CS2110, Recitation 1

Method main,

Packages,

Eclipse,

Characters, and, if there is time,

Strings

Note: Java Objects and classes were introduced in lecture 2, Tuesday, just before this recitation.

Java Applications

Java Applications

public static void main(String[] args) { … }

Parameter: String array

A Java program that has a class with a static procedure main, as declared above, is called an application.

The program, i.e. the application, is run by calling method main. Eclipse has an easy way to do this.

Don't worry about what public static void means.
Concentrate on the Eclipse stuff

Demo: Create application

To create a new project that has a method called main with a body that contains the statement

```
System.out.println("Hello World");
```

do this:

1. Eclipse: File -> New -> Java Project
   Execution environment should be Java 11. Click Finish
   Create module-info.java? No!

2. Highlight directory src. Do File -> New -> Class
   a. Make Package field empty!!!
   b. Give it name C
   c. Check box for public static void main(…)
   d. Click Finish

In the class that is created, write the above println statement in the body of main

1. Hit the green play button or do menu item Run -> Run

Optional:
Putting arguments in the call to method main

Method main and its parameter

public static void main(String[] args) { … }

Parameter: String array

In Eclipse, when you do menu item

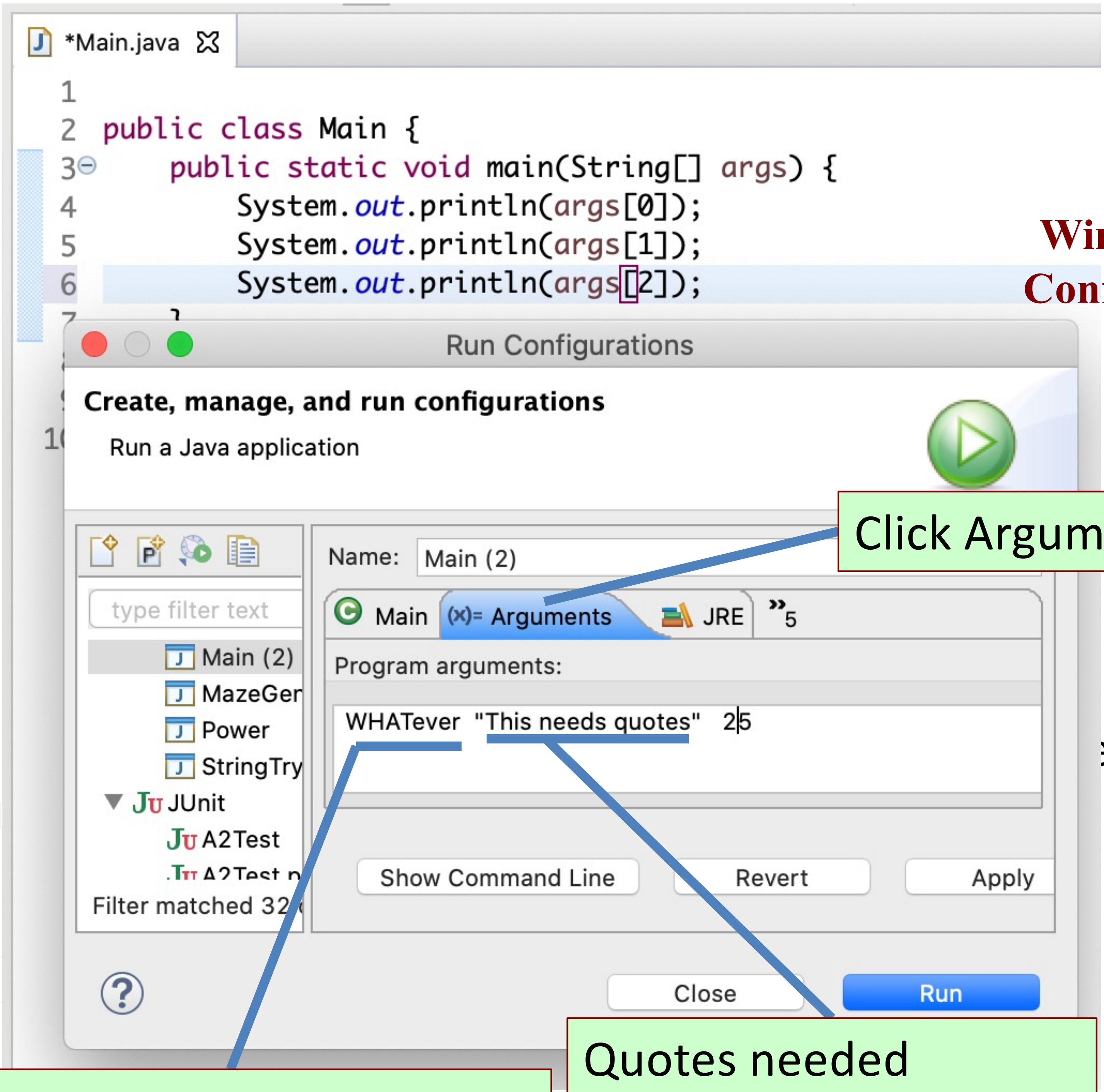
Run -> Run (or click the green Play button)

Eclipse executes the call main(array with 0 elements);

To tell Eclipse what array of Strings to give as the argument, start by using menu item

Run -> Run Configurations…

(see next slide)



```java
public class Main {
    public static void main(String[] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
        }
```

```
WHATever "This needs quotes"  25
```

Window Run Configurations

Click Arguments pane

Quotes OK, but not needed

Quotes needed because of space char

This ... es

argu

args

args

args

DEMO: Giving an argument to the call on main

Change the program to print the String that is in args[0], i.e. change the statement in the body to

```
System.out.println(args[0]);
```

Then

- Do Run -> Run Configurations
- Click the Arguments tab
- In the Program field, type in “Haloooo there!”
- Click the run button in the lower right to execute the call on main with an array of size 1 …

PACKAGES AND
THE JAVA API DOCUMENTATION

Package

Package: Collection of Java classes and other packages.
Type package into the JavaHyperText Filter field

Available here

www.cs.cornell.edu/courses/JavaAndDS/definitions.html

Three kinds of package

(1) The default package: in project directory/src

(2) Java classes that are contained in a specific directory on your hard drive (it may also contain sub-packages)

(3) Packages of Java classes that come with Java, e.g. packages java.lang, java.io

API packages that come with Java

Visit course webpage, click the link to version 11 on the homepage

Link:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/module-summary.html

Better yet, just google this and click the first link:

java 11 API

In left column, click java.base to get a list of packages.

Click package java.lang to get a list of classes that are fundamental to the Java language –are part of it.

Package java.lang vs. other packages

You can use any class in package java.lang. Just use the class name, e.g.

```
Character
```

To use classes in other API packages, you have to give the whole name, e.g.

```
javax.swing.JFrame     // (classes in package javax.swing  are
                       // are used to build GUIs ---Graphical
                       // User Interfaces)
```

So you have to write:

```
javax.swing.JFrame  jf=  new javax.swing.JFrame();
```

Use the import statement!

To be able to use just JFrame, put an import statement before the class definition:

```
import javax.swing.JFrame;

public class  C {
    …
    public void m(…) {
        JFrame  jf=  new JFrame();
        …
    }
}
```

Imports only class JFrame. Use the asterisk, as in line below, to import all classes in package:

```
import javax.swing.*;
```

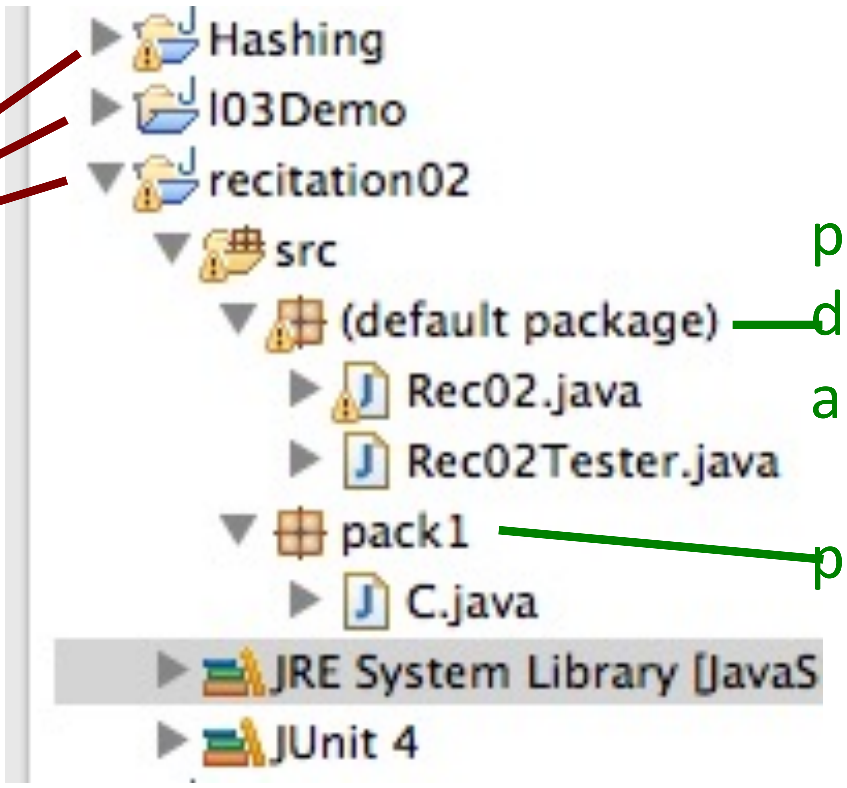
Don't be concerned with all the Java. For now, just think about the import statement.

Other packages on your hard drive

One can put a bunch of logically related classes into a package, which means they will all be in the same directory on hard drive. Reasons for doing this? We discuss much later.

Image of Eclipse Package Explorer:

3 projects:



project has default package and

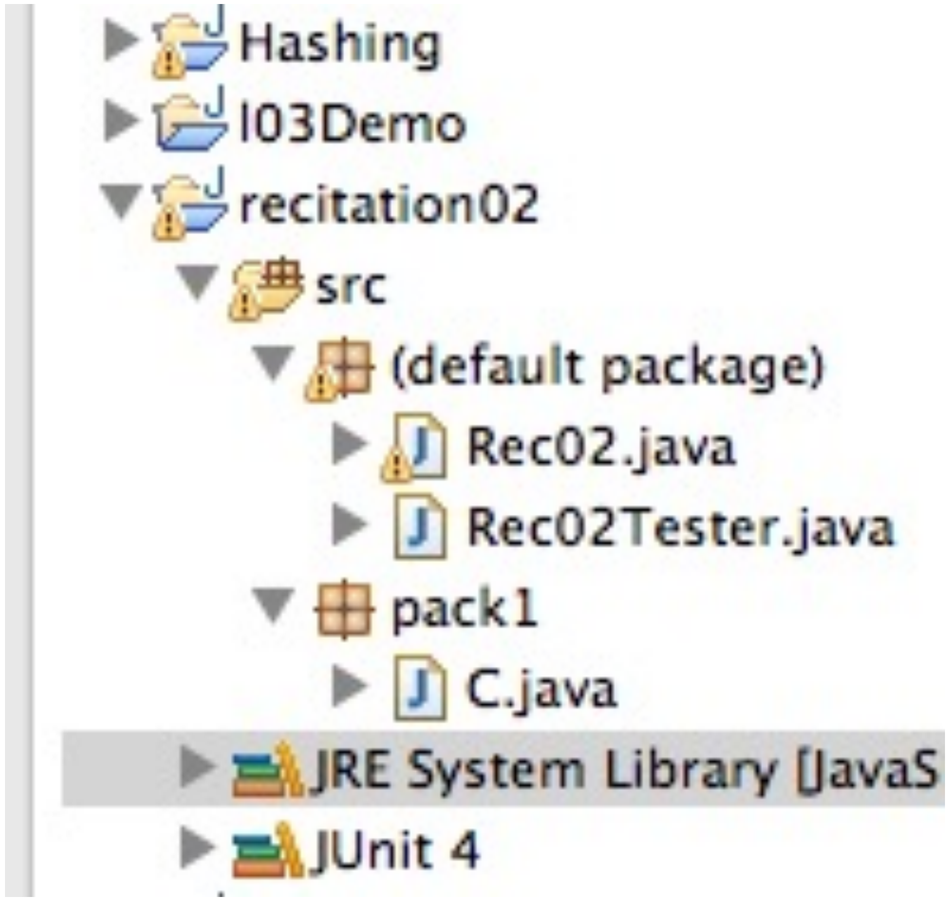
package pack1

Default package has 2 classes: Rec02, Rec02Tester

pack1 has 1 class: C

Hard drive Eclipse Package Explorer

```
Eclipse
    Hashing
    I03Demo
    recitation02
        src
            Rec02.java
            Rec02Tester.java
            pack1
                C.java
```



Eclipse does not make a directory for the default package; its classes go right in directory src

Importing the package

Every class in package pack1 must start with the package statement

```
package pack1;

import javax.swing.*;

public class MyFrame
              extends JFrame {
}
```

Every class outside the package should import its classes in order to use them

```
import pack1.*;

public class DemoPackage {

   public Rec02() {
      MyFrame  v= MyFrame();
       …
   }
}
```

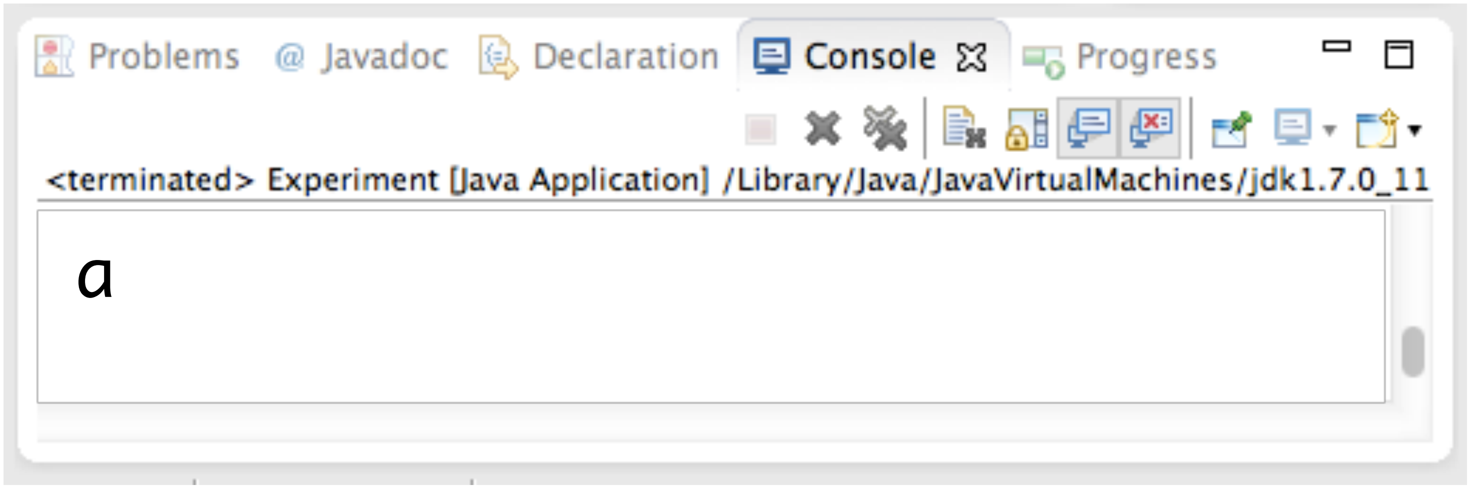
CHAR AND CHARACTER

Primitive type char

Use single quotes

```
char fred= 'a';
char wilma= 'b';
System.out.println(fred);
```

Unicode: 2-byte representation
Visit www.unicode.org/charts/
to see all unicode chars



```
<terminated> Experiment [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_11
a
```

Special `chars` worth knowing about

- `' '` - space
- `'\t'` - tab character
- `'\n'` - newline character
- `'\''` - single quote character
- `'\"'` - double quote character
- `'\\'` - backslash character
- `'\b'` - backspace character - NEVER USE THIS
- `'\f'` - formfeed character - NEVER USE THIS
- `'\r'` - carriage return - RARELY USE THIS

Backslash, called the escape character

Casting char values

Cast a char to an **int** using unary prefix operator (**int**),
Gives unicode representation of char, as an **int**

(**int**) `'a'` gives 97

(**char**) 97 gives `'a'`

(**char**) 2384 gives 'ॐ'

Om, or Aum, the sound of the universe (Hinduism)

No operations on **chars** (values of type char)! **BUT**, if used in a relation or in arithmetic, a **char** is automatically cast to type **int**.

Relations < > <= >= == != ==

```
'a' < 'b'        same as      97 < 98, i.e. true
'a' + 1          gives        98
```

How to check properties of a char c

```
Character.isAlphabetic(c)
Character.isDigit(c)
Character.isLetter(c)
Character.isLowerCase(c)
Character.isUpperCase(c)
Character.isWhitespace(c)

Character.toLowerCase(c)
Character.toUpperCase(c)
```

These return the obvious boolean value for parameter c, a **char**

Whitespace chars are the space ‘ ‘, tab char, line feed, carriage return, etc.

These return a char.

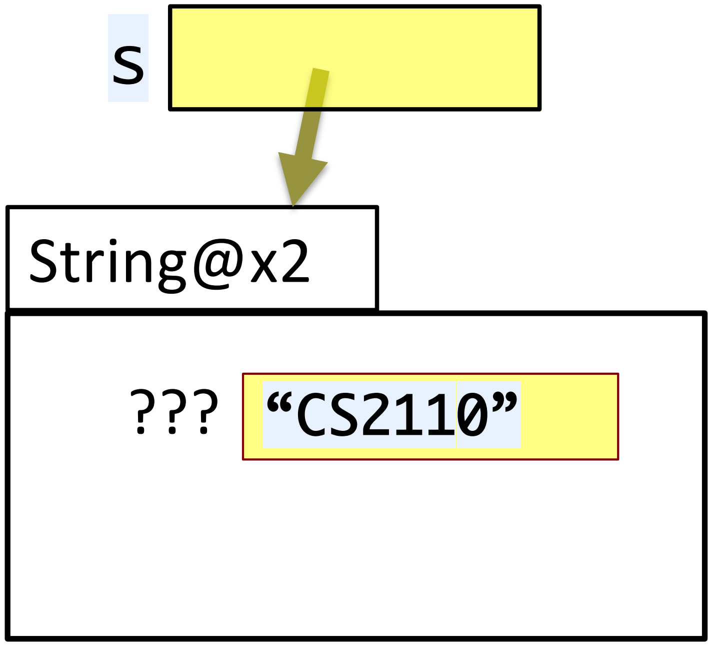
You will learn about Character. later

STRING

Class String

```
String s= “CS2110”;
```

String: special place in Java.
String literal creates object.



String is a class. A variable of type String contains a pointer to an object that contains the String

Important: String object is immutable: can’t change its value. All operations/functions create new String objects

Objects and classes were taught Tuesday, in lecture 2. We introduce Strings here only so you can learn about operation + and functions on this oft-used type.

Operator +

+ is overloaded

`"abc" + "12$"` evaluates to `"abc12$"`

If one operand of catenation is a String and the other isn't, the other is converted to a String.
Sequence of + done left to right

`1 + 2 + "ab$"` evaluates to `"3ab$"`

`"ab$" + 1 + 2` evaluates to `"ab$12"`

Watch out!

Operator +

```
System.out.println("c is: " + c +
                   ", d is: " + d +
                   ", e is: " + e);
```

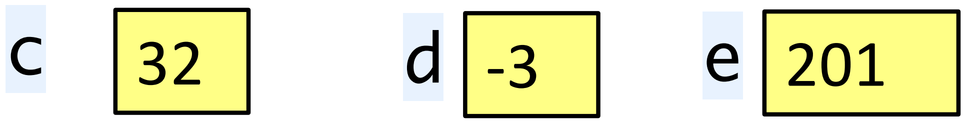
Using several lines increases readability

Can use + to advantage in println statement. Good debugging tool.

- Note how each output number is annotated to know what it is.

Output:

```
c is: 32, d is: -3, e is: 201
```



Picking out pieces of a String

s.length(): number of chars in s — 5

01234 Numbering chars: first one in position 0

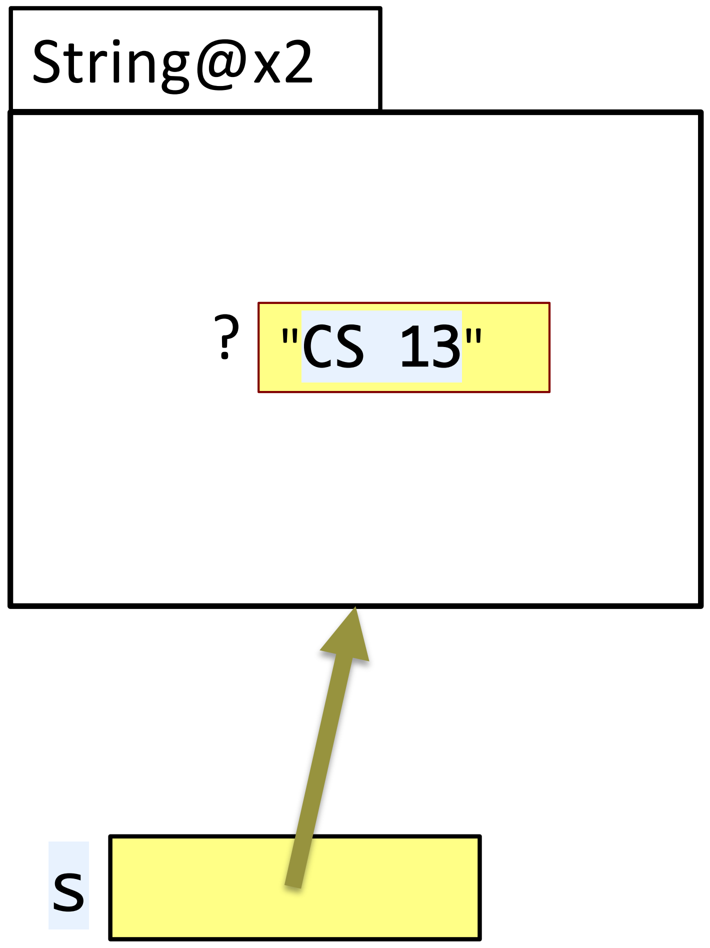
"CS 13"

s.charAt(i): char at position i

s.substring(i): new String containing chars at positions from i to end — s.substring(2) is ' 13'

s.substring(i, j): new String containing chars at positions i..(j-1) — s.substring(2,4) is ' 1'

Be careful: Char at j not included!



Other useful String functions

s.trim() – s but with leading/trailing whitespace removed

s.indexOf(s1) – position of first occurrence of s1 in s (-1 if none)

s.lastIndexOf(s1) – similar to s.indexOf(s1)

s.contains(s1) – true iff String s1 is contained in s2

s.startsWith(s1) – true iff s starts with String s1

s.endsWith(s1) – true iff s ends with String s1

s.compareTo(s1) – 0 if s and s1 contain the same string,
< 0 if s is less (dictionary order),
> 0 if s is greater (dictionary order)

There are more functions! Look at the API specs!