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
- Make Package field empty!!!
  - Give it a name
  - 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) {...}
```

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) {...}
```

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…

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?"

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 "Hallooo 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**

**API packages that come with Java**
Visit course webpage, click Links, then Java 8 API Specs.
Link: [www.cs.cornell.edu/courses/CS2110/2017fa/links.html](http://www.cs.cornell.edu/courses/CS2110/2017fa/links.html)

Better yet, just google something like:

```java
java 8 API
```

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

**Package java.lang vs. other packages**
You can use any class in package `java.lang`. Just use the class name, e.g.

```java
Character
```

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

```java
javax.swing.JFrame
```

So you have to write:

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

**Package**

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

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

```java
import javax.swing.JFrame;
```

```java
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:

```java
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

Hard drive
- Eclipse
  - Hashing
  - I03Demo
  - recitation02
- src
  - Rec02.java
  - Rec02Tester.java
  - pack1
  - C.java

Eclipse Package Explorer

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

```java
package pack1;
import javax.swing.*;
public class DemoPackage {
  public Rec02() {
    MyFrame v = MyFrame();
    ...}
}
```

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

```java
import pack1.*;
pubic class MyFrame extends JFrame { }
```

CHAR AND CHARACTER

Primitive type char

Use single quotes

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

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

Special chars worth knowing about

- Space
- Tab character
- Newline character
- Single quote character
- Double quote character
- Backslash character
- Backspace character - NEVER USE THIS
- Formfeed character - NEVER USE THIS
- 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'

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

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)

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');

Character@c1 Character@c2

charValue() compareTo(Character)
equals(Object)

Don't know field name

Character@b Character@b

charValue() compareTo(Character)
equals(Object)

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)`
- `isLowerCase(c)`
- `isLetter(c)`
- `isUpperCase(c)`
- `isWhitespace(c)`
- `toLowerCase(c)`
- `toUpperCase(c)`

These return boolean. 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  true iff c1, c2 contain same values

c3 == c1  false

c1 == c1  true

c1.equals(c2)  true  true iff c2 is a Character object and contains same char as c1

c1.equals(c1)  true

c3.equals(c1)  Error!!!

Class String

String s = “CS2110”;

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 +

“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$2”

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

<table>
<thead>
<tr>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>-3</td>
<td>201</td>
</tr>
</tbody>
</table>
Picking out pieces of a String

- `s.length()`: number of chars in `s` — 5

<table>
<thead>
<tr>
<th>01234</th>
<th>Numbers chars: first one in position 0</th>
</tr>
</thead>
</table>

- `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` to `j-1` — `s.substring(2, 4)` is "13"

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!