

Packages, Characters, Strings Arguments to method main

CS2110, Week 2 Recitation

1

Package

Package: A 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/CS2110/2014sp/links.html>

- (1) Java classes that are contained in a specific directory on your hard drive (it may also contain sub-packages) or
- (2) Packages of Java classes that come with Java, e.g. packages `java.lang`, `javax.swing`.

Consider first the packages that come with Java. We show you:

- (1) How to refer to them
- (2) How to find out how to use them, using the API (Application Programmer Interface) specifications.

2

API packages that come with Java

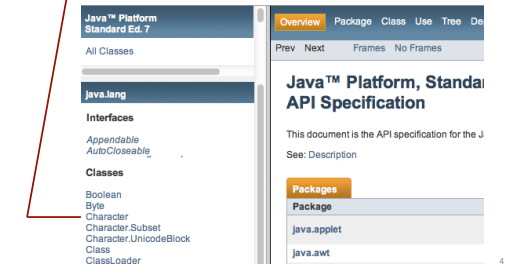
Visit course webpage, click [Links](#), then [Java 1.7 API Specs](#).
Link:
<http://www.cs.cornell.edu/courses/CS2110/2014sp/links.html>
Scroll down in left col (Packages pane), click on `java.lang`



3

API packages that come with Java

On page that opens, left col, you see Classes in package `java.lang`.
Click on `Character`



4

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

More on class `Character` later

5

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();
```

6

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.*;
```

7

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.

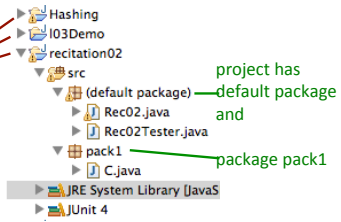
Image of Eclipse Package Explorer:

3 projects:

Default package has 2 classes:

Rec02, Rec02Tester

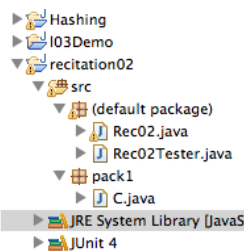
pack1 has 1 class: C



8

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`

9

Importing the package

Every class in package `pack1` must start with the package statement

```
package pack1;

public class C {

    /** Constructor: */
    public C() {
    }

}
```

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

```
import pack1.*;

public class Rec02 {

    public Rec02() {
        C v= new C();
    }

}
```

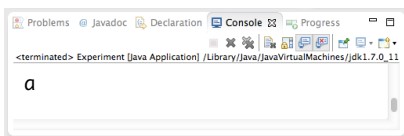
10

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



11

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

12

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
```

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');
```

Don't know field name

Class Character

- Each instance of class **Character** wraps an **int** value —has a field that contains an **int** value. 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/7/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.

== 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 also a Character object and contains same char as c1
c3.equals(c1) Error!!!
```

Class String

```
String s= "CS211";
```

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

```
String@x2 ← s String@x2
??? "CS211"
length()
charAt(int)
substring(int)
substring(int, int)
equals(Object)
trim()
contains(String)
indexOf(String)
startsWith(String)
endsWith(String)
... more ...
```

Find out about methods of class **String**: docs.oracle.com/javase/7/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

Catenation, or concatenation
 ↓
 "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"

19

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

20

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 of s — s.charAt(3) is '1'

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'

String@x2 ? "CS 13"

length()
 charAt(int)
 substring(int, int)
 ... more ...

Be careful: Char at j not included! s String@x2

21

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!

22

Giving method main an argument

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

Parameter: String array

In Eclipse, when you do menu item Run -> Run or Run -> Debug

Eclipse calls method main. Default is main(null);

To tell Eclipse what array of Strings to give as the argument, Use menu item Run -> Run Configurations...

or Run -> Debug Configuration...

(see next slide)

23

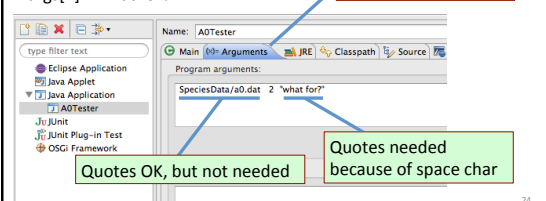
Window Run Configurations

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?"



24