# Strings and Loops

## CSE 1020

moodle.yorku.ca

# Strings

Strings are immutable objects.

The state of an immutable object cannot be changed.

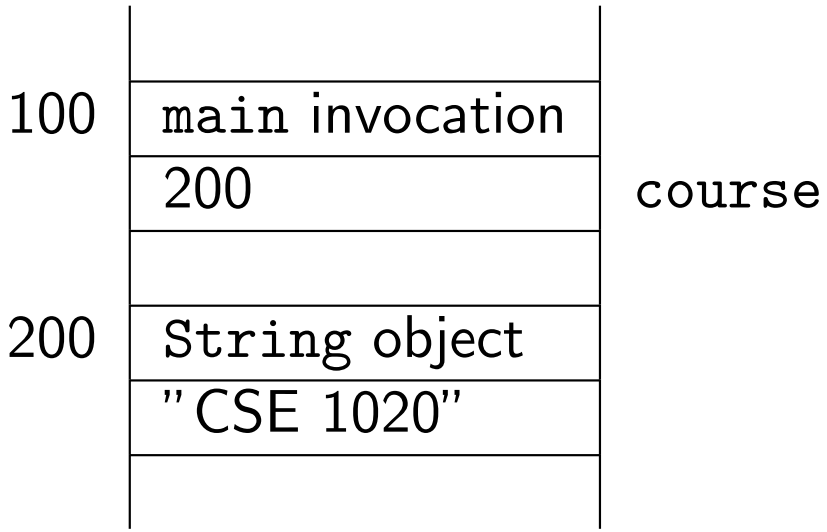
The `String` API does not contain any mutators.

The `StringBuffer` class provides mutable strings. [1]

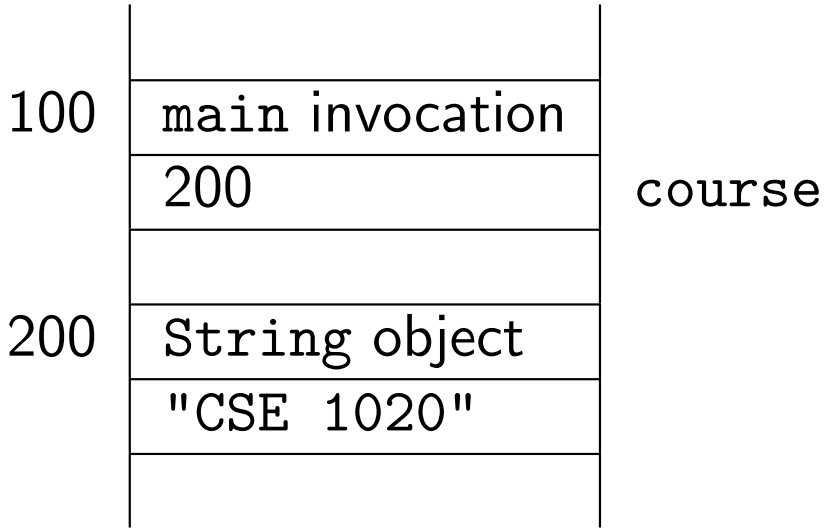
[1] We will come back to the `StringBuffer` class later.

# Strings

String course = new String("CSE 1020");



# Strings




String reference: `course`
String object: object at address 200
String literal: `"CSE 1020"`

# Strings are everywhere

Instead of

```
String course = new String("CSE 1020");
```

we are allowed to write

```
String course = "CSE 1020";
```

Although in most cases you may think of `"CSE 1020"` and `new String("CSE 1020")` as synonyms, they are not always equivalent.[2]

[2] Hardly ever will this difference impact your app.

# Strings are immutable

According to the Java Language Specification,

> *Strings that are the values of constant expressions are "interned" so as to share unique instances*

James Gosling, Bill Joy, Guy L. Steele Jr. and Gilad Bracha. The Java Language Specification. Third edition. Addison-Wesley. 2005.

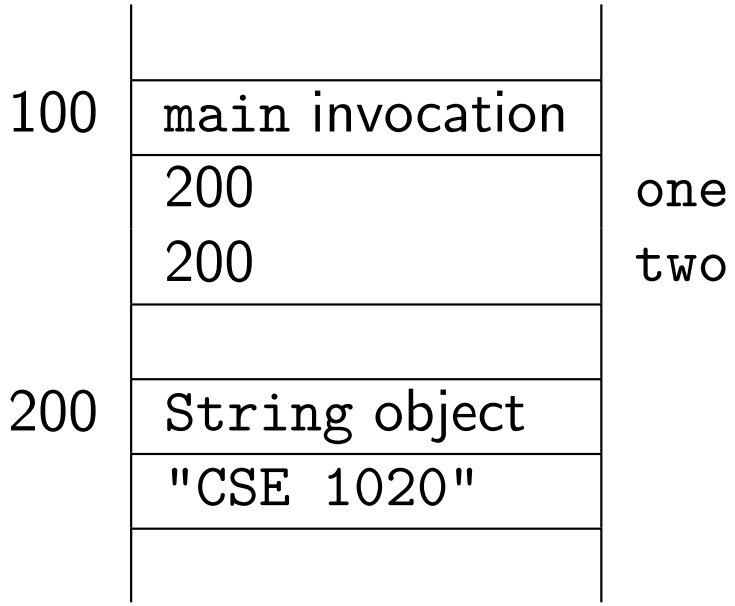
# Strings are immutable

> *Strings that are the values of constant expressions are "interned" so as to share unique instances*

These constant expressions are built from String literals and the binary operator $+$.

# Strings are immutable

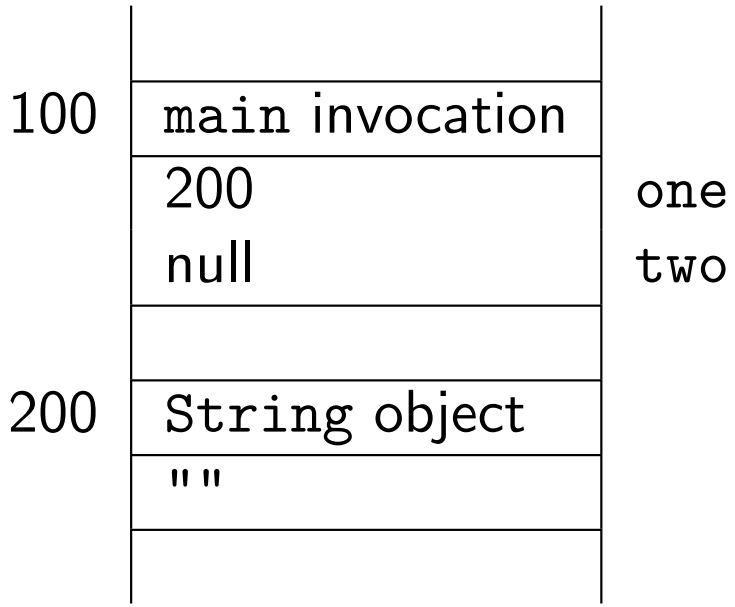
String one = "CSE 1020";
String two = "CSE" + " " + "1020";




This saves memory. Why can `one` and `two` refer to the same `String` object?

# The empty string versus `null`

String one = "";
String two = null;



# My mom . . .

. . . tends to write long sentences. Long sentences are in general more difficult to comprehend.

Rudolf Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3): 221-233, June 1948.

# Long sentences

## Problem

Prompt the user for a file name by printing

```
Enter file name:
```

so that the file name is entered by the user on the same line as the prompt.

You may assume that the file consists of sentences.

Print the last word of every sentence, each on a separate line.

# Long sentences

```
String line = ...
Scanner lineInput = new Scanner(line);
while (lineInput.hasNext())
{
    String token = lineInput.next();
    ...
}
```

# Long sentences

```
String line = ...
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens())
{
    String token = tokenizer.nextToken();
    ...
}
```

# Long sentences

## Problem

Prompt the user for a file name by printing

```
Enter file name:
```

so that the file name is entered by the user on the same line as the prompt.

You may assume that the file consists of sentences.

Print the number of words for every sentence, each on a separate line.

# Long sentences

## Problem

Prompt the user for a file name by printing

```
Enter file name:
```

so that the file name is entered by the user on the same line as the prompt.

You may assume that the file consists of sentences.

Print those sentences that have more than 35 words, each on a separate line.[a]

[a] The New Yorker of October 26, 1946 has on average 20 words per sentence.

# The StringBuffer class

StringBuffers are mutable objects.

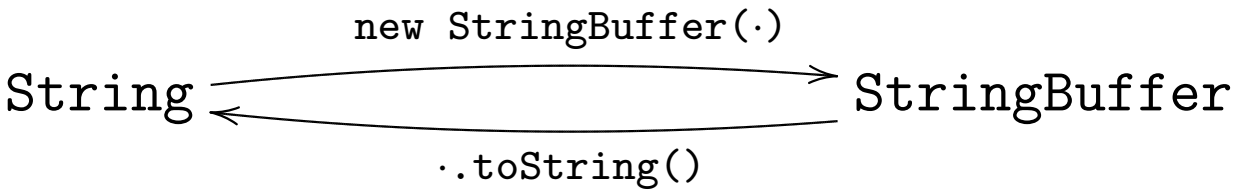
The method

```
public StringBuffer append(String s)
```

adds the String `s` to the end of the StringBuffer.

The method returns a reference to the StringBuffer itself. Although this is not needed (why?), it is convenient.

# From String to StringBuffer and back



# Command-line arguments

## Question

How does the user provide command-line arguments?

# Command-line arguments

## Question

How does the user provide command-line arguments?

## Answer

```
java LongSentences "book.txt"
```

# Command-line arguments

**Question**

How does the user provide command-line arguments?

**Answer**

```
java LongSentences "book.txt"
```

**Question**

How does the client get the command-line arguments?

# Command-line arguments

**Question**

How does the user provide command-line arguments?

**Answer**

```
java LongSentences "book.txt"
```

**Question**

How does the client get the command-line arguments?

**Answer**

As the parameter of the `main` method.

# Command-line arguments

**Question**

How does the user provide command-line arguments?

**Answer**

```
java LongSentences "book.txt"
```

**Question**

How does the client get the command-line arguments?

**Answer**

As the parameter of the `main` method.

**Question**

What is the type of the parameter of the `main` method.

# Command-line arguments

**Question**

How does the user provide command-line arguments?

**Answer**

```
java LongSentences "book.txt"
```

**Question**

How does the client get the command-line arguments?

**Answer**

As the parameter of the `main` method.

**Question**

What is the type of the parameter of the `main` method.

**Answer**

`String[]`: an array of `Strings`.

# Command-line arguments

public static void main(String[] args)

**Question**

How does the client get the first command-line argument?

# Command-line arguments

public static void main(String[] args)

**Question**

How does the client get the first command-line argument?

**Answer**

`args[0]`

# Command-line arguments

public static void main(String[] args)

**Question**

How does the client get the first command-line argument?

**Answer**

`args[0]`

**Question**

How does the client get the second command-line argument?

# Command-line arguments

public static void main(String[] args)

**Question**

How does the client get the first command-line argument?

**Answer**

`args[0]`

**Question**

How does the client get the second command-line argument?

**Answer**

`args[1]`

# Command-line arguments

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

**Question**

How does the client get the number of command-line arguments?

# Command-line arguments

public static void main(String[] args)

**Question**

How does the client get the number of command-line arguments?

**Answer**

`args.length`

# Command-line arguments

## Problem

The file name is provided as a command-line argument.

You may assume that the file consists of sentences.

Print those sentences that have more than 35 words, each on a separate line.

If the user does not provide a command-line argument, print

```
Use:  java LongSentences <file name>
```