Object-Oriented Programming
Handling Data and Print Formatting

Ewan Klein

Inf1 :: 2008/09
1 Tabular Data
   • 2D Arrays
   • Objects
   • Logical Operators

2 Format Strings

3 Maps
Overview

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

<table>
<thead>
<tr>
<th>MN</th>
<th>Name</th>
<th>Age</th>
<th>MBox</th>
<th>Gend</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0189034</td>
<td>Peter</td>
<td>18</td>
<td>peter@math</td>
<td>M</td>
</tr>
<tr>
<td>s0289125</td>
<td>Michael</td>
<td>21</td>
<td>mike@geo</td>
<td>M</td>
</tr>
<tr>
<td>s0378435</td>
<td>Helen</td>
<td>20</td>
<td>helen@phys</td>
<td>F</td>
</tr>
<tr>
<td>s0412375</td>
<td>Mary</td>
<td>18</td>
<td>mary@inf</td>
<td>F</td>
</tr>
<tr>
<td>s0456782</td>
<td>John</td>
<td>21</td>
<td>john@inf</td>
<td>M</td>
</tr>
</tbody>
</table>

- 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[].
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";
```
Now make these be the rows in students:

Assigning values to the rows

```java
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:

|-----|-----|-----|-----|-----|

- 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

```java
// 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**

```java
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
Ewan
Klein
```
Convert a String to an int:

Second Attempt

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

`parseInt()` is a method of the Integer class.
Placing Conditions on the Data, 2

Convert a String to an int:

Second Attempt

```java
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
Combine two conditions using `&&` (AND):

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

```java
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

```java
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.
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.
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?
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.
Combine two conditions using `||` (OR):

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

Output:

- Helen
- Mary
- John

Ewan Klein

Object-Oriented Programming Handling Data and Print Formatting
Combine two conditions using `||` (OR):

```java
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):

```java
NOT Example

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

Output:

```
Peter
Michael
Mary
Ewan Klein
```
Placing Conditions on the Data, 5

Negate a condition using ! (NOT):

```java
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 |
| \( C_1 \mid\mid C_2 \) | true if either or both of \( C_1 \), \( C_2 \) are true |
| \( \neg C \) | 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%:

\( C_1 \&\& C_2 \) vs. \( C_2 \&\& C_1 \)
A Closer Look at Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 &amp;&amp; C_2$</td>
<td>true if both $C_1$ and $C_2$ are true</td>
</tr>
<tr>
<td>$C_1 \mid \mid C_2$</td>
<td>true if either or both of $C_1$, $C_2$ are true</td>
</tr>
<tr>
<td>! $C$</td>
<td>true if $C$ is false</td>
</tr>
</tbody>
</table>

**Short-Circuit Evaluation**

- $\&\&$ Stop if lefthand condition is *false* — we’re bound to fail.
- $\mid \mid$ Stop if lefthand condition is *true* — we’re bound to succeed.
A Closer Look at Logical Operators

$C_1 \&\& C_2$  true if both $C_1$ and $C_2$ are true

$C_1 \mid\mid C_2$  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.

$\mid\mid$  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

```java
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."

%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, 1

Target String
"The student named 'Peter' is aged 18."

String with Gaps
"The student named '_' is aged _."
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."
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

```java
String.format("The student named '%s' is aged %s.", name, age);
```

ar1

arg2
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**

```java
System.out.\textbf{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.
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.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>s0189034</td>
</tr>
<tr>
<td>Michael</td>
<td>s0289125</td>
</tr>
<tr>
<td>Helen</td>
<td>s0378435</td>
</tr>
<tr>
<td>Mary</td>
<td>s0412375</td>
</tr>
<tr>
<td>John</td>
<td>s0456782</td>
</tr>
</tbody>
</table>
Map Words to Length

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;this&quot;</td>
<td>4</td>
</tr>
<tr>
<td>&quot;is&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;the&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;time&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;and&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Map People to their Matric Nos: **Wrong!**

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>s0189034</td>
</tr>
<tr>
<td>Michael</td>
<td>s0289125</td>
</tr>
<tr>
<td>Peter</td>
<td>s0378435</td>
</tr>
<tr>
<td>Mary</td>
<td>s0412375</td>
</tr>
<tr>
<td>John</td>
<td>s0456782</td>
</tr>
</tbody>
</table>
**Map People to their Telephone Nos: Wrong!**

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>504455</td>
</tr>
<tr>
<td>Michael</td>
<td>502331</td>
</tr>
<tr>
<td>Helen</td>
<td>509800</td>
</tr>
<tr>
<td>Mary</td>
<td>506666</td>
</tr>
<tr>
<td>John</td>
<td>501235</td>
</tr>
</tbody>
</table>

### Keys vs. Values

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>504455</td>
</tr>
<tr>
<td>Michael</td>
<td>502331</td>
</tr>
<tr>
<td>Helen</td>
<td>509800</td>
</tr>
<tr>
<td>Mary</td>
<td>506666</td>
</tr>
<tr>
<td>John</td>
<td>501235</td>
</tr>
</tbody>
</table>

---

Ewan Klein  
Object-Oriented Programming Handling Data and Print Formatting
import java.util.HashMap;
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 before us";
String[] words = sent.split(" "); // split on whitespace
put a value for a key

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

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