Overview

Overview of examinable material:

<table>
<thead>
<tr>
<th>Lectures</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>Compilation, variables and assignment</td>
</tr>
<tr>
<td>Week 2</td>
<td>Conditionals, loops</td>
</tr>
<tr>
<td>Week 3</td>
<td>Arrays</td>
</tr>
<tr>
<td>Week 4</td>
<td>Static methods</td>
</tr>
<tr>
<td>Week 5</td>
<td>Objects, data types, API</td>
</tr>
<tr>
<td></td>
<td>ILW</td>
</tr>
<tr>
<td>Week 6</td>
<td>Defining classes</td>
</tr>
<tr>
<td>Week 7</td>
<td>Encapsulation; ArrayList, HashMap</td>
</tr>
<tr>
<td>Week 8</td>
<td>Inheritance and polymorphism</td>
</tr>
</tbody>
</table>

What follows on the slides is a non-exclusive list of things to remember. Everything in the lectures or fundamental labs is examinable, unless I say otherwise.

Variables and Primitive Types

floating point division

- 2/3 vs. 2/3.0

testing if a value is even

- if (num % 2 == 0){...}

converting strings to numbers

- Integer.parseInt(), Double.parseDouble()

use boolean values directly

- Don’t say if (isHappy == true){...}, but if (isHappy){...}
- Where appropriate, use boolean operators in return values: return (isHappy || isAsleep);
# Conditionals and Loops

- Conditionals: pretty straightforward
- Check algorithms (see lecture slides) for using loops to compute
  - finite sums
  - products
  for example.
- Make sure you understand how to use nested loops, and how the flow of control goes.

# Arrays

```java
int[] a = {3, 1, 4, 1};
```

- a’s length is 4.
- Indexing lower bound is 0
- Indexing upper bound is 3 = a.length - 1
- a.length = upper bound + 1

Know how to allocate space for arrays:
```java
int[] b = new int[7];
String[] c = new String[a.length];
```

# Arrays

- Remember that elements of int[] set to 0 by default (similarly for double[])
- Check algorithms (see lecture) for:
  - find max (or min) value of array
  - compute average of an array
  - copy one array to another
  - reverse an array
- Use libraries to sort an array.

# 2D arrays:

- a[i][j] for row i and column j
- use nested loops
- review matrix addition and matrix multiplication

# Arrays and for loops

### Standard for loops

Be familiar with situations where:
- int i = 0 vs. int i = 1 (skipping first element)
- i < a.length vs. i < a.length - 1 (checking a[i + 1])
- i++ vs. i += 2 (pick every other element)
- (int i = a.length-1; i >= 0; i--) (counting backwards)
Standard and enhanced (for-each) for loops

```java
int[] values = {3, 1, 4, 1};
int sum = 0;

standard for loop
for (int i = 0; i < N; i++) {
    int n = values[i];
    sum += n;
}

enhanced for loop
for (int n : values) {
    sum += n;
}
```

- Variable `i` is assigned index values (0, 1, 2, 3)
- Access all elements in an array, from start to finish
- Variable `n` is assigned elements (3, 1, 4, 1)

Enhanced for loop: cannot modify elements in the array.

```java
for (int n : values) {
    sum += n;
}
```

Can’t use enhanced for loop if you want to:
- check if you are looking at first or last element of array
- traverse array backwards
- modify elements in the array

I/O

You are not expected to remember how to use `In` and `StdDraw`. You do need to be able to:
- Use `printf` and `String.format()`
- Be able to use e.g., `%.5.2f` (the 5 reserves five spaces for printing the string — includes the decimal place)

Static Methods

- Implement mathematical functions (using `Math` library).
- Use local variables and parameters.
- Assemble a program by composing multiple functions.
- Define and use overloaded functions.
- Pass-by-value: straightforward for primitive types.
- Pass-by-value for array arguments:
  - data inside an array reference type (or other reference types) can be changed as a side-effect.
  - alternative is to copy input array `a1` to a new array `a2`, then modify and return `a2`.
  - Be familiar with both approaches.
Classes and Objects

In general
▶ Distinguish instance variables from local variables and method parameters
▶ Define and use instance methods
▶ Inside class `A`, create new objects `b` of class `B` and call instance methods of `b`
  ▶ Write `tester` class as client to test code in another class
  ▶ Call instance methods from inside `main`

```java
public class CircleTester {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        double area1 = c1.getArea();
        System.out.printf("Area of circle c1 is %5.2f\n", area1);
        Circle c2 = new Circle(5.0);
        double area2 = c2.getArea();
        System.out.printf("Area of circle c2 is %5.2f\n", area2);
    }
}
```

APIs
▶ Given an API, implement a class that satisfies the API

Encapsulation
▶ Know when to use `private` and `final` modifiers on instance variables.
▶ Know how to create ‘getter’ and ‘setter’ instance methods.

Inheritance and polymorphism

Inheritance
▶ Define a subclass of a given class
▶ Do not duplicate the superclass’s data!
▶ Use `super(...)` to call the superclass’s constructor
▶ Use `super.method()` to call the superclass’s version of a method that you have overridden.

Polymorphism
▶ Understand which assignments work (use “is a” to remember)
▶ Use arrays and collections declared as containing objects of the superclass to store objects of a subclass too.

The String class

String
▶ Use `s.equals(t)` to check equality of strings `s` and `t`
▶ Know when and how to use methods like `substring()`, `startsWith()`, `endsWith()`, `indexOf()`, `split()`,
`charAt()`, `contains()`. 
▶ Review string processing snippets in lecture
String Quiz

Substring 1

String s = "abba";
int idx0 = s.indexOf("a");
int idx1 = s.lastIndexOf("a");
String sub1 = s.substring(idx0, idx1);

Substring 2

String s = "abba"
int idx0 = s.indexOf("a");
int idx2 = s.indexOf("e");
String sub2 = s.substring(idx0, idx2);

Sorting

- Sorting an ArrayList, vs sorting an array. Multiple ways, e.g.
  
  **Array**
  
  import java.util.Arrays;
  ...
  int[] myNums = new int[]{1, 3, 7, 5, 10};
  Arrays.sort(myNums);
  System.out.println(Arrays.toString(myNums)); // [1, 3, 5, 7, 10]

  **ArrayList**
  
  import java.util.ArrayList;
  import java.util.Collections;
  ...
  ArrayList myNums<Integer> = new ArrayList<Integer>();
  myNums.add(1); myNums.add(5); myNums.add(3);
  Collections.sort(myNums);
  System.out.println(myNums); // [1, 3, 5]

ArrayList and HashMap

Remark: you are expected to be most familiar with these collections, but the exam could easily expect you to look up and use other collection classes from the JDK.

**ArrayList**

- Use whenever you need an unspecified number of elements
- Type of element must be reference type (remember wrapper types Integer etc.)
- Be familiar with its API; e.g., does list contain a given element?

**HashMap**

- Use in situations where you want to associate data with unique keys
- Type of value and key must be reference type (e.g., Double not double)
- Can use e.g., ArrayList to collect multiple values
- Be familiar with its API e.g., does map assign a value to a given key?

Packages, using and defining

Using packages:
To use a class from the JDK, you need to import its package, e.g.

```java
import java.util.ArrayList;
```

Defining packages:
In the exam, **DON'T**!
In real systems, you will split your many classes into packages of related classes. Eclipse will tell you that the use of the default package is discouraged. But in Inf10P exams, we have not discussed this, and in the exam you must leave the classes you define in the default package – that is, containing no `package mypackage;` statements. If you do define a package, the basic tests will not pass (unless you modify them too, which you must not!)
Structure of Exam

Two questions, roughly equal length.
▶ Aim to examine both procedural and OO aspects
▶ May give you a skeleton and ask to “insert code here”
▶ May ask to write a class from scratch
▶ May ask you to implement an API
▶ May tell you something about a class, and ask you to use it (i.e. to implement a client)
▶ May ask you to write functions that test your code, but not JUnit
▶ May require you to look things up in the documentation
▶ If you know your stuff, will not be all that hard
▶ We don’t aim to put you under time pressure

Compilation and basic typing

In the exam you will be given some extremely basic JUnit tests, e.g., that call a method you are supposed to write, and succeed provided it runs and returns something of the correct type.
Do not modify the basic tests in any way. If you wish to write your own tests (not for submission but just to help yourself), do so in a separate test class.

Any code that does not compile, or does not pass these tests, will get no marks.

But you may well get partial credit for code that passes these basic tests, but fails some of the tests we use for automarking.

Suggested way of working

Get the basic tests passed first of all, and keep them passed!
E.g. if you have to write a method returning an integer, give it the correct signature and a body that just says return 0; to start with.

▶ Sometimes, code may be 99% correct, with just a very small error preventing compilation.
▶ If you can’t fix a bit of code that is preventing compilation, comment out the smallest part that will allow things to work.
▶ Commented out code will attract no credit itself – but if it lets your code compile, this might let you get credit for what you haven’t commented out.
▶ Do not comment out a whole method so that you fail basic tests!!

Marks for style?

There are no explicit marks for good coding style, laying out your code correctly, choosing good variable names etc.
But these things are their own reward! They will help you keep your thoughts straight.
Let Eclipse help you.
Can I implement more than I’m asked for?

Generally, no: write just what you are asked for, no more, no less.
Two exceptions:

1. If you want to write a helper method and call it from a method you are asked to write, that’s OK, provided your helper method is private.

2. If the question does not ask you to write a standard main method, you may still write one, and use it for testing purposes, if you find that helpful.

3. But note that if the question does ask you to write a standard main method then what you submit should do just what you’re told to do, no more no less. If during development you make your main method do more, for your own testing purposes, comment out the extra (and retest!) before submission.

Open book exam

The exam is open book – this means:

▶ You will be provided with some material on the exam machines, including the Oracle API documentation. (Same material available in the mock, so have a look then.)
▶ You may take in (a reasonable quantity of!) anything you like on paper: books, notes, printouts of your lab exercise solutions...
▶ The exam might well ask you to use something, e.g. an API class, that we didn’t explicitly cover, but that you can look up in the provided material.
▶ They might also forbid you to use something. If they neither instruct nor forbid, you may use anything you like from the JDK.

Think about what you’d like to take in, and make sure you know your way around it.

Mock Exam

▶ Could be/could have been a real exam.
▶ General trend over the last few years, including this year: exams are getting slightly shorter, slightly harder each year. Expect this.
▶ Two hours — plenty of time to make sure that your code compiles!
▶ Mock exam will only be auto-marked, with cruder automarking and less checking than the real exam.
▶ You will get an individual email with feedback; if the same mistakes occur repeatedly there will also be a webpage which I’ll email a link to.
▶ Feedback will arrive by the end of week 12, probably much sooner.
▶ Make sure mail from me to you won’t be spamfiltered!
▶ Strongly recommend reading the feedback, comparing your code with the sample code, trying it out on test arguments yourself, etc.

Main Exam

▶ Check announcements on UoE Registry page:
  http://www.ed.ac.uk/schools-departments/registry/exams
▶ definitive source of information, especially if there are sudden unforeseen circumstances
Finally

Please fill in the course questionnaire when you’re asked to do so.
If there are plenty of responses, this really helps.