Poll Everywhere

PollEv.com/javabear text **javabear** to **22333**

Which of the following are appropriate static type declarations for c in the following code snippet?

```
int a = 7;
double b = 3.4;
_____ c = a / 2 < b ? a : 2 * b;
```

boolean

int double

widened to double

double

- `int`, only **(A)**
- `double`, only **(B)**
- `int` or `double` **(C)**
- neither will compile **(D)**

Lecture 2: Reference Types

CS 2110

January 22, 2026

Today's Learning Outcomes

4. Explain how Java stores and references objects in memory.

5. Use the Java documentation to locate a method for a desired behavior and successfully incorporate that method into a program.

6. Draw object diagrams to visualize reference types.

7. Write Java programs involving arrays that utilize syntax for indexing and array literals.

8. Write and execute code in Java that uses program arguments.

Beyond Primitive Types

Recall: A type models a set of states and behaviors

Java's 8 primitive types only store simple data (numbers, characters, true, false)

Need a way to model more complicated data (Strings, arrays, Bank Accounts, data structures, etc.)

A class is a code blueprint for a non-primitive type

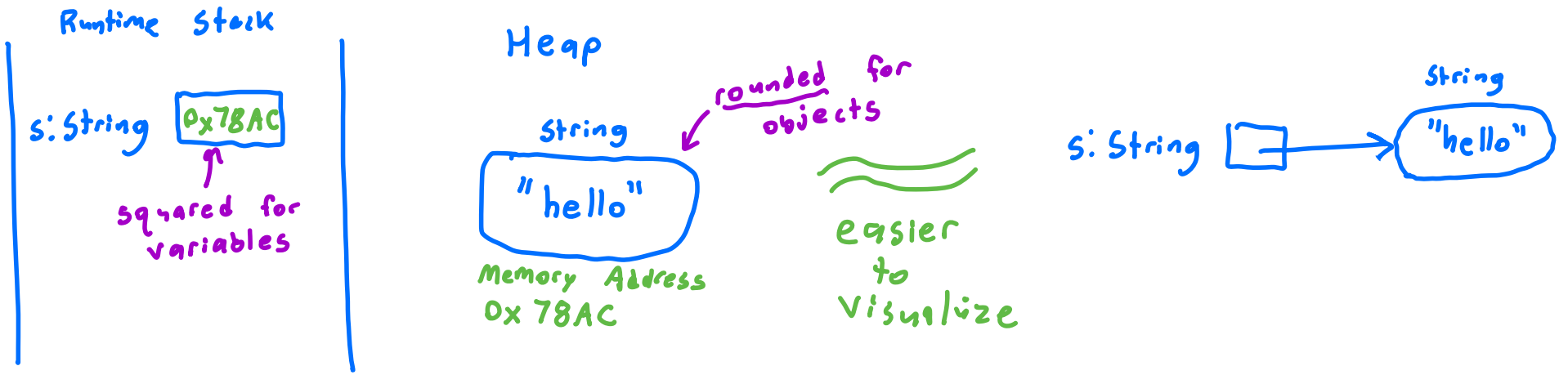
An instance of a non-primitive type is called an object.

References and the Memory Heap

Non-primitive types are also called reference types

In Java, all objects are stored in memory allocated on the heap (separate from the runtime stack)

A non-primitive variable stores a reference of where on the heap the assigned object is located

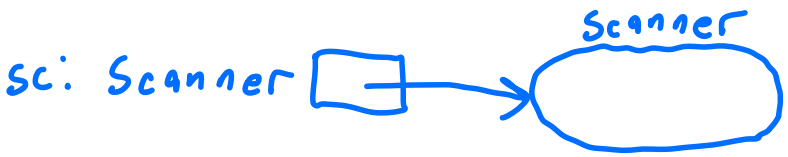


Construction

The process of allocating memory according to its class's blueprint to build a new object

```
Scanner sc = new Scanner();
```

- `Scanner()`: constructor method name matches class
- `new`: Java keyword
- `()`: constructor method arguments



1. finds heap memory for object
2. calls constructor to "set up" the object
3. "returns" (evaluates to) a reference to the new object

```
String s = new String("hello");
```

or

```
String s = "hello";
```

← String literal syntax (special for Strings)

Instance Methods

Behaviors of reference types specified by instance methods

– defined in class and called on object

```
String s = "hello";
int i = s.length(); // 5
```

method target — we're calling the length method on s

member access operator

instance method name

arguments

```
class String {
  ⋮
  int length() { ... }
  ⋮
}
```

```
int j = s.indexOf("e"); // 1
String t = s.substring(1,3); // "el"
```

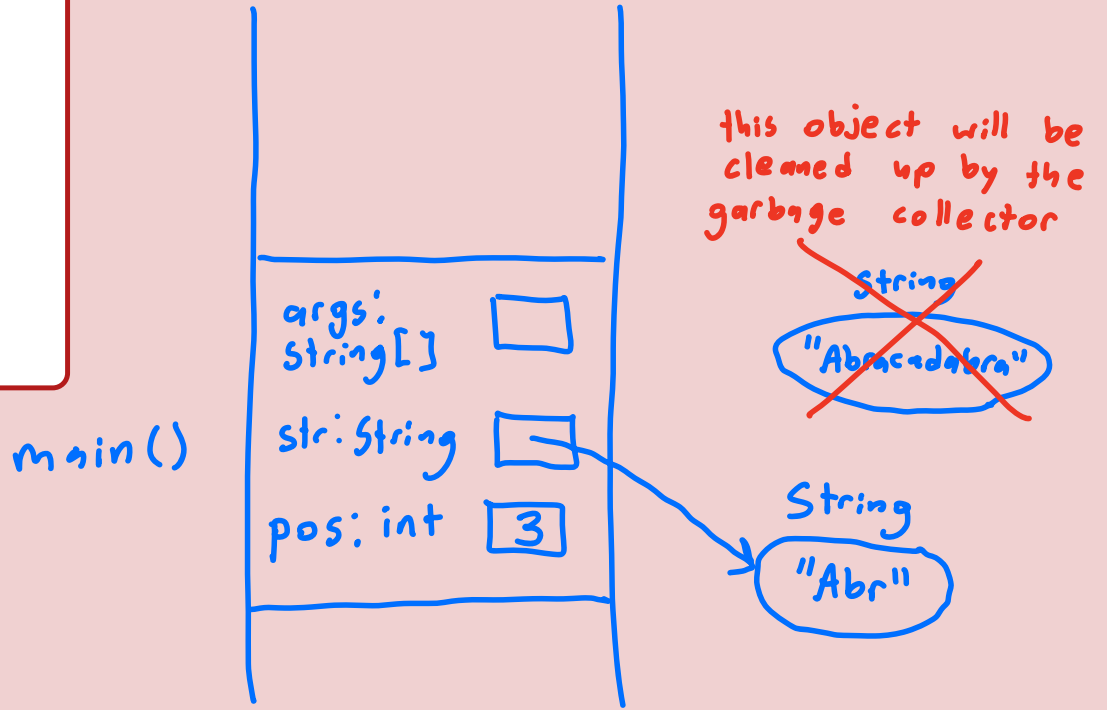
See: http://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

What will be printed by the following code snippet?

```
static void main(String[] args) {
    String str = "Abracadabra";
    int pos = str.indexOf('a');
    str = str.substring(0,pos);
    System.out.print(str);
}
```



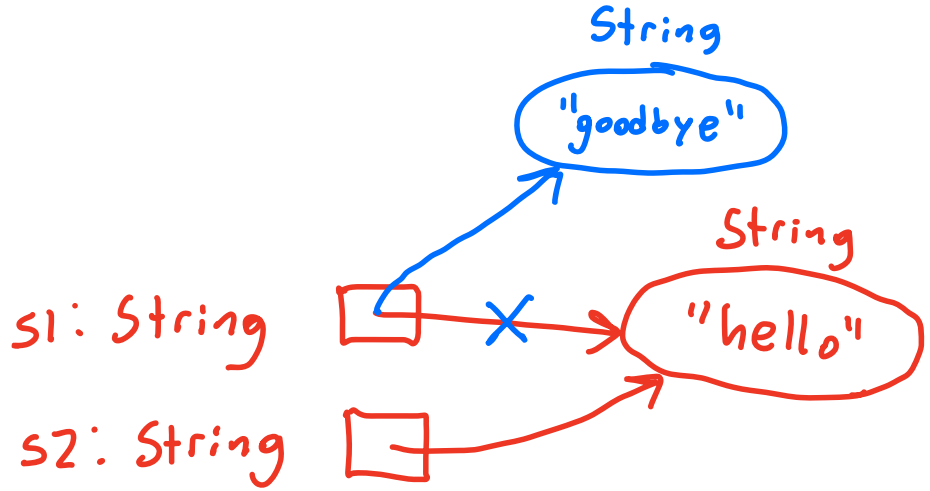
Reference Semantics

The value of a ref. type variable is a reference (an arrow)

Reassigning that variable changes the arrow (to point to a different object)

```
String s1 = "hello";
String s2 = s1; // alias reference

s1 = "goodbye";
System.out.print(s2);
```



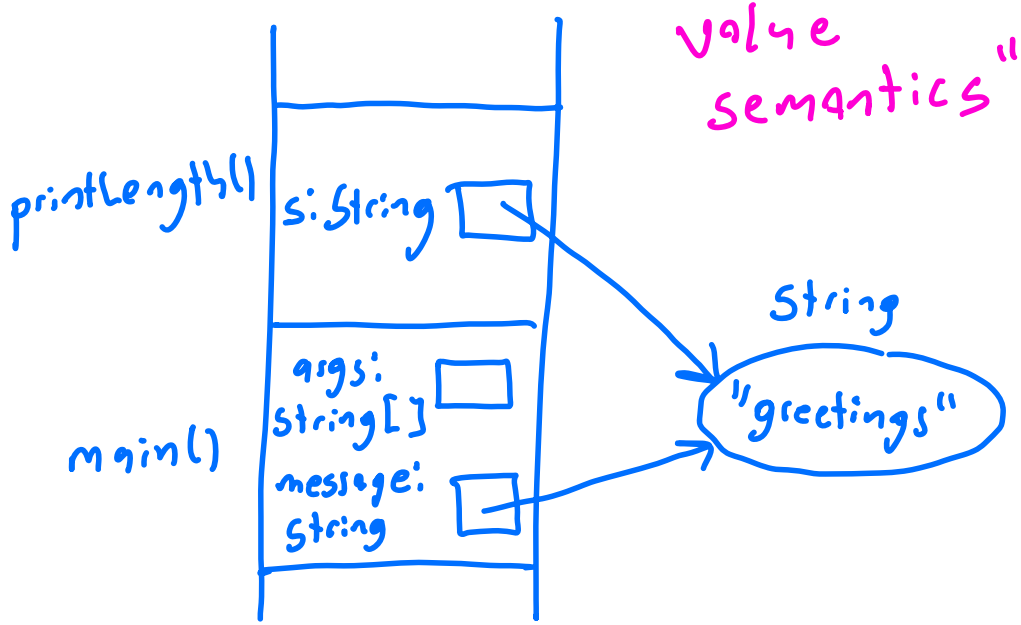
Reference Type Parameters

In Java, the values of all arguments of a method call are copied into the corresponding parameter entries of the new stack frame.

"pass by value semantics"

```
static void printLength(String s) {
  System.out.print(s.length());
}

public static void main(String[] args) {
  String message = "greetings";
  printLength(message);
}
```



Poll Everywhere

PollEv.com/javabear text **javabear** to **22333**

What gets printed by the following code snippet?

```
static void pluralize(String str) {
    str = str + "s";
}

static void main(String[] args) {
    String animal = "fox";
    pluralize(animal);
    System.out.print(animal);
}
```

fox	**(A)**
foxs	**(B)**
foxes	**(C)**
animal	**(D)**

Poll Everywhere

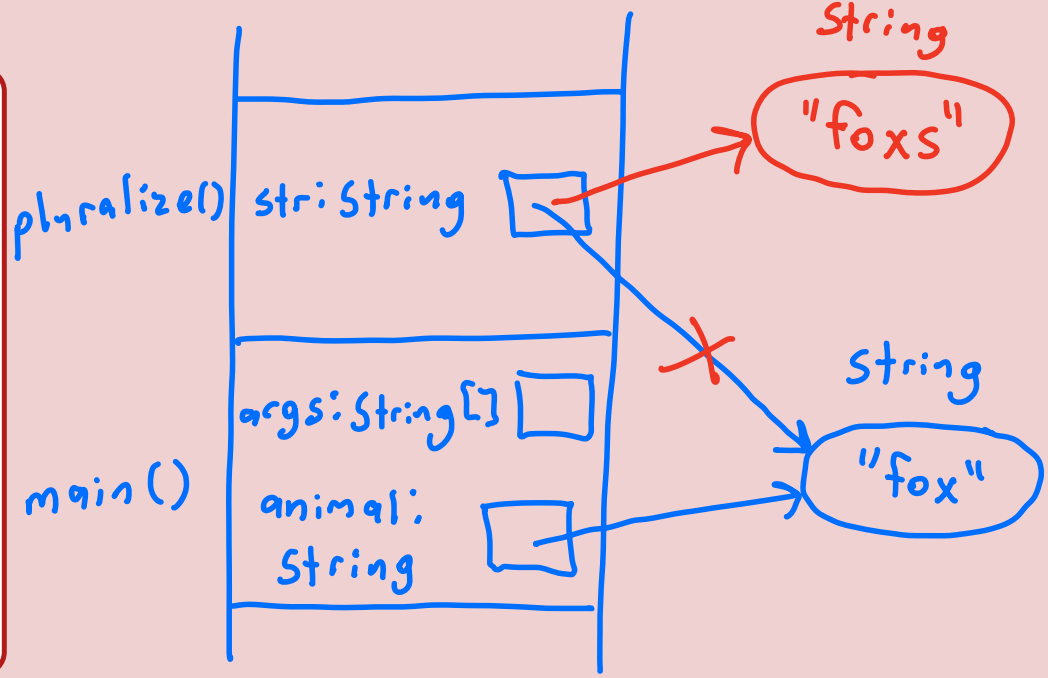
PollEv.com/javabear text **javabear** to 22333

What gets printed by the following code snippet?

```
static void pluralize(String str) {
    str = str + "s";
}

static void main(String[] args) {
    String animal = "fox";
    pluralize(animal);
    System.out.print(animal); // "fox"
}
```

concatenation makes new String



null

null is a special value for every reference type (and a Java keyword) which indicates the absence of a reference.

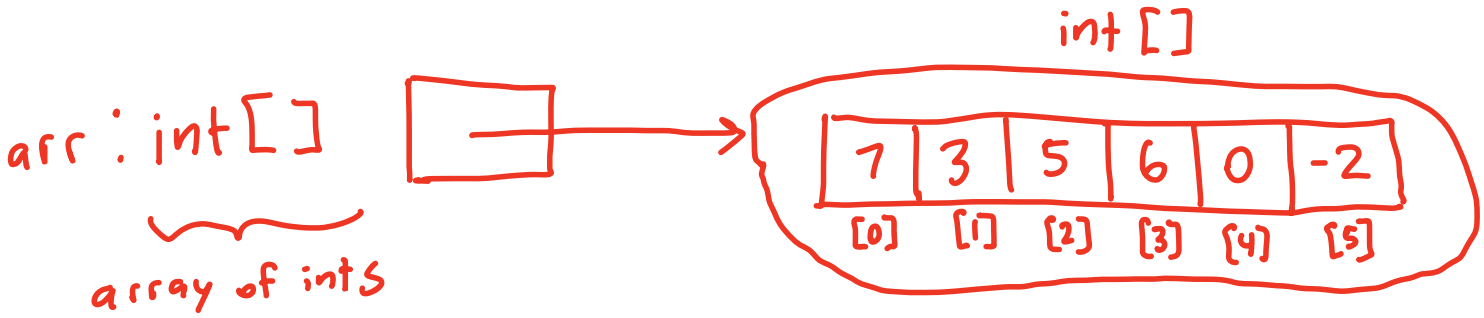
```
String s = null;
```

Trying to invoke an instance method on a target that is null will cause a NullPointerException

* be careful when variables can be null

Arrays

An array is a simple data structure that consists of a fixed number of sequentially numbered cells that store elements of a particular type.



arr.length = 6; ← no parentheses – length is a field of the array, not a method (more on this soon)

arrays are a reference type

array cells can store any primitive or reference type

Array Initialization and Array Literals

Construct an array by specifying its length

```
int[] arr = new int[6];
```

initially filled with default value
- 0 for primitive numerical types
- false for boolean
- null for reference types

or

use an array literal

```
int[] arr = new int[] {7, 3, 5, 6, 0, -2};
```

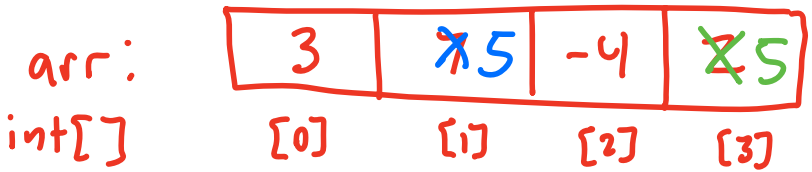
or

```
int[] arr = {7, 3, 5, 6, 0, -2};
```

Array Indexing

To access the elements in an array, use square bracket syntax with index inside

* indices start from 0




```
int i = arr[2]; // -4
arr[1] = 5;
arr[arr[0]] = arr[1];
```

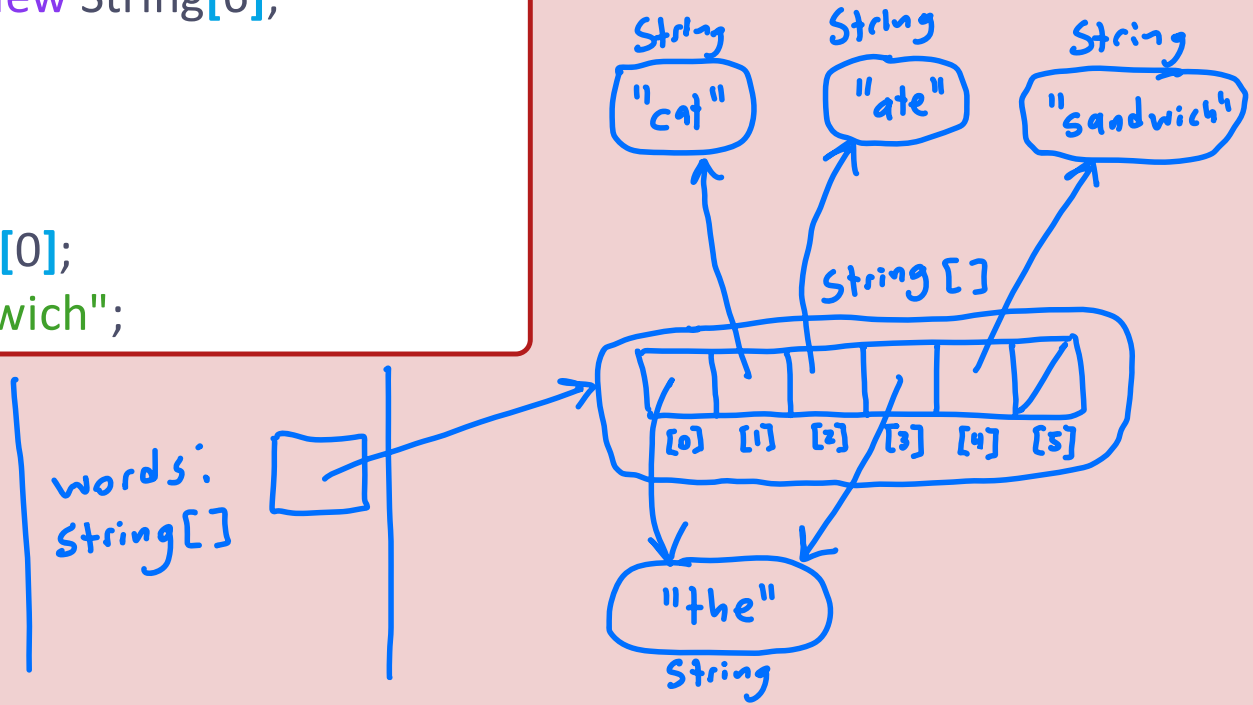
3

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

Draw the memory diagram for the following code snippet. How many objects are allocated on the heap?

```
String[] words = new String[6];
words[0] = "the";
words[1] = "cat";
words[2] = "ate";
words[3] = words[0];
words[4] = "sandwich";
```



4 (A)

5 (B)

6 (C)

7 (D)

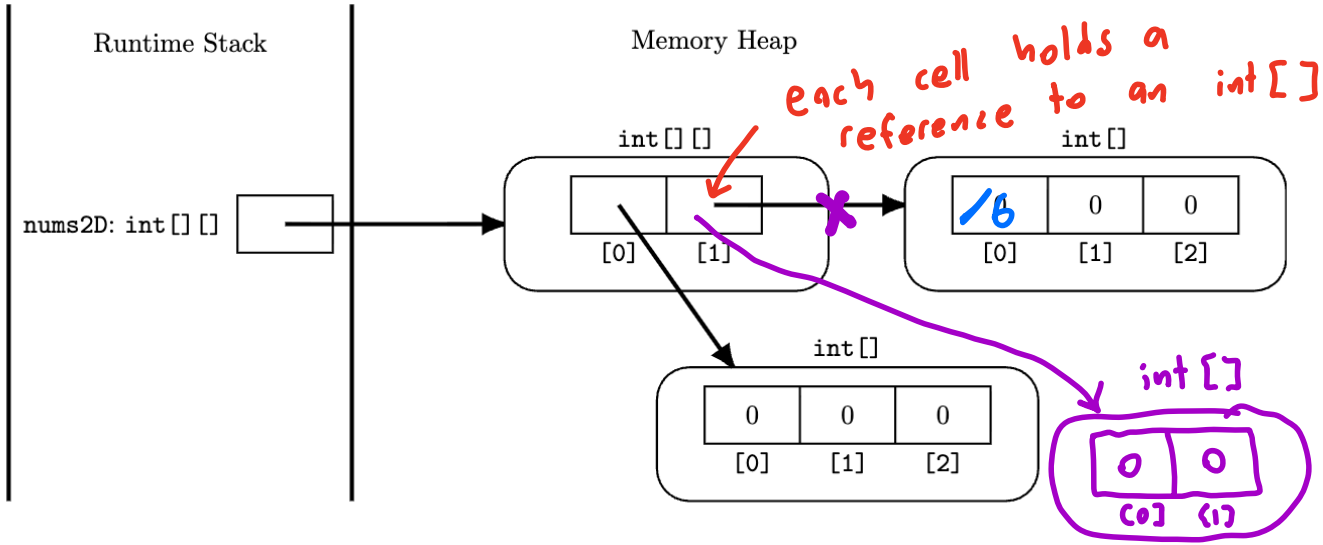
Multi-Dimensional Arrays

```
int[][] nums2D = new int[2][3];
```

// array of arrays

outer size

inner size



indices evaluated left → right

```
nums2D[1][0] = 6;
```

```
nums2D[1] = new int[2];
```

Program Arguments

program entrypoint:

```
public static void main(String args[]) {

}
```

program arguments

passed into Java when we tell it which program to run

often: space-separated String arguments typed on command line

or run configurations in IntelliJ (see demo)

Coding Demo: Echo Program