1. Which of the following declaration statements create a reference type? (Check all that are correct.)

Show Results   22/24 Students Answered

A int x = 5;   **primitive -- value type**

B boolean isOn = true;   **primitive -- value type**

C String message = "Hello";   ✓   **reference type**

D int[] scores = new int[5];   ✓   **reference type**

E I don't know.

3. When a reference variable does not point to an object, it's value is _____.

Hide Answers   Show Names   21/24 Students Answered

null	👁
0	👁
null	👁
0	👁
null	👁
null	👁
primitive	👁
null	👁

String name;   **stack**
name - null

int[] myArray;   **myArray - null**

3. When a reference variable does not point to an object, it's value is _____.

Hide Answers   Show Names   21/24 Students Answered

null	👁
0	👁
null	👁
0	👁
null	👁
null	👁
primitive	👁
null	👁
null	👁

String name;   **stack**
name - null   **The value is null because java doesn't know how many bytes you will need**

int[] myArray;   **myArray - null**

int num = 0;   **RAM**

stack
store primitive types are stored

heap
|

Strings are immutable -- they cannot change

String name = "Margaret";

name = "Bob";

stack   heap

name

"Margaret"
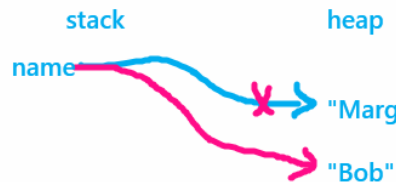
"Bob"

Arrays are immutable

Arrays are fixed in size

---

int[] myArray = new int[5];
int[] temp = new int[6];
//copy each element from myArray into temp
// rename temp to myArray

Strings are immutable -- they cannot change

String name = "Margaret";

name = "Bob";

name = name + "Smith";

stack   heap

name

"Margaret"   garbage collector
              will come along
              and release the
"Bob"   memory

Arrays are immutable

Arrays are fixed in size

5. String s1 = "ALL MEN ARE CREATED EQUAL";
String s2 = s1.substring(16, 19);

0 1 2345 6789101112

16

System.out.println(s2);   T E D

substring(starting index, one past the ending index);

What is the output of the preceding code snippet?

TED

Hide Answers	Show Names	21/24 Students Answered

TED	👁
TED	👁
E, U	👁

---

6. char[] awesomeArray = new char[] { 'A', 'w', 'e', 's', 'o', 'm', 'e' };
String awesomeString1 = new String(awesomeArray);
String awesomeString2 = new String(awesomeArray);

new operator will give you a new location in the heap

boolean areEqual = awesomeString1 == awesomeString2;

compare that they have the same address
or reference the same location in the heap

What is the value of the *areEqual* variable after this code executes?

== comparison operator for primitive types

.equals method needs to be used to compare reference types

Show Results	21/24 Students Answered

A true

B false

String answer = ch1 + "" + ch2

C I don't know

ⓘ No Explanation

---

6. char[] awesomeArray = new char[] { 'A', 'w', 'e', 's', 'o', 'm', 'e' };
String awesomeString1 = new String(awesomeArray);
String awesomeString2 = new String(awesomeArray);

new operator will give you a new location in the heap

String st1 = "Awesome";
String st2 = new String("Awesome");

boolean areEqual = awesomeString1 == awesomeString2;

compare that they have the same address
or reference the same location in the heap

What is the value of the *areEqual* variable after this code executes?

== comparison operator for primitive types

.equals method needs to be used to compare reference types

Show Results	21/24 Students Answered

A true

B false

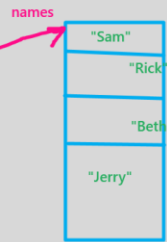String answer = ch1 + "" + ch2

C I don't know

System.out.println(awesomeString);

ⓘ No Explanation

String name = new String("Bob");

List <T> objectName = new ArrayList<>();

Programming to an Interface

```java
List <String> names = new ArrayList<>();
```
Interface   Class

```java
names.add("Rick");
names.add("Beth");
names.add("Jerry");
names.add(0, "Sam");

for (int i = 0; i < names.size(); i++) {
    System.out.println(names.get(i));
}
```

names

| "Sam" |
| "Rick" |
| "Beth" |
| "Jerry" |

- T stands for data type
- The add method is overloaded – add name is the same, but takes in different parameter listings


```
####################
Lists can be sorted
####################
Bilbo    0
Gandalf  1
Merry    2
Pippin   3
Sam      4
####################
Lists can be reversed too
####################
Sam      0
Pippin   1
Merry    2          Indexes have changed permanently
Gandalf  3
Bilbo    4
####################
        FOREACH
####################
```