Paul Anderson
dcspaul@ed.ac.uk
http://www.inf.ed.ac.uk/teaching/courses/ijp
A talking fox. A pig doing stand-up. A singing pile of poo. A robot reciting slam poetry. You’re going to see all these things and more all over your iMessages and social media feeds in the coming months. They’re called “Animoji” and they’re Apple’s frivolous, fun, and downright addictive way to show off the capabilities of the TrueDepth camera in the iPhone X.

By Matthew Field
6 November 2017 • 10:49AM

An update to iOS has irritated iPhone users trying to type the letter "i", with the software's autocorrect feature changing the letter to a capital "A" and a question mark symbol.

The bug with iOS 11.1 has affected dozens of users, many of whom reported it after receiving their £999 iPhone X on its release on Friday.
We are being recorded (hopefully) ....

But slides & sound only (no video)
This Talk

A few common issues with the first assignment ...
- But I won’t discuss this in detail
  - individual marks & feedback later this week
- Make sure that you understand the feedback
  - ask a demonstrator

A bit about HashMap and Collections
- A few people didn’t “get” this - and it is important

A brief overview of the second assignment...
- What’s involved & Something about JavaFx/Scenebuilder

Designing a representation
- This is one of the most important parts of the course
  - both for the assignment, and “real life” programming
  - People who rely on their experience with other languages often do badly on this
Assignment 1

In general ...
- Many people had good implementations which passed my tests

But there were a few common issues ...
Assignment 1

HashMaps
- A number of people didn’t “get” the HashMaps
- I will talk a bit about this

Dependencies
- Quite a few people submitted code with missing dependencies
- Property files, other classes
- Test your code on a clean system - e.g. when submitting assignment 2

Over-Complex code
- If the code looks too convoluted, there is probably an easier way
- Especially if you have lots of numbers and arrays
- Use the features of the Java language
- Don’t rely on your previous experience with other languages
Worksheet Questions

- Many solutions would simply not work
  - This needs more careful thought
  - Ask a demonstrator to explain if you do not understand this

Readability

- Too little attention to the readability of the code
  - No excuse for bad indention - Eclipse will fix this for you
- Poor Javadoc
  - Un-edited (i.e. “wrong”) template content
  - Wrong filenames/format
To create bigger applications reliably ...

- You need to think at a “higher level” of abstraction
- Java provides you with lots of components which you can use as “building blocks”
- And it provides you with the tools to create your own building blocks - either using your own detailed code, or combining other components
- Look at some of the solutions to the first assignment and see how little code is involved
- If you find yourself writing lots of detailed code, using lots of numbers, or lots of arrays, then you are probably missing out

The container classes are one example of this

- Particularly, the Maps
An Example

What do we want to do?

- Add a name/number pair
- Given a name, return the number
- Delete a pair (maybe?)
- ...

For now..

- Ignore the fact that we have duplicate names
- Notice that the lookup is one-way - we can’t (easily) look up the number to find the name
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEATSON C</td>
<td>1 Brunswick Terrace Edinburgh</td>
<td>(0131) 477 4440</td>
</tr>
<tr>
<td></td>
<td>C, 38 Greenbank Rd EH10</td>
<td>(0131) 447 1498</td>
</tr>
<tr>
<td>D, 11</td>
<td>Brixwold Bank Bonnyrigg</td>
<td>(0131) 660 5133</td>
</tr>
<tr>
<td>D, 11</td>
<td>Featherhall Crescent Sth Edinburgh</td>
<td>(0131) 476 6664</td>
</tr>
<tr>
<td>D, 11</td>
<td>Featherhall Cres South 12</td>
<td>(0131) 334 3425</td>
</tr>
<tr>
<td>K, 4</td>
<td>Park Ct Musselburgh</td>
<td>(0131) 665 4243</td>
</tr>
<tr>
<td>R, 80</td>
<td>Woodburn Gardens Dalkeith</td>
<td>(0131) 660 3662</td>
</tr>
<tr>
<td>W, 11</td>
<td>Bryce Road Currie</td>
<td>(0131) 449 7744</td>
</tr>
<tr>
<td>BEATTIE A</td>
<td>16 Auction Mart Haddington</td>
<td>(01620) 823 442</td>
</tr>
<tr>
<td>A, 5</td>
<td>Bonaly Terrace Edinburgh</td>
<td>(0131) 441 4123</td>
</tr>
<tr>
<td>A, 18</td>
<td>Laird Ter Bonnyrigg</td>
<td>(0131) 472 1871</td>
</tr>
<tr>
<td>A, 35</td>
<td>Rennie PI E Linton</td>
<td>(01620) 861 232</td>
</tr>
<tr>
<td>A, 5/89</td>
<td>Stenhouse St Wst EH11</td>
<td>(0131) 443 3020</td>
</tr>
<tr>
<td>A, 32</td>
<td>Teviot Grove PENICUIK</td>
<td>(01968) 381 021</td>
</tr>
</tbody>
</table>
If we don’t have objects, or collections, we might ...

```java
int count = 0;
string names[1000];
string numbers[1000];

To add a pair ...

int count = 0;
names[count] = theName;
numbers[count] = theNumber;
count = count + 1;

To search for a name ...

for i = 0 to count-1
    if name[i] == theName then theNumber = number[i]
```
This “linear search” will be very slow

- If we were looking up a name manually, we would use the fact that the names are ordered ...
- Start in the middle and decide whether the name we want comes before or after this, then repeat ...
- So, we could program this ("binary search"), but ....
- We would have to insert the items in the right order, and ..... 

We don’t need to write this ourselves!

- A Java “Map” encapsulates this “lookup” behaviour

```java
Map(<String>,<String>) directory = new Map;
directory.put(theName,theNumber);
theNumber = directory.get(theName);
```
Java Collections

Java provides several implementations of the Map interface
- Just like assignment1 provided different service implementations
- But they have different performance characteristics
  - memory? insert speed? search speed? delete speed?
- The TreeMap is similar to the binary search
- The HashMap is a different implementation
- They all provide the same basic interface, so can be exchanged
  - but some implementations provides extras (e.g. ordered iteration)

As well as Maps ...
- Lists, Queues, Sets, Bags

Arrays are a good fit for some things
- But don’t use them by default - they are not the best choice in many cases

https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
If you need a different behaviour ....

‣ Create your own container class
‣ Try to keep it “generic”
‣ Make use of the other containers wherever possible

An Exercise ...

‣ Define your own Directory class for the telephone directory
‣ Extend this with an extra method to lookup by number as well as by name
‣ Think about the different possible ways of doing this, and their relative advantages/disadvantages
‣ Extend this so that it can handle duplicate names
‣ what kind of an object should get(name) return now?

Notice ...

‣ The design choices that you make for the class depend heavily on how you intend to use it - we will discuss that later
Feel free to produce a creative interface & application

- But the basic model must be good to obtain a pass

Do ...

- Think carefully about the design before getting too deep in the code - this will make your life easier!
- Discuss your design with a demonstrator

Do not tackle the advanced part of the assignment ...

- Unless you are confident that you can maintain a good design
Good “large-scale” structure is essential

- Can different people work on different parts independently?
  - you won’t be able to write all the code yourself
- Can we change one part without affecting lots of others?
  - things will change!
- Can we isolate parts for testing individually?

An “object-oriented” approach is intended to help with this

- The code is structured into separate “classes”
- With clear “interfaces” which define their external behaviour
- Loose “coupling” is good …
  - changing one class does not require lots of changes elsewhere
- High “cohesion” is good …
  - classes have a single, well-focussed purpose

This is the most important thing to learn from this course

- You will need to demonstrate this in order to pass
Rooms Demo
What does a GUI builder do?

**Output: Layout the screen elements graphically**
- Manage the appearance: colour, fonts, ...
- Manage the layout: resizing behaviour, ...

**Input: Handle events**
- Mouse buttons - multiple clicks, ...
- Mouse movement - mouse entry & exit to elements, ...
- Object actions - button clicks, menu selections, window closure, ..
- External events - network

This requires an “event-driven” style of programming
- You provides methods to respond to the various events
- You do not control the overall flow of the program
- Executing long-running methods usually requires threads
JavaFx & SceneBuilder

1. SceneBuilder

2. Your code

3. CSS Style file

- FXML
- JavaFx Library

Application Objects
You application will be manipulating representations of real-world (or imagined) entities
- There will be many different ways of representing these
  - A good choice will make the code easy to write
  - A bad choice can make it very difficult

There probably won’t be a single “best” representation
- There will be trade-offs (speed/readability/flexibility/memory ..)
- It will depend on the operations that you need to perform
- It will change over time as the application develops
Int season = 2;

String season = “summer”;

enum Season { SPRING, SUMMER, AUTUMN, WINTER };

public class Season { … }

Which is “best”?
It depends what we want to do!
Operations: “next”

if (season.eq “summer”) { next=“autumn”; } 
else if (season.eq “autumn”) { next=“winter”; } 
else if (season.eq “winter”) { next=“spring”; } 
else if (season.eq “spring”) { next=“summer”; } 

next = (season%4)+1;

Which is “better”?
- Clearer?
- Shorter?
- Faster? (does it matter?)
- Less Memory? (does it matter?)

What about …
- An Enum representation?
print( season );

String names[] =
    { "spring", "summer", "autumn", "winter" };
print( names[season-1] );

Another exercise ...

- This example uses an array to avoid having a lot of conditionals to work out the name for the season. Could you do anything similar for the first example of the “next” operation on the previous slide?
- What would a “Previous” operation look like?
Using either strings or integers ...

- We can't change the representation without changing every place where it is used
- We can't add extra information to the representation (average temperature?)
- Invalid values are possible: season="rabbits", season=-32

Defining a class solves all of these problems

- You have to decide if it is worth the extra "overhead"

```java
public Season season = Spring();
next = season.Next();
season.print();
```
public class Season {
    public static Season Spring();
    public static Season Summer();
    public static Season Autumn();
    public static Season Winter();
    public Season next();
    public void print();
}

Notice ...

‣ The Season class may use either of the previous representations internally - but we can change this at any time without affecting the interface
‣ The static constructors create new objects. It is not possible to create invalid Season objects
The Season class is “Immutable”

- Having created an object, we can’t change the values of the instance variables
- The “next” method returns a new Season object

```java
public Season season1 = Spring();
public Season season2 = season.Next();
season1.print();
season2.print();
```
What is “The Same”? 

Bean X is the “the same” bean as bean Y

So, If I grind bean X ...

- Is bean Y still whole?
Equality

Two Java objects are only equal (by default) if ...

- they are the **same object**

These may not be “equal”:

- `a = Spring().Next();`
- `b = Spring().Next();`

If you want a different definition of equality (as we do here) ...

- you need to create (and use) your own `Equals()` method
- If you want to use the objects as `HashMap` keys, you also need to create a `HashCode` method
- If you use some simpler representation of the seasons, these problems are avoided

If you have more complex objects ...

- You need to think carefully about how the equality relates to the equality of each of the members
Questions?