

Unit 2: Using Objects

Unit 2 is about using objects that other people wrote. It covers how objects are created and stored, how to call their methods with and without parameters, and how to work with the `String` class, the wrapper classes, and the `Math` class. This is the basic vocabulary of Java, and every unit after this one assumes you can read it fluently.

AP Computer Science A

Using Objects

About 15 minutes to read

How to use this guide

Read it in order the first time. Objects and references come first because every method call in the unit depends on understanding what a reference is. `String` methods get the most space because they are the most tested. The trap boxes mark the distinctions that show up as wrong answers on the exam.

After the first read, use the trap boxes and the Confusions table to review. Then do the practice questions without looking back, and finish with the recall check on the last page. Anything you cannot say out loud belongs in your flashcard deck.

What this unit is worth. Unit 2 is about 5 to 7.5 percent of the AP CSA exam. More importantly, object references, method calls, and `String` methods are the vocabulary of every free-response question. If you cannot read a method call fluently, Units 5 through 10 will be hard.

2.1 Objects: Instances of Classes

A **class** is a blueprint. It describes the data each object will hold, called its **attributes**, and the actions each object can perform, called its **behavior**. An **object** is one instance built from that blueprint. The `String` class is the blueprint. The text "hello" sitting in your program is one `String` object.

In this unit you only use classes that someone else wrote. Java ships with classes like `String`, `Scanner`, and `Math`, and the AP exam expects you to use them without seeing their code. Writing your own classes comes in Unit 5. For now the job is to read code that uses objects and predict what it does.

Trap. A class and an object are not the same thing. The class defines what is possible. The object is the actual thing in memory holding actual data. Two objects built from the same class can hold different values, the way two `String` objects can hold different text.

2.2 Creating and Storing Objects

The **new** keyword builds an object. The expression `new String("Maya")` asks Java to create a `String` object holding the text `Maya`. The part after `new` that looks like a method call is the **constructor**. A constructor has the same name as the class, and its job is to set the new object up.

The variable on the left does not hold the object. It holds a **reference**, which is the location of the object in memory. In `String name = new String("Maya")`, the variable `name` stores a reference to the `String` object, not the text itself. Picture the variable holding an arrow that points at the object.

```
String name = new String("Maya");
```

Trap. Assignment copies the reference, not the object. After `String b = a`, both variables point at the same object. Reassigning `b` afterward, with `b = new String("Lee")`, moves only `b`'s arrow. The object `a` points at is unchanged.

2.3 Calling a Void Method

A method call on an object uses the dot. The pattern is `objectName.methodName()`. A **void method** performs an action and returns nothing. `System.out.println("Hello")` calls the `println` method on the `System.out` object, and it prints the text and moves to a new line. `System.out` is itself an object, and `println` is one of its methods.

The method you call must exist on that object's class, or the code will not compile. Java checks the method name and the parameter list against the class definition before the program ever runs. A misspelled method name is a compile error, not a runtime surprise.

```
System.out.println("Hello");  
System.out.println("Line two");
```

2.4 Calling a Void Method with Parameters

Many methods need input. The values you pass in a call are **arguments**. The names listed in the method's definition are **parameters**. In `System.out.println(score + 1)`, the argument is the value of `score + 1`, and `println`'s parameter receives a copy of that value.

Arguments must match the parameters in number, order, and type. If the method expects an `int` and you hand it a `String`, the code will not compile. When a method takes several parameters, the first argument goes to the first parameter, the second to the second, and so on.

Trap. The argument is evaluated before the call. In `System.out.println(2 + 3)`, the expression `2 + 3` becomes 5 first, and `println` receives 5. The method never sees the expression, only the result.

2.5 Calling a Non-void Method

A **non-void method** returns a value. The **return type** in the method's signature tells you what kind of value comes back. String's length method returns an int, so `int n = word.length()` stores that number in `n`.

You can use the returned value anywhere a value of that type fits. Print it, store it, or compute with it. `System.out.println(word.length() + 3)` prints the length plus three. Ignoring a return value is legal Java, but the value is lost the moment the call ends.

```
String word = "object";
int n = word.length();           // n is 6
System.out.println(word.length() + 3); // prints 9
```

2.6 String Objects: Concatenation, Literals, and More

A **String literal** is text in double quotes. Writing `"hello"` creates a String object directly, with no new needed. The `+` operator joins Strings together, which is called **concatenation**. When one side of `+` is a String, Java converts the other side to text, so `"Age: " + 16` produces `"Age: 16"`.

Evaluation runs left to right, and that decides when numbers add and when they become text. In `2 + 3 + "!"` the `2 + 3` happens first because no String has appeared yet, giving `"5!"`. In `"!" + 2 + 3` the String appears first, so everything after it becomes text, giving `"!23"`. Parentheses can force the order you want: `"!" + (2 + 3)` gives `"!5"`.

Three **escape sequences** appear on the exam. `\n` starts a new line inside a String, `\"` puts a double quote inside a String, and `\\` puts a single backslash. Each escape counts as one character.

```
System.out.println("Total: " + 4 + 5); // prints Total: 45
System.out.println(4 + 5 + " total"); // prints 9 total
```

Trap. `"Total: " + 4 + 5` is `"Total: 45"`, not `"Total: 9"`. Once a String enters the expression, every `+` after it concatenates. The exam loves this exact question.

2.7 String Methods

String methods are how you inspect and slice text. Each one is called on a String object with the dot, and each returns a value you can use. The exam tests **length**, **substring**, **indexOf**, **equals**, and **compareTo**.

length() returns the number of characters as an int. **indexOf(str)** returns the index of the first occurrence of str, or -1 if str is not in the String. Indices start at 0. There is also a two-argument form, **indexOf(str, fromIndex)**, which starts the search at fromIndex instead of 0.

```
String s = "banana";
s.length();      // 6
s.indexOf("na"); // 2
s.indexOf("z");  // -1
```

Trap. **indexOf** returns -1 when the text is not found. It does not throw an error, and it does not return 0. Code that uses the result as an index must check for -1 first.

substring takes indices. **substring(start)** returns the text from start to the end of the String. **substring(start, end)** returns the text from start up to but not including end. The end index is exclusive, and that is the single most tested detail in this unit.

```
String word = "computer"; // c0 o1 m2 p3 u4 t5 e6 r7
word.substring(2);        // "mputer"
word.substring(2, 5);     // "mpu"
```

Trap. **substring(2, 5)** gives the characters at indices 2, 3, and 4. The character at index 5 is not included. If you want the first three characters of a String, use **substring(0, 3)**, not **substring(0, 2)**.

To compare the contents of two Strings, use **equals**. The call `s1.equals(s2)` returns true when both hold the same text. The `==` operator compares references, so `s1 == s2` is true only when both variables point at the same object. Two different objects with identical text will fail `==`.

This is the number one trap in the unit. String literals can mislead you, because Java sometimes reuses one object for identical literals, which makes `==` look correct in a quick test. With objects built by `new`, `==` fails even when the text matches. The rule is simple. For String contents, always use `equals`. Never use `==`.

```
String a = new String("code");
String b = new String("code");
boolean sameObject = (a == b); // false, different objects
boolean sameText = a.equals(b); // true, same text
```

Trap. `==` on Strings checks identity, not content. Two String objects holding "code" are `==` only if they are literally the same object. The exam will hand you two equal-looking Strings built with `new` and ask about `==`. The answer is false.

compareTo compares Strings in dictionary order and returns an int. The call `s1.compareTo(s2)` returns 0 when the contents are equal, a negative number when `s1` comes first, and a positive number when `s2` comes first. `"apple".compareTo("banana")` is negative because `a` comes before `b`. Case matters, because uppercase letters sort before lowercase ones.

Strings cannot be changed. Every String method that looks like it edits the text actually returns a brand new String and leaves the original alone. This is called **immutability**. Calling `word.substring(0, 3)` without storing the result does nothing to `word`. To keep the change, reassign the result.

```
String word = "AP CSA";
word.substring(3); // result thrown away, word unchanged
word = word.substring(3); // now word holds "CSA"
```

2.8 Wrapper Classes: Integer and Double

`int` and `double` are primitives. They hold values directly and have no methods. Sometimes Java needs an object instead, so each primitive has a **wrapper class**: Integer wraps `int`, Double wraps `double`. Writing `Integer n = 7` converts the 7 automatically, which is called **autoboxing**. Using `n` in arithmetic converts it back, which is called **unboxing**.

The exam tests the named constants. `Integer.MAX_VALUE` is the largest possible `int`, 2147483647. `Integer.MIN_VALUE` is the smallest. Double has `MAX_VALUE` and `MIN_VALUE` too, though `Double.MIN_VALUE` is the smallest positive double, not the most negative one. One more difference from primitives is that an Integer variable can be null, which means it points at no object. An `int` variable cannot be null.

Trap. Using a null Integer in arithmetic throws a `NullPointerException`. If Integer `n` was never assigned and your code evaluates `n + 1`, Java tries to unbox nothing and the program crashes at runtime.

2.9 Using the Math Class

The Math class holds methods you call on the class itself, with no object: `Math.abs(-9)`. These are **static methods**. You will write your own static methods in Unit 5, but here you only call them.

METHOD	WHAT IT RETURNS
<code>Math.abs(x)</code>	The absolute value, in the same type as x. <code>Math.abs(-9)</code> is the int 9. <code>Math.abs(-9.5)</code> is the double 9.5.
<code>Math.pow(a, b)</code>	a raised to the power b, always a double. <code>Math.pow(2, 3)</code> is 8.0, not 8.
<code>Math.sqrt(x)</code>	The square root of x, a double. <code>Math.sqrt(16)</code> is 4.0.
<code>Math.random()</code>	A double from 0 up to but not including 1. It can return 0.0, but it never returns exactly 1.0.

`Math.random` needs scaling to be useful. Multiplying stretches the range, and casting to int drops the decimal part. `(int)(Math.random() * 10)` gives an integer from 0 through 9. Adding 1 shifts the range: `(int)(Math.random() * 6) + 1` gives 1 through 6, like a die roll.

Trap. Two details ruin random ranges. First, `Math.random()` never returns exactly 1.0, so `(int)(Math.random() * 10)` can never be 10. Second, the cast must wrap the multiplication. `(int)Math.random() * 10` casts first, which always gives 0, so the whole expression is always 0.

Confusions That Cost Points

PAIR	HOW TO KEEP THEM STRAIGHT
<code>==</code> vs <code>.equals()</code> for Strings	<code>==</code> asks whether two references point at the same object. <code>.equals()</code> asks whether two Strings hold the same text. For content, use <code>.equals()</code> .
substring start vs end	Both are indices, but the end is exclusive. <code>substring(2, 5)</code> covers indices 2, 3, and 4.
<code>indexOf</code> found vs not found	Found returns the first matching index. Not found returns -1, not an error.
Using vs keeping a String result	String methods return new Strings. The original changes only when you reassign the result.
<code>(int)(Math.random() * n)</code>	Gives 0 through n - 1. The cast goes around the multiplication, and 1.0 never appears.
<code>b = a</code> with objects	Copies the reference. Both variables point at the same object until one is reassigned.

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 code.

```
String s1 = new String("code");  
String s2 = new String("code");  
System.out.println(s1 == s2);  
System.out.println(s1.equals(s2));
```

What is printed?

1. true
false
2. false
true
3. true
true
4. false
false

2. Consider the following code.

```
String word = "computer";  
System.out.println(word.substring(2, 5));
```

What is printed?

1. mpu
2. mp
3. ompu
4. mput

3. Consider the following code.

```
String s = "banana";  
System.out.println(s.indexOf("na") + s.indexOf("z"));
```

What is printed?

1. 1
2. 2
3. 3
4. -1

4. What is printed by the following statement?

```
System.out.println("Total: " + 4 + 5);
```

1. Total: 45
2. Total: 9
3. 9
4. Total: 4 + 5

5. Which of the following expressions evaluates to a random integer from 1 to 6, inclusive, each time it runs?

1. `(int)(Math.random() * 6)`
2. `(int)(Math.random() * 6) + 1`
3. `(int)(Math.random() * 7)`
4. `(int) Math.random() * 6 + 1`

6. Consider the following code.

```
String a = new String("data");  
String b = a;  
b = new String("info");  
System.out.println(a);
```

What is printed?

1. data
2. info
3. null
4. The code does not compile.

7. Consider the following code.

```
String word = "AP CSA";  
word.substring(3);  
System.out.println(word.length());
```

What is printed?

1. 6
2. 3
3. 2
4. 0

8. What is printed by the following statement?

```
System.out.println("She said \"hi\"");
```

1. She said "hi"
2. She said \"hi\"
3. She said hi
4. The code does not compile.

Answer Key

- 1. B.** `s1 == s2` is false because new built two separate objects, and `==` compares references. `s1.equals(s2)` is true because both hold the text "code". A claims `==` is true, which would require both variables to point at the same object. C claims both are true, which ignores that the objects are different. D claims `equals` is false, which ignores that `equals` compares contents.
- 2. A.** The indices in "computer" are c0 o1 m2 p3 u4 t5 e6 r7. `substring(2, 5)` takes indices 2, 3, and 4, which is "mpu". B uses an exclusive end of 4 instead of 5. C starts at index 1. D treats the end index as inclusive.
- 3. A.** `indexOf("na")` finds "na" starting at index 2, and `indexOf("z")` returns -1 because "z" is absent. $2 + (-1) = 1$. B forgets to add the -1. C adds 1 instead of -1. D reports only the not-found result and drops the 2.
- 4. A.** Evaluation runs left to right. "Total: " is a String, so `+ 4` concatenates to "Total: 4", and then `+ 5` concatenates to "Total: 45". B adds `4 + 5` first, which only happens when the String comes later in the expression. C drops the label entirely. D treats the expression as literal text instead of evaluating it.
- 5. B.** `Math.random()` returns a value from 0 up to but not including 1. Multiplying by 6 gives 0 up to but not including 6, casting to `int` gives 0 through 5, and adding 1 shifts the range to 1 through 6. A gives 0 through 5, which includes 0 and misses 6. C gives 0 through 6, which includes 0. D casts before multiplying, so `(int)Math.random()` is always 0 and the result is always 1.
- 6. A.** The line `b = a` copies the reference, so both variables point at the object holding "data". Then `b` is reassigned to a new object holding "info", which moves only `b`'s reference. `a` still points at "data". B reports `b`'s object instead of `a`'s. C confuses reassignment with clearing the reference. D is wrong because every line here is legal Java.
- 7. A.** `substring` returns a new String, and the result here is thrown away, so `word` still holds "AP CSA", whose length is 6. B is the length the word would have if `substring` edited it in place, which Strings never do. C is the length of "CSA", the discarded result. D would require `word` to be empty, which nothing here does.
- 8. A.** The sequence `\'` is the escape for a quotation mark inside a String, so the printed line is She said "hi". B prints the backslashes themselves, which is what the line would show if each backslash were doubled. C drops the quotes, which is what would print if the escapes were absent. D is wrong because the escapes make the line legal.

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 AP Computer Science A Unit 2 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 AP Computer Science A Unit 2 deck and let spaced review bring them back over the next few days.

- Explain the difference between a class and an object, using String as an example.
- State what the new keyword does and what a constructor is for.
- Explain what a reference variable stores, and what it means for a reference to be null.
- Predict what changes when one of two references to the same object is reassigned.
- Write the syntax for calling a void method on an object.
- Explain the difference between a parameter and an argument.
- Show two ways to use the value returned by a non-void method.
- Predict the output of `System.out.println("Score: " + 4 + 5)` and explain each step.
- Write the escape sequences for a newline, a quotation mark, and a backslash.
- State what `length()`, `substring(start, end)`, `indexOf(str)`, `equals(str)`, and `compareTo(other)` each return.
- Explain why `==` gives the wrong answer when comparing String contents.
- Explain String immutability and show the line that actually shortens a String.
- State the range of `Math.random()` and write the expression for a random integer from 1 to 6.
- State the return types of `Math.abs(-4)`, `Math.pow(2, 3)`, and `Math.sqrt(16)`.

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 AP Computer Science A Unit 2 deck. 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 of a class), Attribute, Behavior, Constructor, new keyword, Reference variable, Null reference, Void method, Parameter, Argument, Return value, String literal, Concatenation, Escape sequence, `length()`, `substring()`, `indexOf()`, `equals()`, `compareTo()`, String immutability, Wrapper class, Autoboxing, Unboxing, `Integer.MAX_VALUE`, `Integer.MIN_VALUE`, `Math.abs()`, `Math.pow()`, `Math.sqrt()`, `Math.random()`.

About this guide. Written for Rycal and aligned to the College Board AP Computer Science A course framework, Unit 2. All questions and explanations are original Rycal writing. Rycal is independent and is not affiliated with or endorsed by the College Board.