CS2110, Recitation 1

Arguments to method main,
Packages,
Wrapper Classes,
Characters,
Strings

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:

- Eclipse: File -> New -> Project
- File -> New -> Class
- Check the method main box
- In the class that is created, write the above statement in the body of main
- Hit the green play button or do menu item Run -> Run

Java Application

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.

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 arguments);

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

Run -> Run Configurations…

(see next slide)

Window Run Configurations

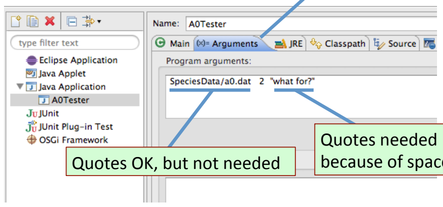
This Arguments pane of Run Configurations window gives argument array of size 3:

args[0]: "SpeciesData/a0.dat"

args[1]: "2"

args[2]: "what for?"

Click Arguments pane



Name: A0Tester

Program arguments:

```
SpeciesData/a0.dat  2  "what for?"
```

Quotes OK, but not needed

Quotes needed because of space char

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

Package

Package: Collection of Java classes and other packages.
See JavaSummary.pptx, slide 20
Available in the course website in the following location:
http://www.cs.cornell.edu/courses/CS2110/2017sp/links.html

Three kinds of packages

(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, javax.swing.

API packages that come with Java

Visit course webpage, click Links, then Java 8 API Specs.
Link:
http://www.cs.cornell.edu/courses/CS2110/2017sp/links.html

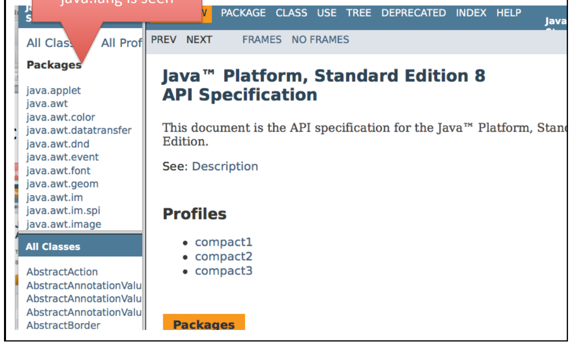
Better yet, just google something like:

java 8 API

Scroll down in left col (Packages pane), click on java.lang

ckage documentation

Scroll down until java.lang is seen


PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV NEXT FRAMES NO FRAMES
All Clas
All Prof
Packages
All Classes
Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image

AbstractAction
AbstractAnnotationValu
AbstractAnnotationValu
AbstractAnnotationValu
AbstractBorder

Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Stanc Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

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

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.*;
```

Other packages on your hard drive

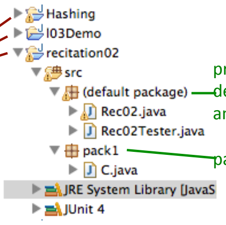
One can put a bunch of logically related classes into a package, which means they will all be in the same directory on hard drive. Reasons for doing this? We discuss much later.

Image of Eclipse Package Explorer:

3 projects:

Default package has 2 classes: Rec02, Rec02Tester

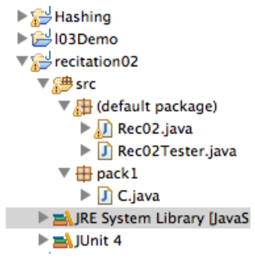
pack1 has 1 class: C



project has default package and package pack1

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();
      …
   }
}
```

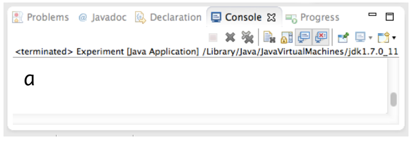
CHAR AND CHARACTER

Primitive type char

Use single quotes

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

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



```
<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 - NEVER 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

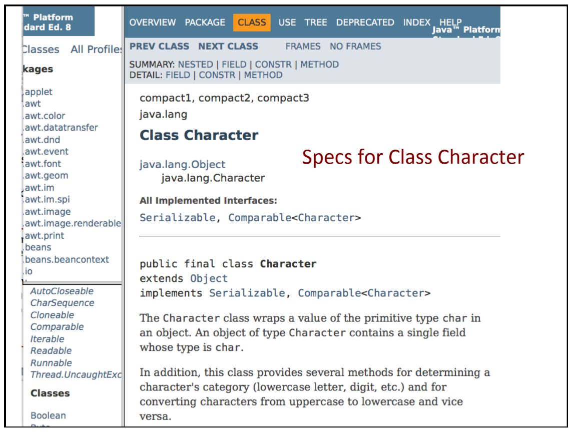
(**char**) 97 gives 'a'

(**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. false
'a' + 1       gives       98
```



Specs for Class Character

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

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

compact1, compact2, compact3
java.lang

Class Character

java.lang.Object
    java.lang.Character

All Implemented Interfaces:

Serializable, Comparable<Character>

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

The Character class wraps a value of the primitive type char in an object. An object of type Character contains a single field whose type is char.

In addition, this class provides several methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.

Specs for Class Character

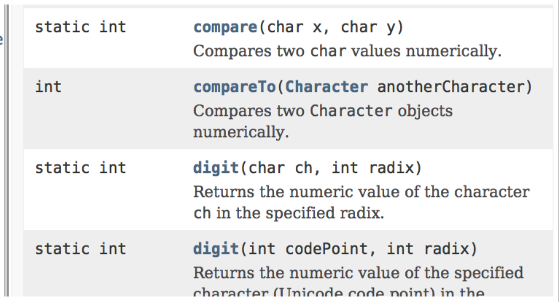
Main pane now contains description of class Character:

1. The header of its declaration.
2. A description, including info about Unicode
3. Nested class summary (skip it)
4. Field summary (skip it)
5. Constructor summary (read)
6. Method summary (read)
7. Field detail (skip it)
8. Method detail (read)

Find method compareTo
See a 1-sentence description

Click on method name
Takes you to a complete description in Method detail section

Character: summary, function compareTo

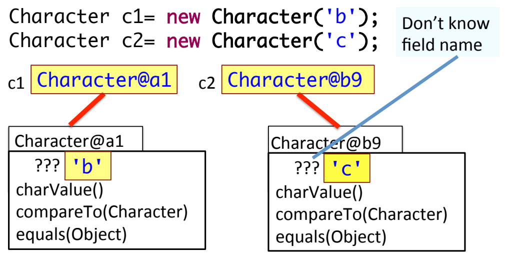


static int	**compare**(char x, char y) Compares two char values numerically.
int	**compareTo**(Character anotherCharacter) Compares two Character objects numerically.
static int	**digit**(char ch, int radix) Returns the numeric value of the character ch in the specified radix.
static int	**digit**(int codePoint, int radix) Returns the numeric value of the specified character (Unicode code point) in the

Class Character

An object of class Character **wraps** a single **char**
(has a field that contains a single **char**)

```
Character c1= new Character('b');
Character c2= new Character('c');
```



Class Character

- Each instance of class Character wraps a **char** value —has a field that contains a **char** value. Character allows a **char** value to be treated as an object.
- Find methods in each object by looking at API specs on web:
  docs.oracle.com/javase/8/docs/api/java/lang/Character.html

c.charValue()	c's wrapped char, as a **char**
c.equals(c1)	True iff c1 is a Character and wraps same char
c.compareTo(c1)	0 if c == c1. < 0 if c < c1. > 0 if c > c1.
c.toString()	c's wrapped char, as a String
…	…

Static methods in class Character

Lots of static functions. You have to look to see what is available. Below are examples

```
isAlphabetic(c)
isDigit(c)
isLetter(c)
isLowerCase(c)
isUpperCase(c)
isWhitespace(c)
toLowerCase(c)
toUpperCase(c)
```

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

We'll explain "static" soon

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

These return a char.

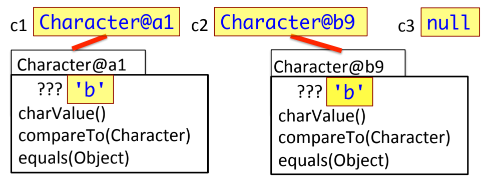
You can import these using "import static java.lang.Character.*;"

== versus equals

```
c1 == c2          false
c3 == c1          false
c1 == c1          true
c1.equals(c2)     true
c3.equals(c1)     Error!!!
```

true iff c1, c2 contain same values

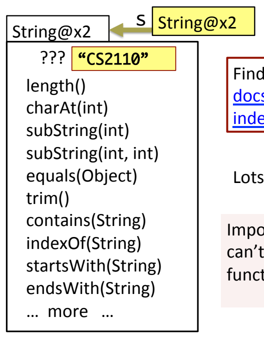
true iff c2 is also a Character object and contains same char as c1



STRING

Class String

```
String s= "CS2110";
```



String: special place in Java: no need for a new-expression. String literal creates object.

Find out about methods of class String: docs.oracle.com/javase/8/docs/api/index.html?java/lang/String.html

Lots of methods. We explain basic ones

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

Operator +

+ is overloaded

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

If one operand of concatenation 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$1`

Watch out!

Operator +

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

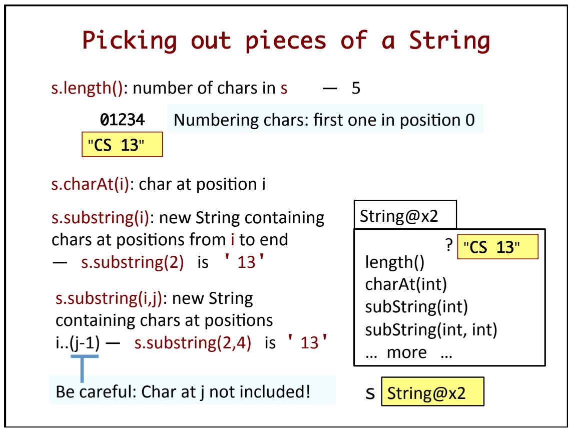
Using several lines increases readability

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

c 32   d -3   e 201



Picking out pieces of a String

s.length(): number of chars in s — 5

```
  01234
 "CS 13"
```

Numbering chars: first one in position 0

s.charAt(i): char at position i

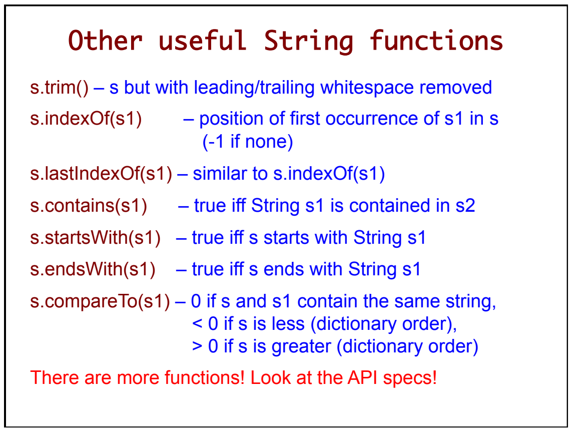
s.substring(i): new String containing chars at positions from i to end — s.substring(2) is ' 13'

s.substring(i,j): new String containing chars at positions i..(j-1) — s.substring(2,4) is ' 13'

Be careful: Char at j not included!

```
String@x2
          ? "CS 13"
length()
charAt(int)
subString(int)
subString(int, int)
… more …
```

s String@x2



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!