Object-Oriented Programming

Ewan Klein

Inf1 :: 2008/09

Overview

- Link to D & A Course: simple processing of tabular data.
- Some useful odds and ends not covered yet in HFJ.

A Familiar Table

[]1blue!20blue!10

MN	Name	Age	MBox	Gend
s0189034	Peter	18	peter@math	M
s0289125	Michael	21	mike@geo	M
s0378435	Helen	20	helen@phys	F
s0412375	Mary	18	mary@inf	F
s0456782	John	21	john@inf	M

- How can we represent and manipulate data like this in Java?
- We look at two approaches.

Two Dimensional Arrays, 1

Array of Arrays

```
String[][] students = new String[5][];
```

- `students` is an array of length 5
- Each element of `students` is itself an array, of type `String[]`.

Two Dimensional Arrays, 2

Directly assign values to 5 new `String[]` arrays:

Initialize the Elements

```
String[] peter   = "s0189034", "Peter", "18", "peter@math", "M" ;
String[] michael = "s0289125", "Michael", "21", "mike@geo" , "M" ;
String[] helen   = "s0378435", "Helen", "20", "helen@phys" , "F" ;
String[] mary    = "s0412375", "Mary", "18", "mary@inf" , "F" ;
String[] john    = "s0456782", "John", "21", "john@inf" , "M" ;
```


Two Dimensional Arrays, 3

Now make these be the rows in `students`:

Assigning values to the rows

```
students[0] = peter;
students[1] = michael;
students[2] = helen;
students[3] = mary;
students[4] = john;
```


Two Dimensional Arrays, 4

`students[i][j]` selects the i^{th} row and the j^{th} column of the table.
Schematically:

Indexing into the 2D array

```
[0][0], [0][1], [0][2], [0][3], [0][4]
[1][0], [1][1], [1][2], [1][3], [1][4]
[2][0], [2][1], [2][2], [2][3], [2][4]
[3][0], [3][1], [3][2], [3][3], [3][4]
[4][0], [4][1], [4][2], [4][3], [4][4]
```

- Michael's Mbox?
- Mary's Name?

Placing Conditions on the Data, 1

Task: print out only the names of students aged more than 18.

First Attempt

```
// Select every row in the table
for (String[] row : students) {
    if (row[2] > 18) {
        System.out.println(row[1]);
    }
}
```

Doesn't work! (Why not?)

Placing Conditions on the Data, 2

Convert a `String` to an `int`:

Second Attempt

```
for (String[] row : students) {
    if ( Integer.parseInt(row[2]) > 18) {
        System.out.println(row[1]);
     }
}
```

`parseInt()` is a method of the `Integer` class.

Output

```
Michael
Helen
John
```


Placing Conditions on the Data, 3

Combine two conditions using && (AND):

AND Example

```
for (String[] row : students) {
   if (Integer.parseInt(row[2]) > 18 && row[4].equals("F")) {
        System.out.println(row[1]);
     }
}
```

Output

```
Helen
```


Using an Object to Hold Data, 1

The Student Class

```
public class Student {
    private String mn;
    private String name;
    private int age; // NB Not a String
    private String mbox;
    private String gender;
    ...
}
```

Plus setters and getters for these fields.

Using an Object to Hold Data, 2

Initialize Student Object

```
Student peter = new Student();
peter.setMn("s0189034");
peter.setName("Peter");
peter.setAge(18);
peter.setMbox("peter@math");
peter.setGender("M");
```

Similarly for four other `Student` objects.

Using an Object to Hold Data, 3

Create an ArrayList

```
ArrayList<Student> students = new ArrayList<Student>();
students.add(peter);
students.add(michael);
students.add(helen);
students.add(mary);
students.add(john);
```

Why is `ArrayList` a good idea here?

Which Approach?

- Object-based approach is, well, more Object-oriented.
- Relaxes constraints on types of data values.
- More self-documenting — easier to tell what the different data 'columns' mean.

Placing Conditions on the Data, 4

Combine two conditions using `||` (OR):

OR Example

```
for (Student s : students) {
    if (s.getAge() > 18 || s.getGender().equals("F")) {
        System.out.println(s.getName());
    }
}
```

Output

```
Helen
Mary
John
```

Placing Conditions on the Data, 5

Negate a condition using ! (NOT):

NOT Example

```
for (Student s : students) {
    if ( ! (s.getAge() > 18)) {
        System.out.println(s.getName());
    }
}
```

Output

```
Peter
Michael
Mary
```


A Closer Look at Logical Operators

	C_1 && C_2	true if both C_1 and C_2 are true
[]1blue!10blue!05	C_1 || C_2	true if either or both of C_1, C_2 are true
	! C	true if C is false

Short-Circuit Evaluation

&&	Stop if lefthand condition is `false` — we're bound to fail.
||	Stop if lefthand condition is `true` — we're bound to succeed.

Suppose C_1 is true of 90% of our data items and C_2 is true of 20%:

- C_1 && C_2?
- C_2 && C_1?

Format Strings

How to gain more finegrained control over print strings.

`println` can be Clunky

The student named 'Peter' is aged 18.

Using string concatenation

```
System.out.println("The student named '"
                   + peter.getName()
                   + "' is aged "
                   + peter.getAge()
                   + ".");
```

String with Format Specifiers, 1

Target String

```
"The student named 'Peter' is aged 18."
```

String with Gaps

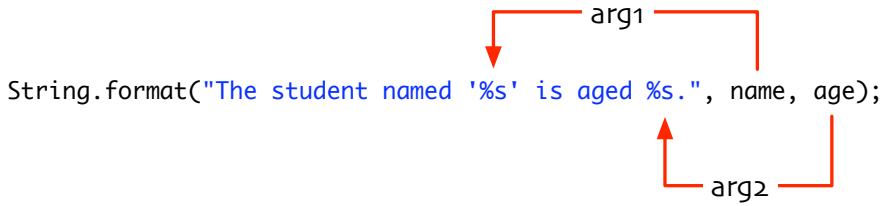
```
"The student named '_' is aged _."
```

String with Format Specifiers

```
"The student named '%s' is aged %s."
```

- %s is a placeholder for a string.
- Called a format specifier.
- Each format specifier in a string gets replaced by an actual value.

String with Format Specifiers, 2



String with Format Specifiers, 3

Define a Format String

```
String str =
    String.format("The student named '%s' is aged %s.",
                  peter.getName(), peter.getAge());
System.out.println(str);
```

Output

The student named 'Peter' is aged 18.

printf, 1

Shorter version

```
System.out.printf("The student named '%s' is aged %s.",
                  peter.getName(), peter.getAge());
```

Output

The student named 'Peter' is aged 18.

printf, 2

Convert char to String

```
System.out.printf("'%s' is for Apple.", 'A');
```

Output

'A' is for Apple.

printf, 2

Round to 2 decimal places

```
System.out.printf("The average is %.2f.", 3.34509);
```

Output

The average is 3.35.

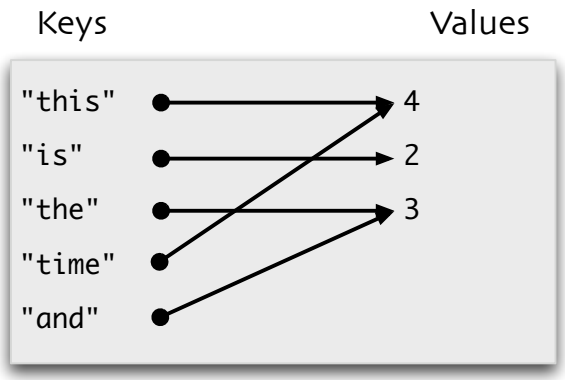
Maps

Maps

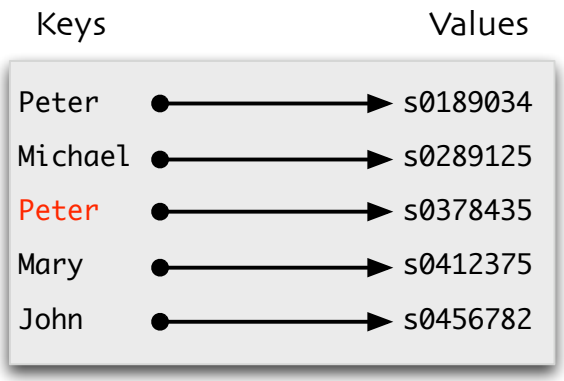
- Associate keys with values.
  1. Given a name, look up a telephone number.
  2. Given a domain, look up an IP address.
  3. Given a location look up GPS coordinates.
  4. Given a word, look up its frequency in a text.

Map People to their Matric Nos.

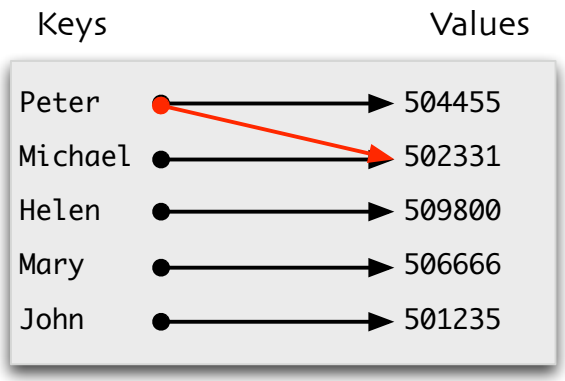
Map Words to Length



Map People to their Matric Nos: Wrong!



Map People to their Telephone Nos: Wrong!



HashMap, 1

Import HashMap

```
import java.util.HashMap;
```


HashMap, 2

Declare HashMap

```
HashMap<String, Integer> map = new HashMap<String, Integer>();
```


Digression: Get Some Words

split() method of String

```
String sent = "I stand here today humbled by the task be-
fore us";
String[] words = sent.split(" "); // split on whitespace
```


HashMap, 3

put a value for a key

```
for (String word : words) {
    map.put(word, word.length());
    }
```

`put`(*Value*, *Key*): put *Value* as the value of *Key* in `map`.

HashMap, 4

get a value for a key

```
for (String key : map.keySet()) {
    System.out.printf("%s => %s\n", key, map.get(key));
    }
```

- `map.keySet()`: get the set of keys in `map`.
- `get(`*Key*`)`: get the value of *Key* in `map`.
- `\n`: a print 'escape sequence' for specifying a newline.

Looking at the Keys and Values

Output

```
the => 3
today => 5
I => 1
stand => 5
us => 2
before => 6
by => 2
humbled => 7
task => 4
here => 4
```

NB We don't have control in the order in which keys are printed out.

Summary

- Lots of new stuff today!
  - Using two-dimensional arrays for tabular data.
  - Using objects and instance variables for tabular data.
- `Integer.parseInt`
- Logical operators `&&`, `||`, `!`
- `String.format`, `split()`, `printf()`.
  - There's a short section on format strings in HFJ, p. 294–300
- `HashMap`s for associating keys with values.
- For more on `HashMap` (and `Map`), look at the Java API:

DICE:
`file:///usr/share/doc/java-1.6.0-sun-manual-1.6.0/api/index.html`

Sun's website:
`http://java.sun.com/javase/6/docs/api/`