CS2110, Recitation 2

Arguments to method main
Packages
APIspecs
Characters
Strings

Demo: Create application

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

```java
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 blue statement in the body of main
- Hit the green play button or do menu item Run -> Run

Java Application

```java
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

```java
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 gives argument array of size 3:
args[0]: "SpeciesData/a0.dat"
args[1]: "2"
args[2]: "what for?"

Click Arguments pane

Quotes needed because of space char

Quotes OK, but not needed

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

```java
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: 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/CS2110/2016sp/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.

Every class in package
pack1 (in directory
pack1) must start with
the package statement

package pack1;
public class C {
    ...
}

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

import pack1.*;
public class Rec02 {
    ...
}

Finding package documentation

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();

Importing the package

Importing API packages

You can do this:

public class C {
    javax.swing.JFrame jf= new javax.swing.JFrame();
    ...
}

Or this:

import javax.swing.JFrame;
public class C {
    JFrame jf= new JFrame();
    ...
}

To import all classes in
package javax.swing, use
import 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;

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

Imports only 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.

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/](http://www.unicode.org/charts/) to see all unicode chars

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

\[(\text{int}) 'a' \text{ gives } 97\]
\[(\text{char}) 97 \text{ gives 'a'}\]
\[(\text{char}) 2384 \text{ gives 'ॐ'}\]

No operations on \texttt{chars} (values of type char)! BUT, if
used in a relation or in arithmetic, a \texttt{char} is automatically cast to
\texttt{type int}.
Relations \(<\ >\ <=\ >=\ ==\ !=\ ==\Provides

'\texttt{a} < \texttt{b}' \text{ same as } 97 < 98, \text{i.e. false}
'\texttt{a} + 1 \text{ gives } 98

== versus equals

c1 == c2 \text{ false}
c3 == c1 \text{ false}
c1 == true
\texttt{c1.equals(c2)} \text{ true}
\text{true iff c2 is also a Character object and contains same char as c1}
c3.equals(c1) \text{ Error!!!}

\texttt{Character@c1} \text{ ????}
\texttt{charValue()}
\texttt{compareTo(Character)}
\texttt{equals(Object)}

\texttt{Character@c2} \text{ ????}
\texttt{charValue()}
\texttt{compareTo(Character)}
\texttt{equals(Object)}

\texttt{null}

\texttt{Character@c3}

Class Character

An object of class Character \texttt{wraps} a single char
(has a field that contains a single char)

\texttt{Character c1= new Character(’b’);} \texttt{;}
\texttt{Character c2= new Character(’c’);} \texttt{;}
\texttt{c1 \texttt{Character@a1}}
\texttt{c2 \texttt{Character@b9}}

\texttt{charValue()}
\texttt{compareTo(Character)}
\texttt{equals(Object)}

\texttt{Character@c1}
\texttt{Character@b9}

\texttt{null}

STRING

Class String

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

\texttt{String s= “CS2110”;}
\texttt{String@x2}

\texttt{String@x2}
\texttt{s}

length()
\texttt{charAt(int)}
\texttt{subString(int)}
\texttt{indexOf(String)}
\texttt{contains(String)}
\texttt{startsWith(String)}
\texttt{endsWith(String)}
\texttt{trim()}
\texttt{equals(Object)}
\texttt{equals(String)}
\texttt{indexOf(String)}
\texttt{substring(int, int)}
\texttt{... more...}

Find out about methods of class String:
\texttt{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\texttt{t change its value. All operations/}
funtions create new String objects

Operator +

“\texttt{abc}” + “\texttt{12$}” evaluates to “\texttt{abc12$}”

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 + “\texttt{ab$}” evaluates to “\texttt{3ab$}”

“\texttt{ab$}” + 1 + 2 evaluates to “\texttt{ab$}”
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

"CS 13"

Numbering chars: first one in position 0

s.charAt(): char at position i

s.substring(): 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!