {  
    private int length;  
    private int width;  
  
    public Rectangle() {  
        length = 1;  
        width = 1;  
    }  
  
    public Rectangle(int side) {  
        length = side;  
        width = side;  
    }  
  
    public int area() {  
        return length * width;  
    }  
}
```

Consider the following code segment.

```
Rectangle r1 = new Rectangle();  
Rectangle r2 = new Rectangle(4);  
int total = r1.area() + r2.area();
```

What is the value of `total` after the code segment executes?

- 1. 5
- 2. 8
- 3. 17
- 4. 32

Answer Key

1. **B.** `public void Ticket(int p)` is a method, not a constructor, because a constructor has no return type. Since the class declares no real constructor, Java provides the default no-argument constructor, which sets price to 0. A is wrong because the void method never runs when an object is created. C is wrong because a method that shares the class name is legal Java as long as it has a return type. D is wrong because nothing here fails at runtime.
2. **C.** The class declares a constructor that takes a double, so the default no-argument constructor no longer exists and `new Circle()` does not compile. A and B correctly call the declared constructor. D compiles because the int literal 2 widens to double automatically.
3. **C.** `resetAll` is a static method that uses the instance field `count`, which is a compile-time error. Static methods have no object, so they cannot refer to instance fields. A compiles because a constructor may use both instance and static fields. B compiles because a static method may use static fields. D is wrong because `resetAll` fails.
4. **B.** The parameter `name` shadows the field `name`, so `name = name` assigns the parameter to itself and the field keeps its default value, which is null for a String. A ignores the shadowing and assumes the field was set. C confuses null with the empty string. D is wrong because self-assignment of a parameter compiles without complaint.
5. **C.** `numAccounts` is static, so there is one shared copy. Each constructor call increments it, and two objects were created, so the value is 2. A treats the static field as if the constructor never ran. B treats the static field as if each object got its own copy. D confuses the balances, 100 and 200, with the account count.
6. **C.** A correct mutator is public, returns void, and takes the new value as a parameter. A declares a double return type but returns nothing, so it does not compile. B compiles but the parameter shadows the field, so `gpa = gpa` changes nothing. D compiles but is private, so it cannot be called from another class.
7. **C.** The field `dollars` is private, so line 3, which reads it from a different class, does not compile. Line 1 correctly calls the public constructor. Line 2 correctly calls the public mutator. Line 4 correctly calls the public accessor, which is exactly how outside code is meant to read the field.
8. **C.** The constructors are overloaded. `new Rectangle()` sets length and width to 1, so `r1.area()` is 1. `new Rectangle(4)` sets both to 4, so `r2.area()` is 16. The total is 17. A adds the side lengths instead of the areas. B and D come from mixing up which constructor ran or doubling the wrong value.

When you check your answers, note which distinction each miss came from. Make a flashcard for that distinction and drill it spaced out over the next few days instead of rereading the whole section. If you missed one of these questions, the same distinction is worth practicing again in Rycal, where the Writing Classes deck has flashcards for it and more practice questions use the same kinds of traps.

One-Page Recall Check

Say each answer out loud before you look back, and mark the ones you cannot finish. Anything you cannot say out loud yet belongs in your flashcard deck. In Rycal, add those items to the Writing Classes deck and let spaced review bring them back over the next few days.

- State the three parts of a class: fields, constructors, and methods.
- Explain what private and public mean and why fields are private while the methods other classes use are public.
- State the three rules that define a constructor.
- Explain when Java provides a default constructor and when it disappears.
- Explain what overloaded constructors are and how Java picks which one runs.
- Describe what a documentation comment looks like and what @param and @return mean.
- Write an accessor method for a private int field and explain the naming pattern.
- Write a mutator method for a private double field, including validation, and explain why it returns void.
- State the three parts of a method signature and the rule about returning on every path.
- Explain the difference between a static variable and an instance variable, and why a static method cannot use this.
- Describe the three kinds of scope and give an example of each.
- Explain shadowing and show how this fixes it in a constructor.
- Give one example of an ethical question a programmer faces when writing a class that stores personal data.

Where to go next. Turn every missed item above into flashcards and drill them spaced out over several days rather than in one sitting. In Rycal, open the Writing Classes deck under AP Computer Science A. The deck covers the terms in this guide, and its practice questions target the same traps named here. If you have a test date, add it in the Test Planner. You can also start your next review with a Brain Dump, then check what you missed against this guide.

Key terms for this unit

Class, Object, Instance variable (field), Constructor, Default constructor, Overloaded constructor, Encapsulation, Private, Public, Accessor method (getter), Mutator method (setter), Method signature, Return type, Parameter, Static variable, Static method, Scope, Local variable, Shadowing, this keyword, Documentation comment, @param, @return.

About this guide. Written for Rycal and aligned to the College Board AP Computer Science A course framework, Unit 5. All questions and explanations are original Rycal writing. Rycal is independent and is not affiliated with or endorsed by the College Board.