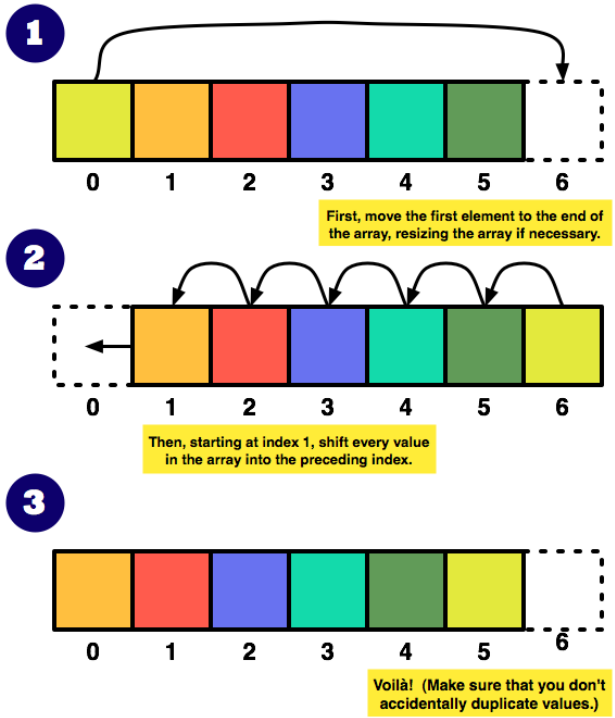
API



Java
String Class

Working with String Objects

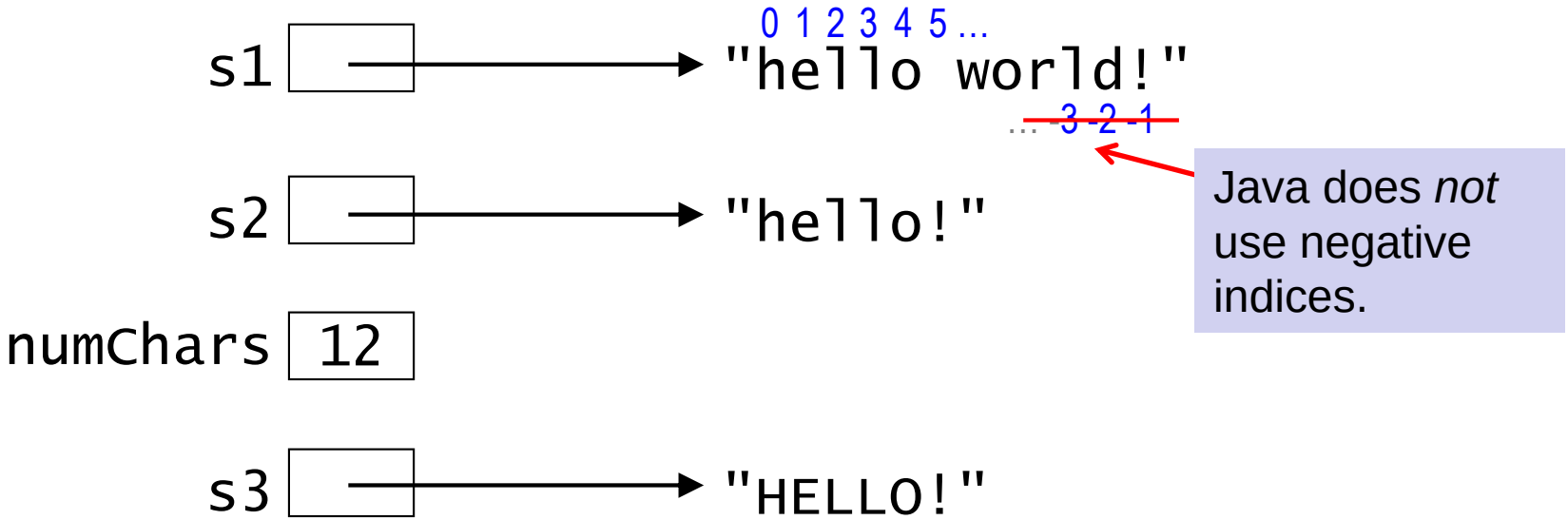
Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
     +  s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



The API of a Class

- The methods defined within a class are known as the *API* of that class.
  - API = application programming interface

- We can consult the API of an existing class to determine which operations are supported.

- The API of all classes that come with Java is available here:
  `https://docs.oracle.com/javase/8/docs/api/`
  - there's a link on the resources page of the course website

Consulting the Java API

select the class name

String (Java 2 Platform SE 5.0) - Mozilla Firefox

File Edit View Go Bookmarks Tools Help 5.7 hours saved

http://java.sun.com/j2se/1.5.0/docs/api/index.html

java.awt.print
java.beans
java.beans.beanconte
java.io
java.lang
java.lang.annotation
java.lang.instrument

SecurityManager
Short
StackTraceElement
StrictMath
String
StringBuffer
StringBuilder
System
Thread
ThreadGroup
ThreadLocal
Throwable
Void

Enums
Thread.State

Exceptions
ArithmeticException

Overview Package Class Use Tree Deprecated Index Help

Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.lang

Class String

```
java.lang.Object
  └ java.lang.String
```

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

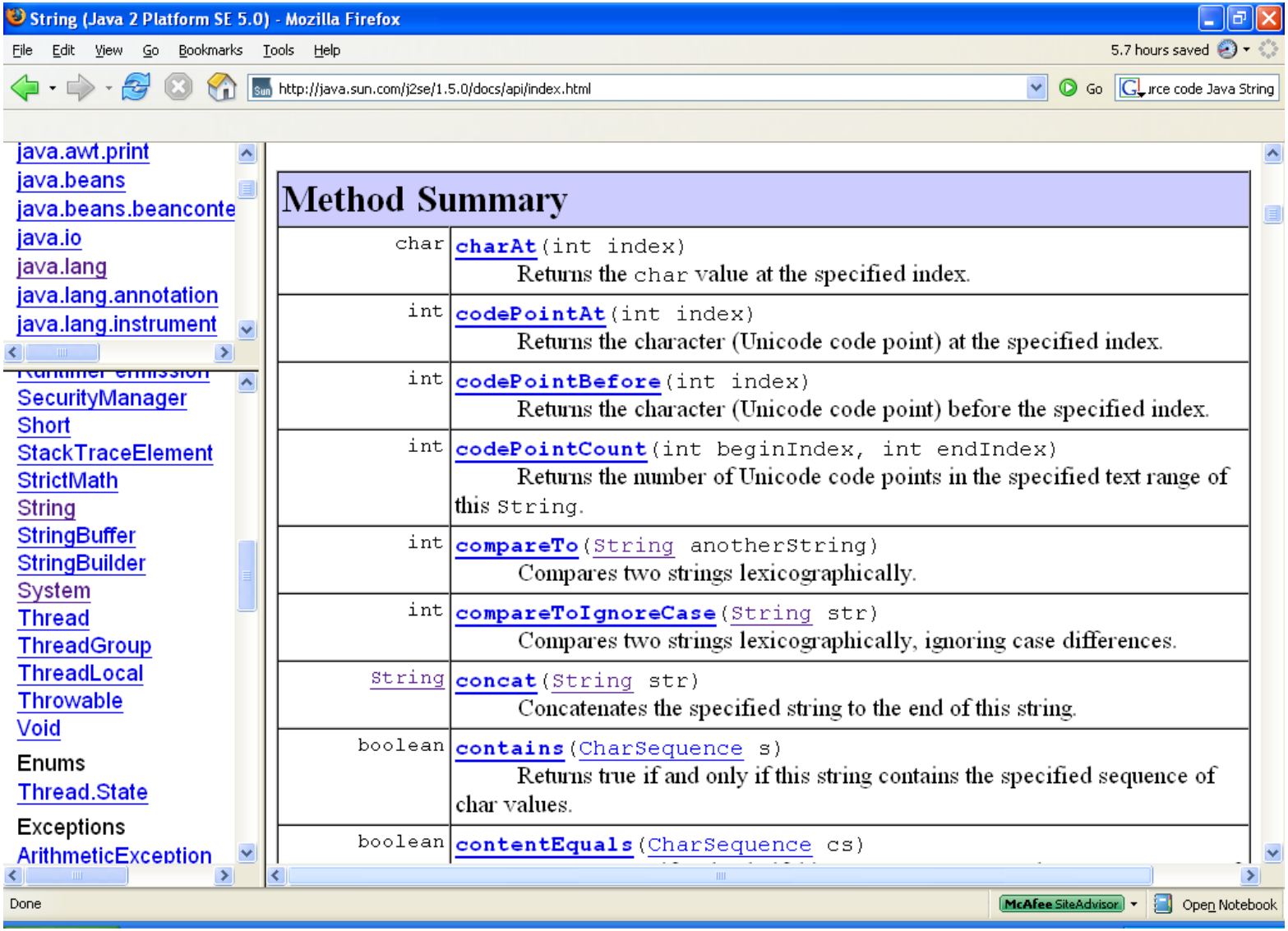
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

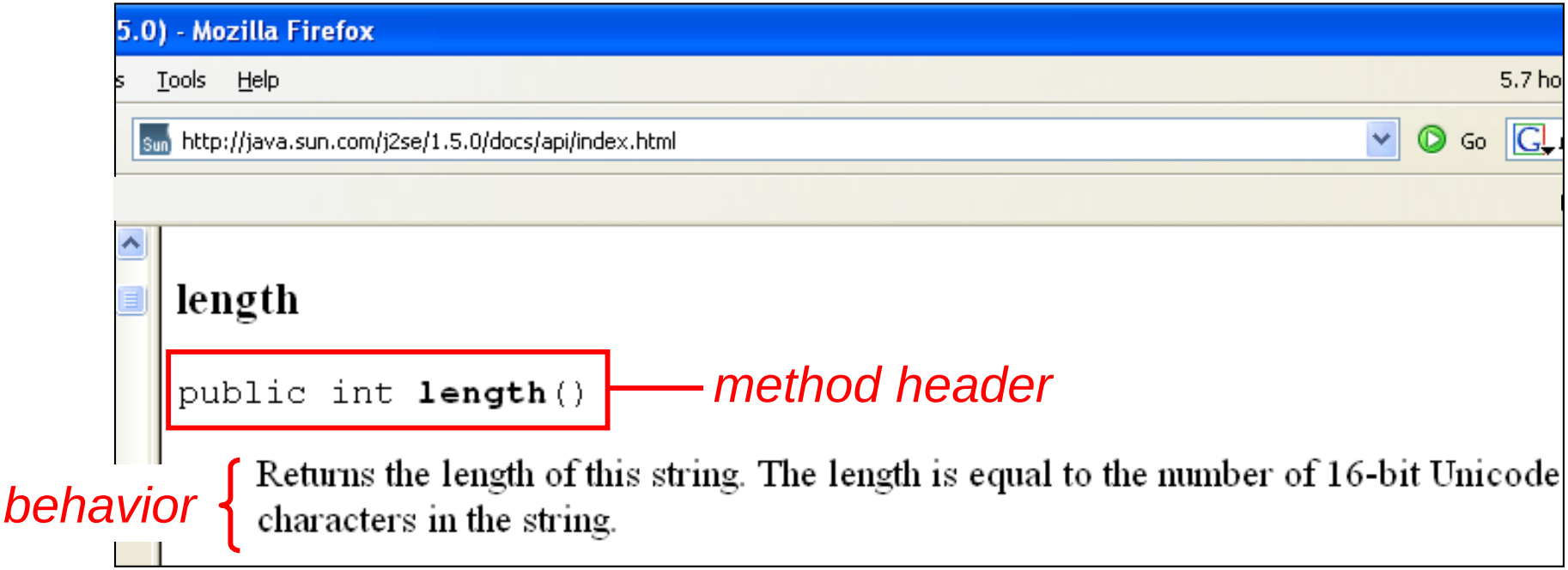
Consulting the Java API (cont.)

- Scroll down to see a summary of the available methods:



Consulting the Java API (cont.)

- Clicking on a method name gives you more information:



- From the header, we can determine:
  - the return type: `int`
  - the parameters we need to supply:
    the empty `()` indicates that `length` has no parameters

`substring` Method

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new 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`.

<u>`String substring(int beginIndex, int endIndex)`</u>

- return type: `String`
- parameters: `(int beginIndex, int endIndex)`
- behavior: returns the substring that:
  - begins at `beginIndex`
  - ends at `endIndex` – `1`

Representing Individual Characters

- The `char` type is used to represent individual characters.
- It is a **primitive** type.
- To specify a `char` literal, we surround the character by single quotes:
  - examples: `'a'` `'z'` `'0'` `'7'` `'?'`
  - can only represent single characters
  - don't use double-quotes!

```
"a" is a string
'a' is a character
```

charAt Method

charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ra

<u>`char charAt(int index)`</u>

- return type: `char`
- parameters: `(int index)`
- behavior: returns the character at positi ne string

Recall that the operation performed is based on the types of the operand (s).

- We have to be careful when we use its return va

- Example: What does this print?

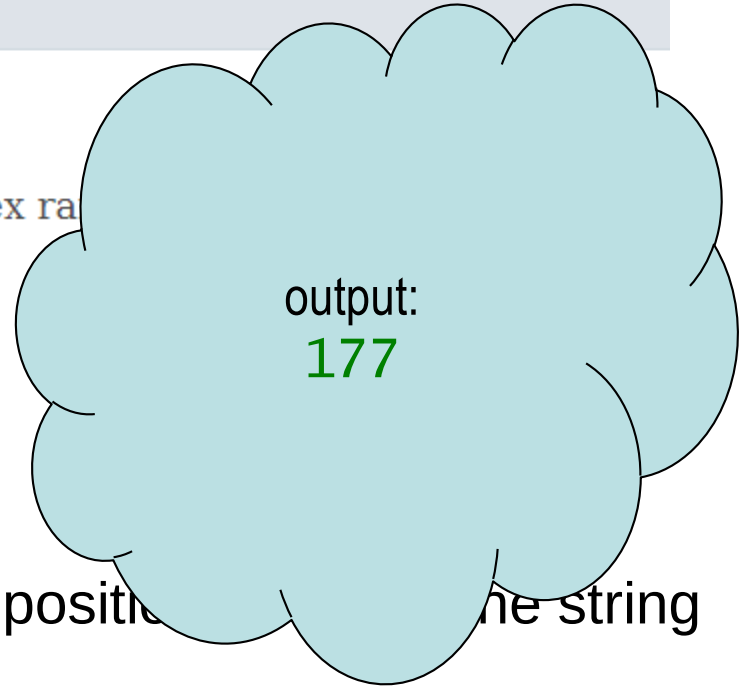
```
String name = "Evangeline Kanaris";
System.out.println( name.charAt(0) +
                    name.charAt(6) );
```

charAt Method

charAt

`public char charAt(int index)`

Returns the `char` value at the specified index. An index ra



`char charAt(int index)`

- return type: `char`
- parameters: `(int index)`
- behavior: returns the character at positi ne string

- We have to be careful when we use its return va

  - Example: What does this print?

```
String name = "Evangeline Kanaris";
System.out.println( name.charAt(0) +
                    name.charAt(6) );
```

charAt Method

charAt

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ra

sum of the
ASCII codes
for 'E' and 'l'

<u>char charAt(int index)</u>

- return type: char
- parameters: (int index)
- behavior: returns the character at positi ne string

- We have to be careful when we use its return va

- Example: What does this print?

```
String name = "Evangeline Kanaris";
System.out.println( name.charAt(0) +
                    name.charAt(6) );
```

charAt Method

- Example: Now what does this print?

```
String name = “Evangeline";
System.out.println(name.charAt(0) + "" +
                   name.charAt(6));
```

charAt Method

- Example: Now what does this print?

```
String name = “Evangeline";
System.out.println(name.charAt(0) + "" +
                   name.charAt(6));
```

⇓

```
System.out.println('E' + "" + 'l');
```

⇓

charAt Method

- Example: Now what does this print?

```
String name = “Evangeline";
System.out.println(name.charAt(0) + "" +
                   name.charAt(6));
```

⬇

```
System.out.println('E' + "" + 'l');
```

⬇

```
System.out.println("E" + 'l');
```

charAt Method

- Example: Now what does this print?

```
String name = “Evangeline";
System.out.println(name.charAt(0) + "" +
                   name.charAt(6));
```

⇓

```
System.out.println('E' + "" + 'l');
```

⇓

```
System.out.println("E" + 'l');
```

⇓

```
System.out.println("El");
```

charAt Method

- Example: Now what does this print?

```
String name = “Evangeline";
System.out.println(name.charAt(0) + "" +
                   name.charAt(6));
```

```
System.out.println('E' + "" + 'l');
```

```
System.out.println("E" + 'l');
```

```
System.out.println("El");
```

Note that the operation performed is based on the type of the operand. And since one operand is a string, string concatenation is performed!

Which of these correctly fills in the blank?

charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`.

```
String s = "PS 1 is due next Friday!";
int len = s.length();
_________________  // get the last character in s
```

A. ~~String last = s.charAt(int len - 1);~~

B. ~~String last = s.charAt(len - 1);~~

C. `char last = s.charAt(~~int~~ len - 1);`

D. **`char last = s.charAt(len - 1);`**

E. more than one of them

`indexOf` Method

`int indexOf(char ch)`

- return type: `int`
- parameter list: `(char ch)`
- returns:
  - the index of the first occurrence of `ch` in the string
  - -1 if the `ch` does not appear in the string
- examples:

```
String name = “Evangeline";
System.out.println(name.indexOf(‘v'));
System.out.println(name.indexOf('X'));
```

`indexOf` Method

`int indexOf(char ch)`

- return type: `int`
- parameter list: `(char ch)`
- returns:
  - the index of the first occurrence of `ch` in the string
  - -1 if the `ch` does not appear in the string
- examples:

```
String name = “Evangeline";
System.out.println(name.indexOf(‘v'));
System.out.println(name.indexOf('X'));
```

`indexOf` Method

`int indexOf(char ch)`

- return type: `int`
- parameter list: `(char ch)`
- returns:
  - the index of the first occurrence of `ch` in the string
  - -1 if the `ch` does not appear in the string
- examples:

```
String name = “Evangeline";
System.out.println(1);
System.out.println(name.indexOf('X'));
```

`indexOf` Method

`int indexOf(char ch)`

- return type: `int`
- parameter list: `(char ch)`
- returns:
  - the index of the first occurrence of `ch` in the string
  - -1 if the `ch` does not appear in the string
- examples:

```
String name = “Evangeline";
System.out.println(1);
System.out.println(-1);
```

The Signature of a Method

- The *signature* of a method consists of:
  - its name
  - the number and types of its parameters

```
public String substring(int beginIndex, int endIndex)
```

the signature

- A class cannot include two methods with the same signature.

Two Methods with the Same Name

- There are actually two `String` methods named `substring`:

  ```
  String substring(int beginIndex, int endIndex)
  ```

  ```
  String substring(int beginIndex)
  ```

  - returns the substring that begins at `beginIndex` and continues to the end of the string

- Do these two methods have the same signature?
  no, because they don't have the same number of parameters

- Giving two methods the same name is known as *method overloading*.

- When you call an overloaded method, the compiler uses the *signature of the method* (i.e. number and types of the actual parameters) to figure out which version to use.

Recall:
Testing for Equivalent *Primitive* Values

- The `==` and `!=` operators are used to compare primitives.
  - `int`, `double`, `char`, etc.

Comparing
the value stored
at the address
location associated
with the variable.

Example:
Testing for Equivalent *Primitive* Values

- The `==` and `!=` operators are used to compare primitives.
  - `int`, `double`, `char`, etc.

- Example:

```
Scanner console = new Scanner(System.in);
System.out.print("Choose an option: ");
int choice = console.nextInt();
if (choice == 1) {    // this works just fine
    playSudoku();
} else if (choice == 2) {
    playChess();
} else {
    System.out.println("invalid input");
}
```

Example:
Testing for Equivalent *Primitive* Values

- The `==` and `!=` operators are used to compare primitives.
  - `int`, `double`, `char`, etc.

- Example:

```
Scanner console = new Scanner(System.in);
System.out.print("Choose an option: ");
int choice = console.nextInt();
if (choice == 1) {    // comparing two int values
    playSudoku();
} else if (choice == 2) {
    playChess();
} else {
    System.out.println("invalid input");
}
```

Example:
Testing for Equivalent *String* Objects

- The `==` and `!=` operators do *not* typically work when comparing *objects*.

- Example:

```
Scanner console = new Scanner(System.in);
System.out.print("regular or diet? ");
String choice = console.next();
if (choice == "regular") {  // doesn't work
    processRegular();
} else {
    ...
}
```

Example:
Testing for Equivalent *String* Objects

- The `==` and `!=` operators do *not* typically work when comparing *objects*.

- Example:

```
Scanner console = new Scanner(System.in);
System.out.print("regular or diet? ");
String choice = console.next();
if (choice == "regular") {  // doesn't work
    processRegular();
} else {
    ...
}
```

- `choice == "regular"` compiles, but it evaluates to `false`, even when the user does enter `"regular"`!
  - What is stored in the memory location of variable choice?
  - The address location of the string entered!

Example:
Testing for Equivalent *String* Objects

- We use a special method called the **`equals`** method to test if two objects are equivalent.
  - example:

```
Scanner console = new Scanner(System.in);
System.out.print("regular or diet? ");
String choice = console.next();
if (choice.equals("regular")) {
    processRegular();
} else {
    ...
}
```

- **`choice.equals("regular")`** compares the string represented by the variable `choice` with the string `"regular"`
  - returns `true` when they are equivalent
  - returns `false` when they are not

`equalsIgnoreCase()`

- We often want to compare two strings without paying attention to the case of the letters.
  - example: we want to treat as equivalent:

    `"regular"`

    `"Regular"`

    `"REGULAR"`

    etc.

- The `String` class has a method called `equalsIgnoreCase` that can be used for this purpose:

```
if (choice.equalsIgnoreCase("regular")) {
    ...
}
```

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

Style Convention

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- When a name in Python has more than one word, the convention is to separate the words with _ characters.

- In Java, we instead capitalize the first letter of each new word.

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

- Python has a `len()` function that takes a string as input.

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- In Java, strings have a `length()` method.
  - inside the `String` object
  - a *non-static* method

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

- Python has special operators for slicing and indexing strings.

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- In Java, we use non-static methods instead:
  - `substring(start, end)` for slicing

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

- Python has special operators for slicing and indexing strings.

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- In Java, we use non-static methods instead:
  - `substring(start, end)` for slicing
  - `charAt(index)` for indexing

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

- Python has special operators for slicing and indexing strings.

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- In Java, we use non-static methods instead:
  - `substring(start, end)` for slicing
  - `charAt(index)` for indexing
- We can't use negative indices.
  - use `s.length() - 1` for the last character

More about String Objects

Python

```
s1 = "hello"
s2 = "world!"
s1 = s1 + " " + s2
num_chars = len(s1)
s2 = s1[0:5] + s1[-1]

s3 = s2.upper()
```

- Python strings also have methods.

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

- In Java, the equivalent methods often have different names.

After running this, what is the value of `s3`?

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```

A. `"helloo"`

B. `"HELLOO"`

C. `"HELLOL"`

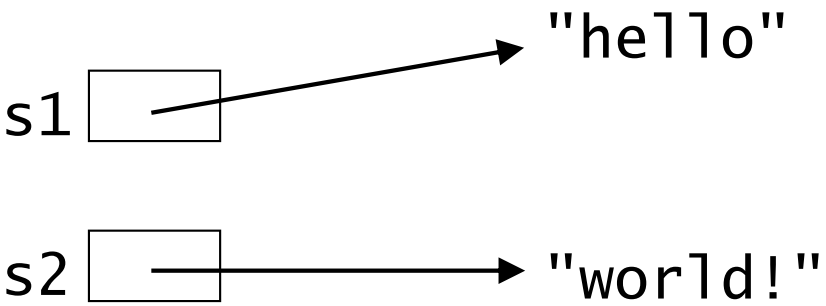
D. `"HELLO!"`

E. `"hello!"`

After running this, what is the value of s3?

Java

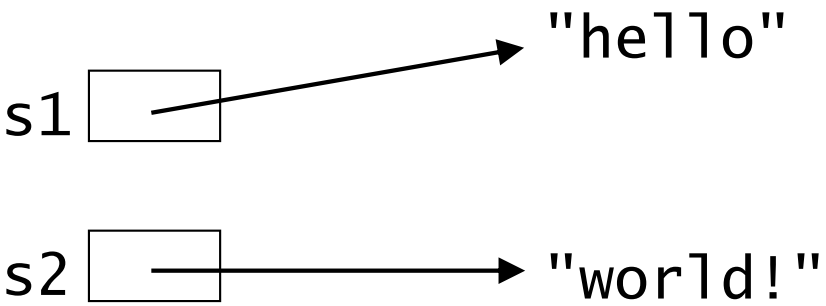
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

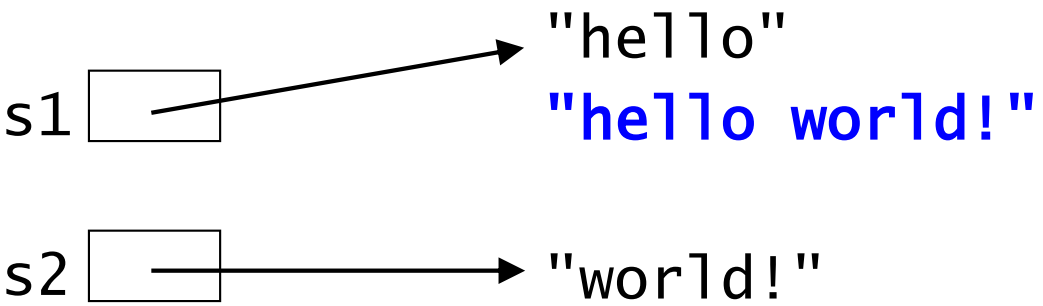
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

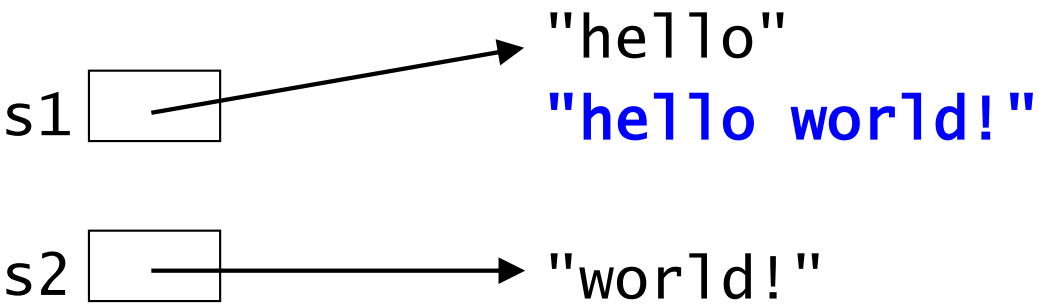
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

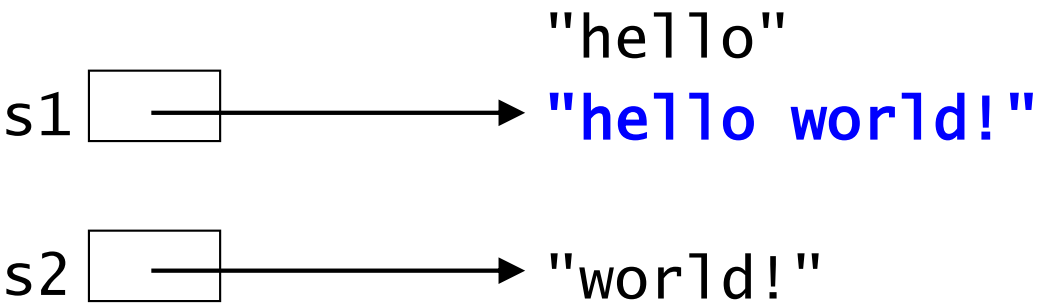
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

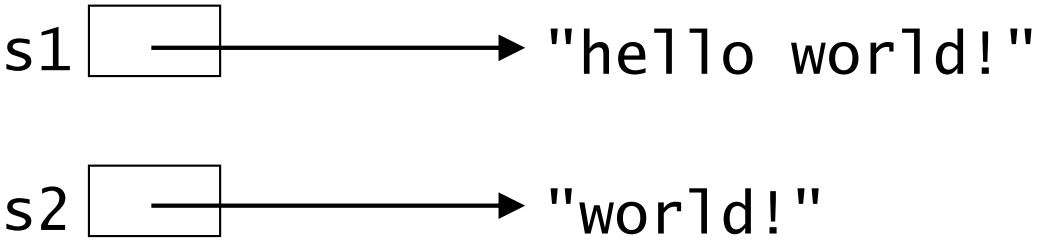
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

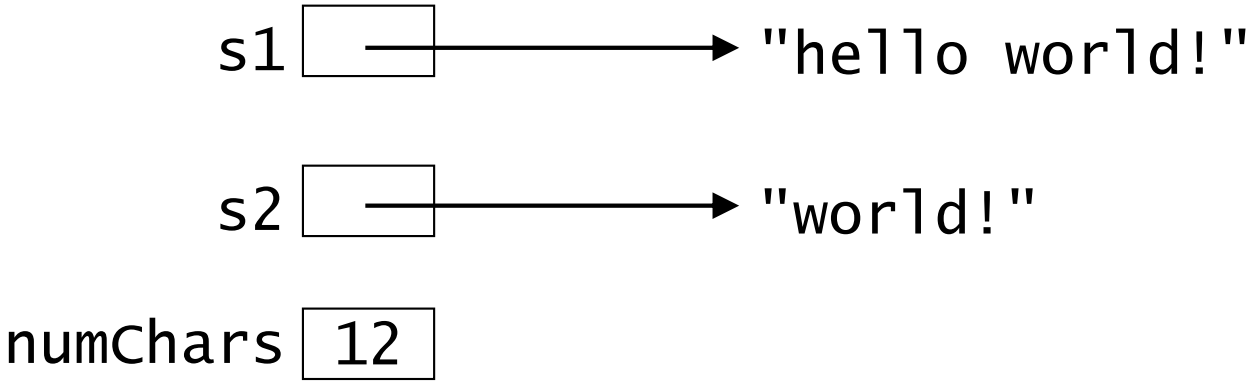
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

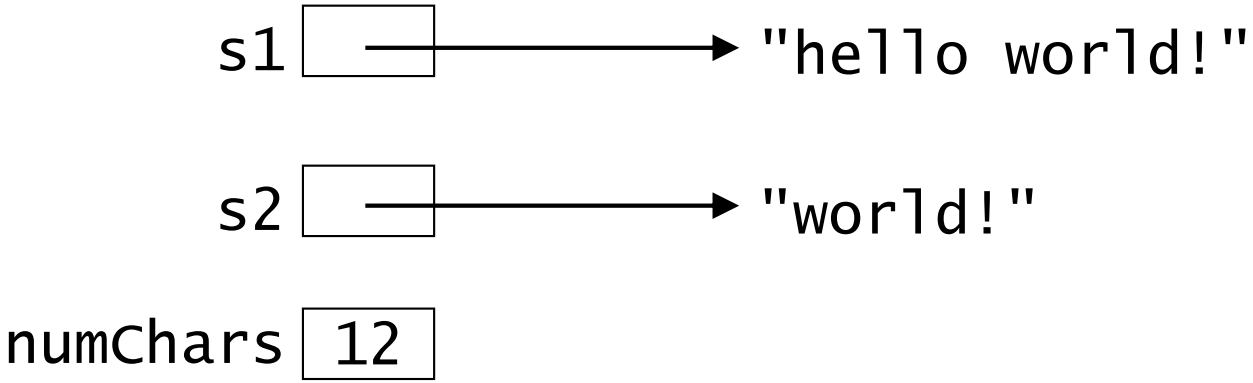
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

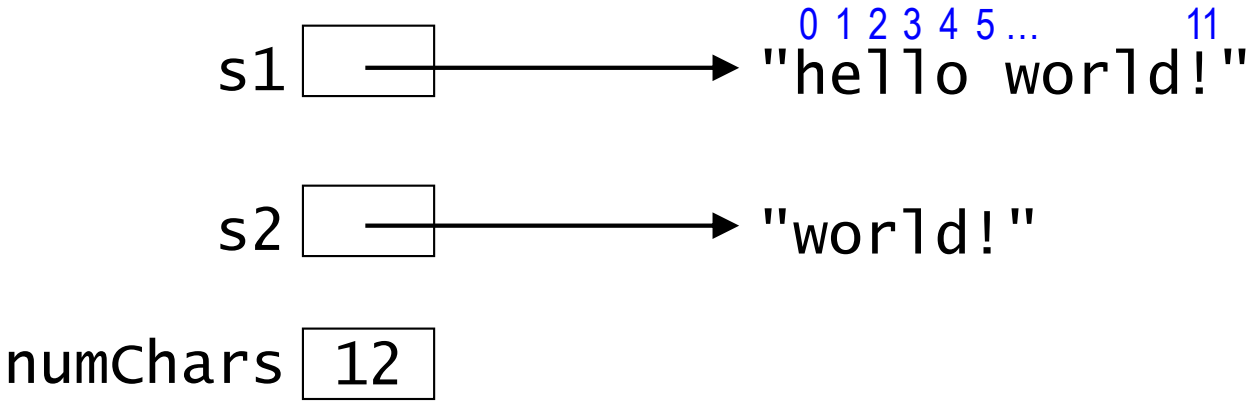
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

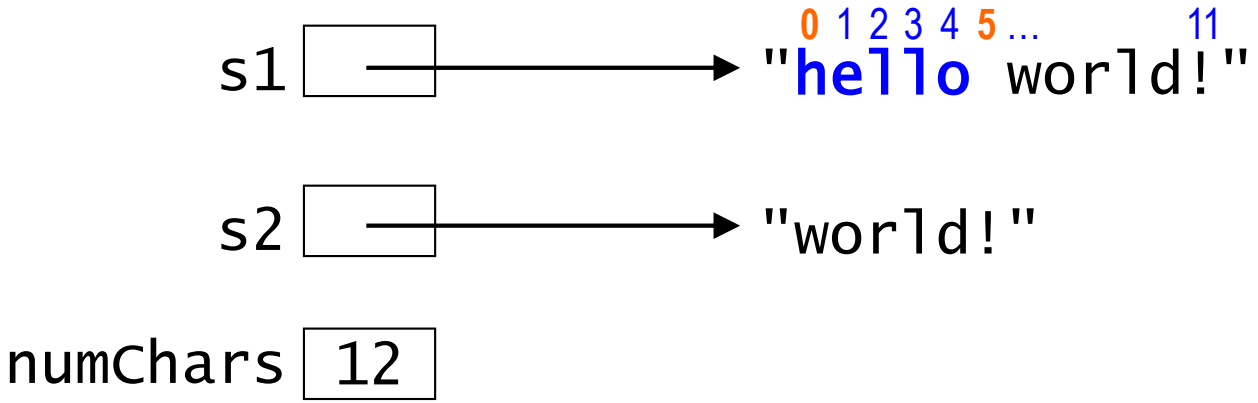
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

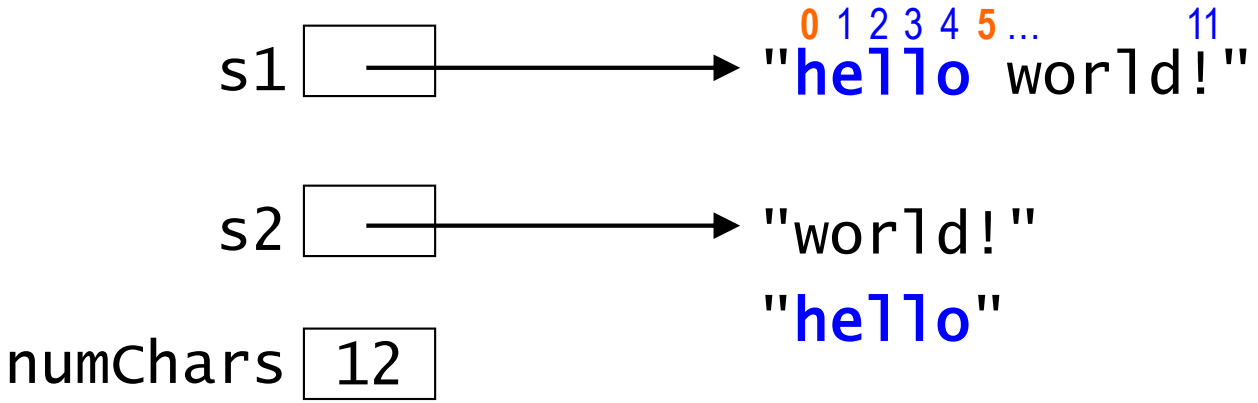
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

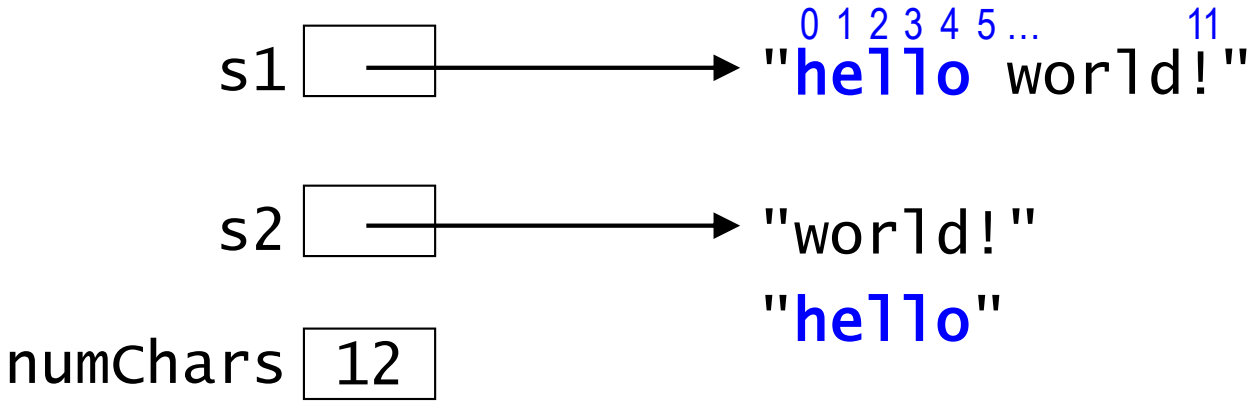
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

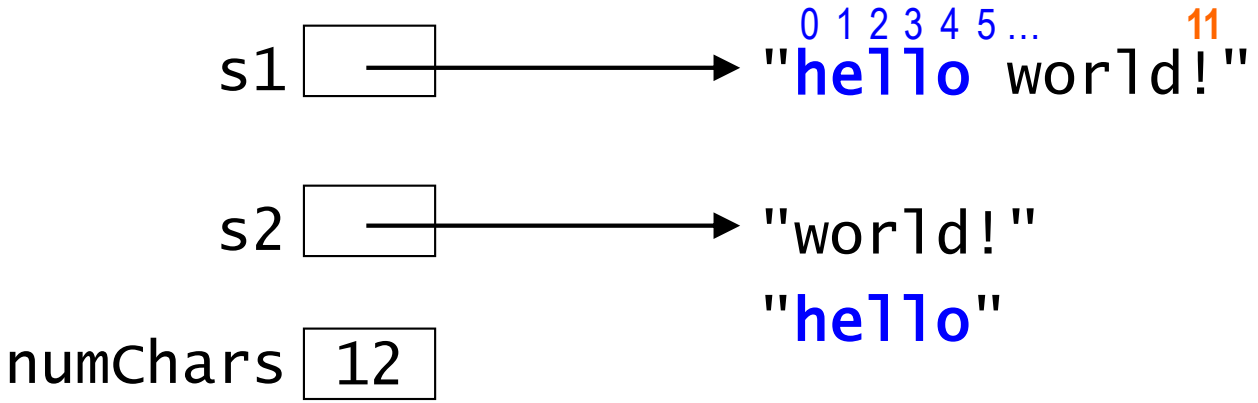
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

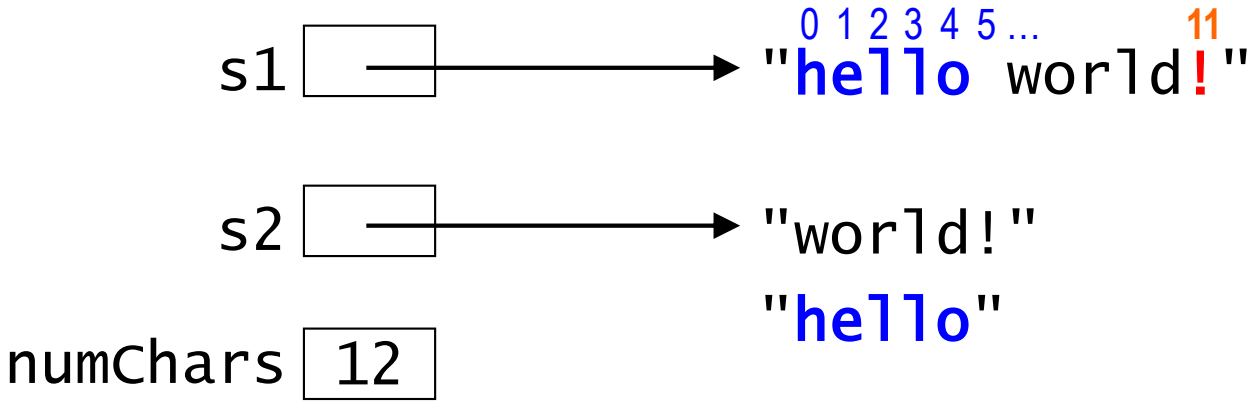
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

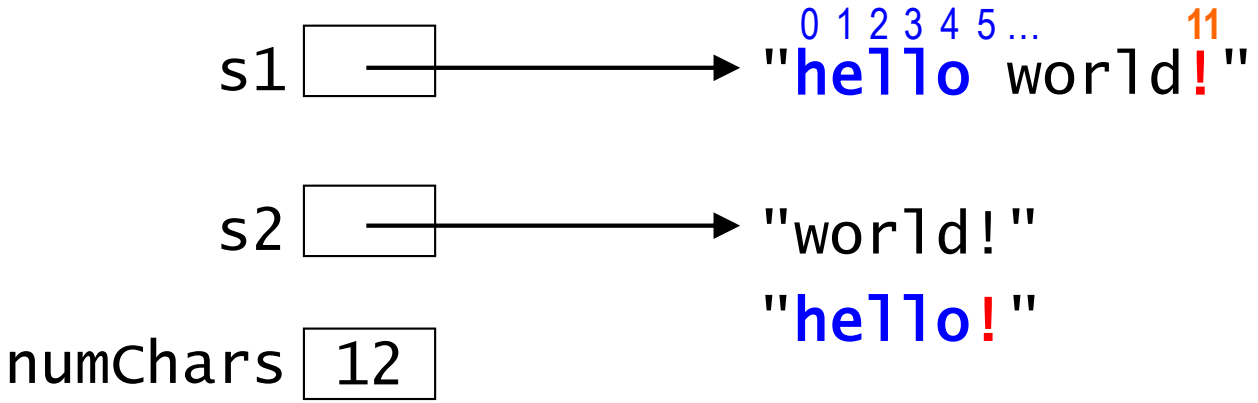
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

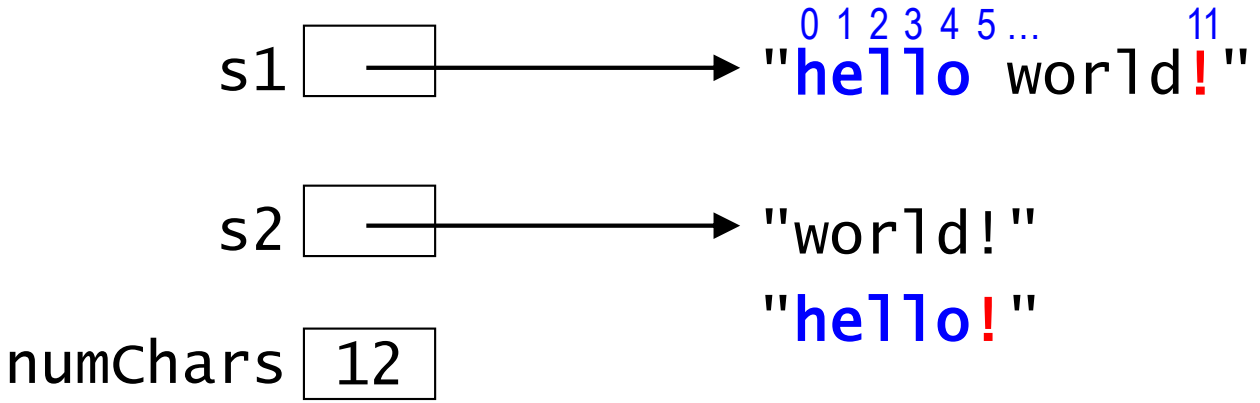
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

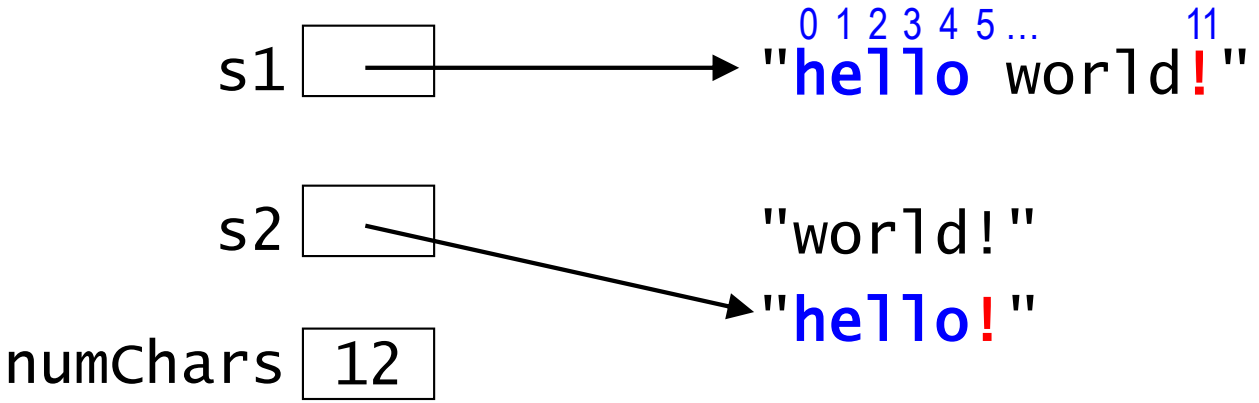
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

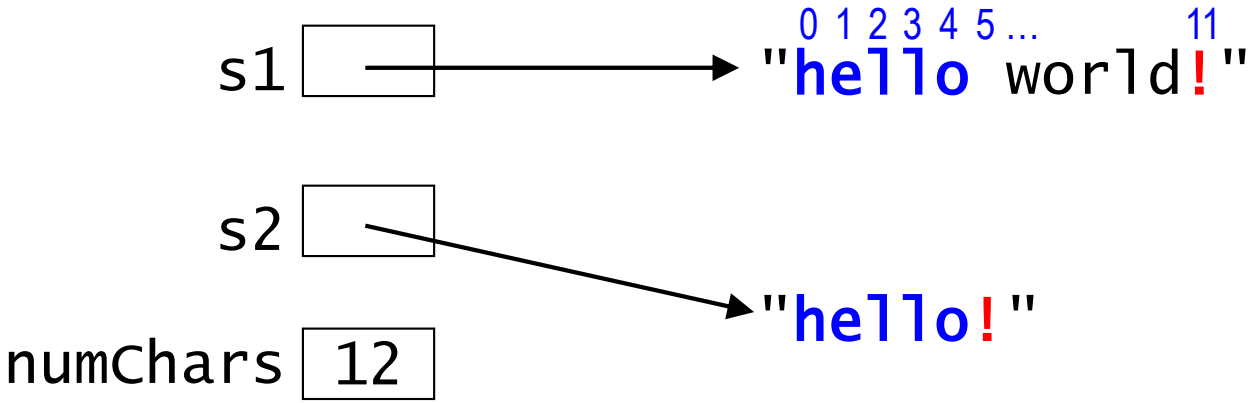
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

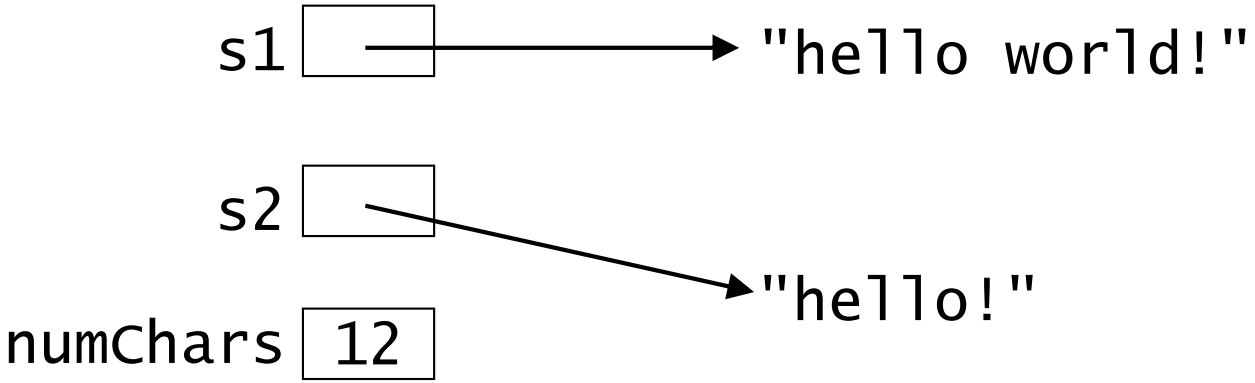
```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```



After running this, what is the value of s3?

Java

```
String s1 = "hello";
String s2 = "world!";
s1 = s1 + " " + s2;
int numChars = s1.length();
s2 = s1.substring(0, 5)
      + s1.charAt(numChars - 1);
String s3 = s2.toUpperCase();
```