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?

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

conditional operator

boolean int double

widen

double

double

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

Lecture 2: Reference Types

CS 2110

August 27, 2026

Today's Learning Outcomes

4. Describe how Java stores and references objects in memory and how references are aliased across method calls.

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

6. Draw a memory diagram that visualizes instantaneous state of one or more objects.

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** (possible values) and **behaviors** (ways to use its data).

Java's 8 primitive types store simple data (numbers, characters, true/false) and are manipulated using built-in operators.

We need a way to model more complicated data (Strings, arrays, input/output, bank accounts, data structures, ...)

A class provides a code blueprint for a non-primitive type.

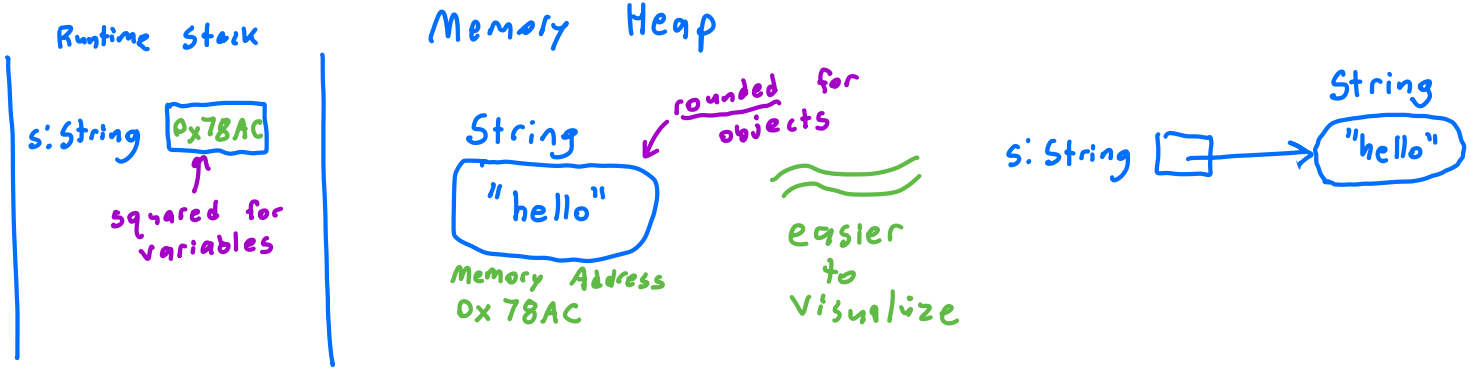
An object is an instance of a non-primitive type.

References and the Memory Heap

Non-primitive types are also called **reference types**.

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

A non-primitive variable stores the heap memory address where an object of that type is located.



Construction

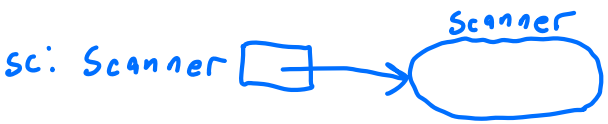
= The process of allocating memory to build a new object according to its class's "blueprint".

constructor method name matches class

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

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");
```

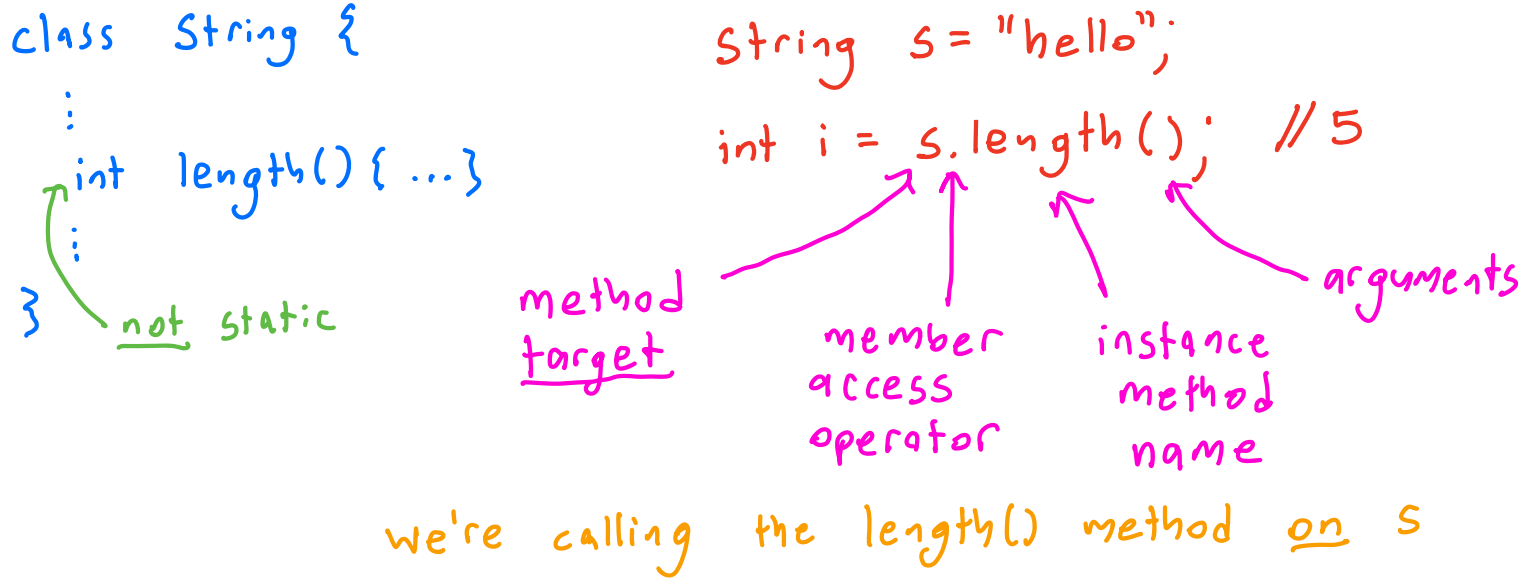
or

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

← String literal syntax (special for Strings)

Instance Methods

Behaviors of reference types are modeled by **instance methods**, which are defined in that type's class and called (or *invoked*) on an object.



Some String Instance Methods

Documented in the Java 21 API:
http://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

RETURN TYPE	METHOD NAME/PARAMETERS	DESCRIPTION
`char`	`charAt(int index)`	Returns the `char` value at the specified index.
`int`	`indexOf(char c)`	Returns the index within this string of the first occurrence of the specified character.
`int`	`length()`	Returns the length of this string.
`String`	`substring(int beginIndex, int endIndex)`	Returns a string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus, the length of the substring is `endIndex - beginIndex`.

String s = "javabear"

s.charAt(3)
'a'

s.indexOf('a')
1

s.length()
8

s.substring(1,5)
"avab"

Poll Everywhere

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

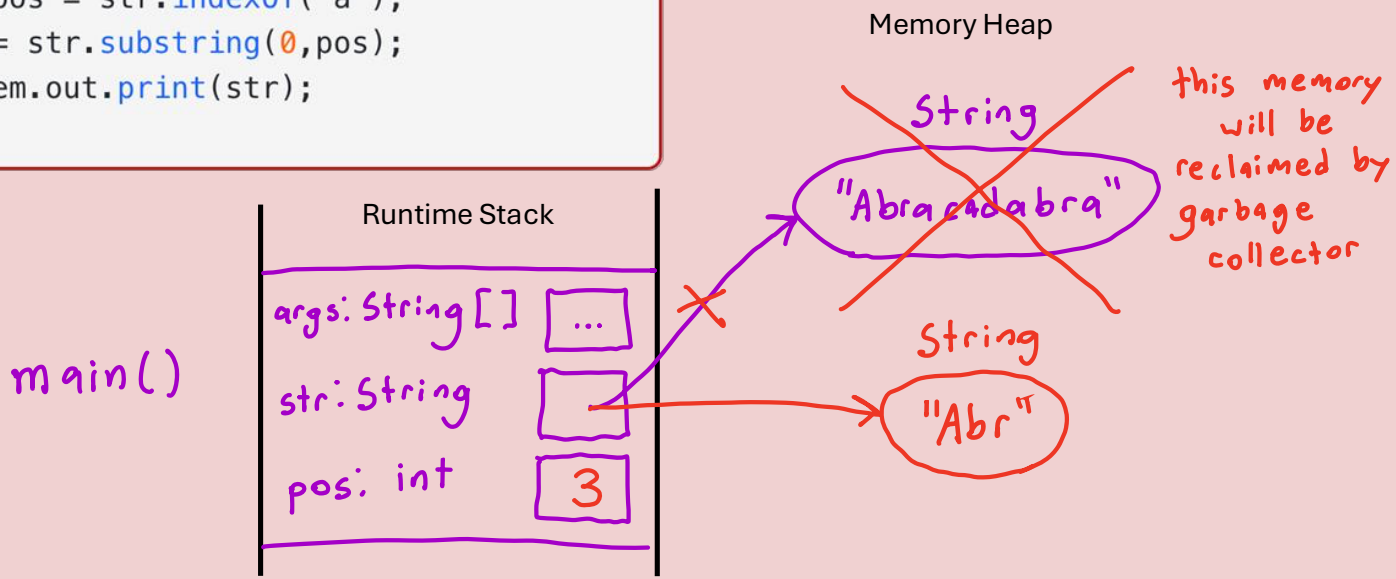
What will be printed by the following code snippet?

```
1  public static void main(String[] args) {
2    String str = "Abracadabra";
3    int pos = str.indexOf('a');
4    str = str.substring(0,pos);
5    System.out.print(str);
6  }
```

Poll Everywhere

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

```
1  public static void main(String[] args) {
2    String str = "Abracadabra";
3    int pos = str.indexOf('a');
4    str = str.substring(0,pos);
5    System.out.print(str);
6  }
```



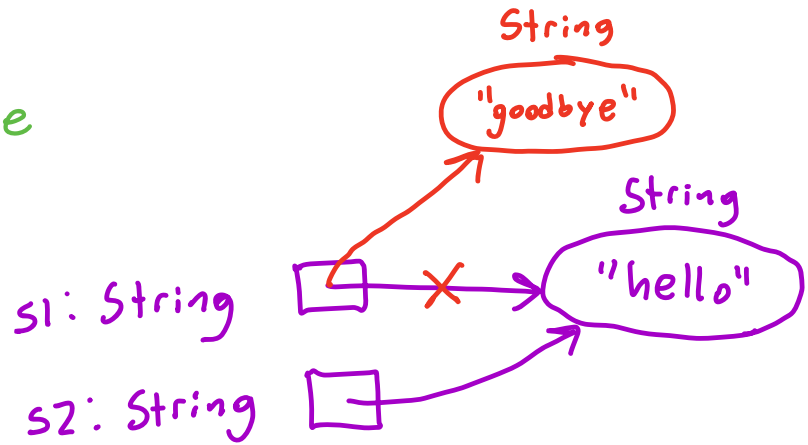
Reference Semantics

The value of a reference type variable is a reference (i.e., an arrow).

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

alias reference

```
1  String s1 = "hello";
2  String s2 = s1;
3  s1 = "goodbye";
4  System.out.print(s2);
```

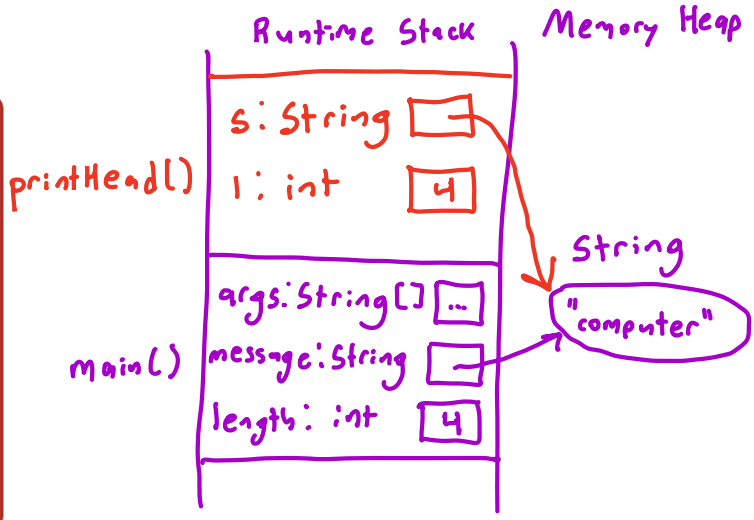


Reference Type Parameters

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

"pass by value semantics"

```
1  void printHead(String s, int l) {
2    System.out.print(s.substring(0, l));
3  }
4
5  public static void main(String[] args) {
6    String message = "computer";
7    int length = 4;
8    printHead(message, length);
9  }
```



Poll Everywhere

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

What will be printed by the following code snippet?

```
1  static void pluralize(String str) {
2    str = str + "s";
3  }
4
5  public static void main(String[] args) {
6    String animal = "fox";
7    pluralize(animal);
8    System.out.print(animal);
9  }
```

fox **(A)**

foxs **(B)**

foxes **(C)**

animal **(D)**

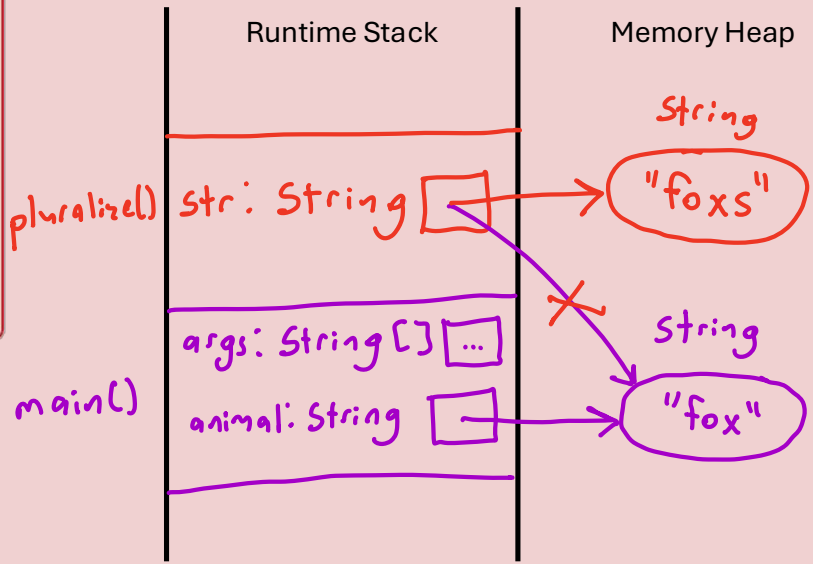
Poll Everywhere

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

What will be printed by the following code snippet?

```
1  static void pluralize(String str) {
2    str = str + "s";
3  }
4
5  public static void main(String[] args) {
6    String animal = "fox";
7    pluralize(animal);
8    System.out.print(animal); // "fox"
9  }
```

concatenation makes new String



null

= a keyword in Java that is a special value for every reference type. It indicates the absence of a reference.

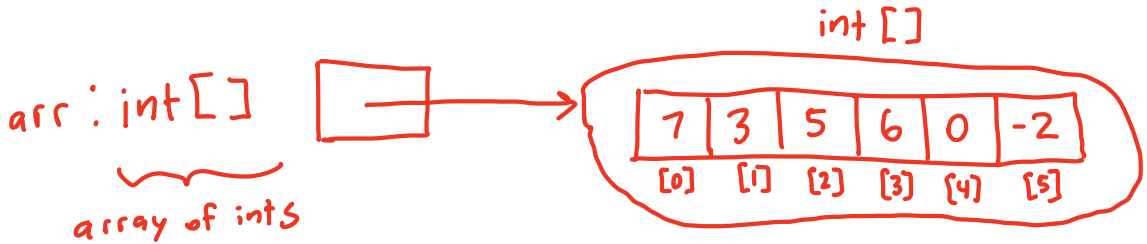
```
String s = null;
```

Trying to invoke an instance method on a `null` target will cause a `NullPointerException` at runtime.

* not caught by compiler
* need to be very careful when variables might be null

Arrays

= simple data structures that consist 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

When you construct an array, you must specify its length

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

Cells are 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};
```

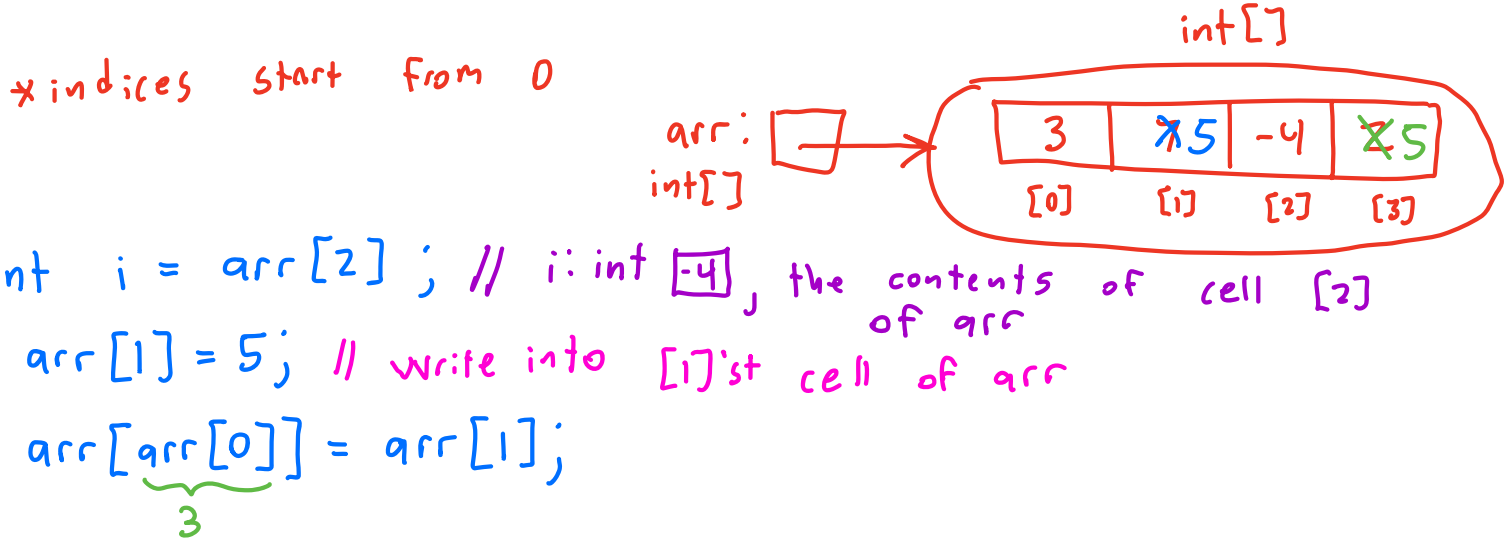
or

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

Array Indexing

To access the entries of an array, use square brackets with the index inside.

This syntax works on both sides of an assignment statement.

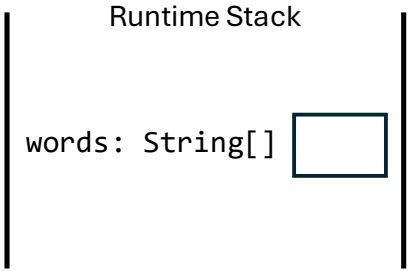


Exercise

Finish drawing the memory diagram for the following code snippet:

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

Memory Heap

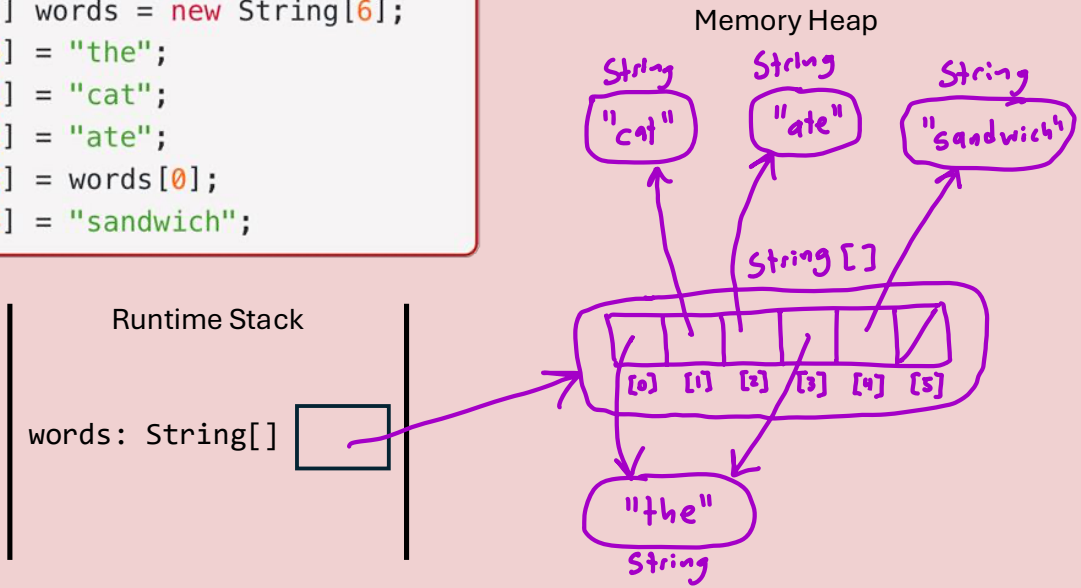


Poll Everywhere

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

How many objects are allocated on the heap by this code?

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



4 (A)

5 (B)

6 (C)

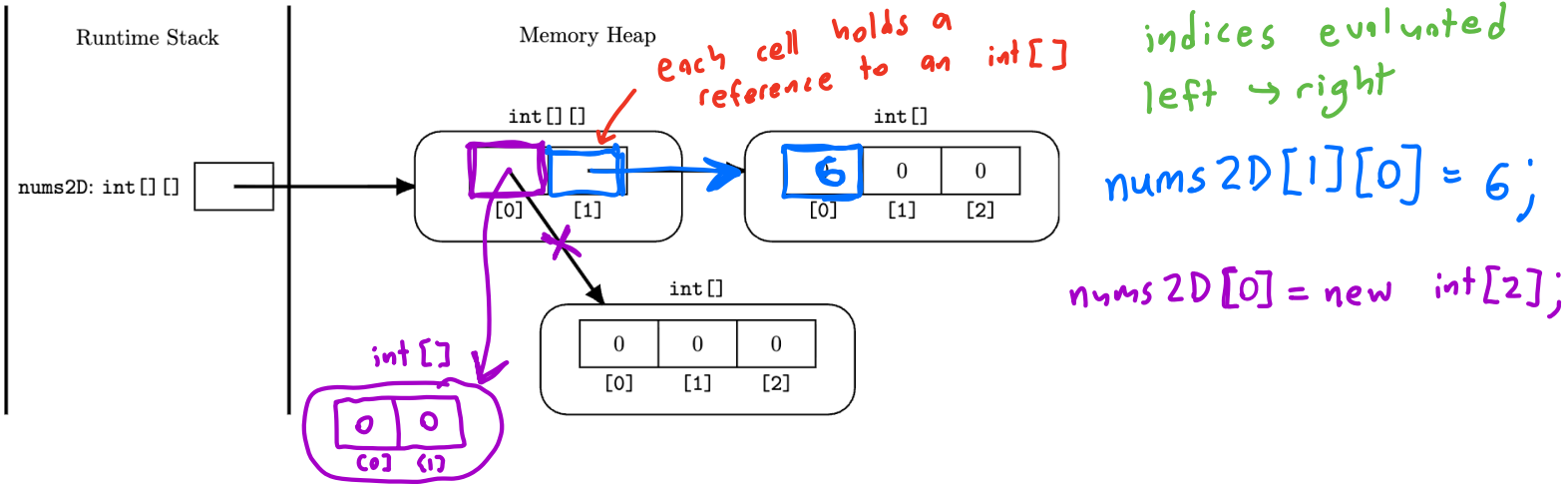
7 (D)

Multi-Dimensional Arrays

= array of arrays

outer size inner size

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



Program Arguments

Recall: The `main()` method serves as the entry point for a Java program.

```
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

IntelliJ: set in run configurations menu (see demo + IntelliJ guide)

Coding Demo: Echo Program