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.

6th edition of Java Tutorial

▶ pp1-121
▶ pp193-226
▶ pp287-308
▶ pp423-505

Find the Electronic Version online.
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

- **int[] a = {3, 1, 4, 1};**
  - a’s length is 4.
  - Indexing lower bound is 0
  - Indexing upper bound is $a.length - 1$
  - $a.length = upper\ bound + 1$

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

- **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

- **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)

enhanced (for-each) for loops

Enhanced for loop: cannot modify elements in the array.

for (int n : values) {
    n = 0; // Wrong: doesn't modify array
}

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)

Standard and enhanced (for-each) for loops

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;
}

▶ Access all elements in an array, from start to finish
▶ Variable n is assigned elements (3, 1, 4, 1)
### 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

#### Constructors

- Define one or more constructors
- Use `this.x = x;` inside constructor with parameter `x`
- Use `this(...)` to call one constructor from another

#### APIs

- Given an API, implement a class that satisfies the API

#### Encapsulation

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

### 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);
    }
}
```

### 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

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?

Passing Command Line Arguments to Main Method

To write a sensible program, you need to be able to get access to data you want to work with, hence ...

You need to be able to:
- Pass a sequence of command line arguments to the main method of a program: `java MyClass a b 1 3` or Eclipse menu
- Access those arguments in your main method: `String[] args`
- Provide a path to a file in your project as command line argument: `java MyClass data.txt`

Next week's tutorials will have exercises to practice the last point.

Execute and Interpret JUnit Tests

Running unit tests to guarantee high production quality is essential, hence ...

You need to be able to:
- Import a Java file with JUnit tests into your project
- Add the required JUnit libraries to your build path (version 4)
- Compile and run the JUnit tests
- Interpret the test result to adapt your code

Those steps depend on your development environment (Eclipse or command line).

You are NOT required to write your own JUnit tests.
Working with Existing Code

More often then not, you have to work with other people’s code, hence if you are given a Java file with existing code...

You need to be able to:

▶ Use its methods or instances of it in your code (Library class)
▶ Fill in method stumps with your code (Skeleton class)

Know all the necessary steps in the development process to do so, e.g. importing into Eclipse project.

Working with Existing Code - Library Example

You are given a class `Greet.java`
```java
class Greet {
    /** Prints birthday whishes to the command line. */
    * @param name The name of the person to be greeted. */
    public static void birthday(String name) {
        System.out.printf("Happy Birthday, %s!\n", name);
    }
    /** Prints Christmas whishes to the command line. */
    * @param name The name of the person to be greeted. */
    public static void christmas(String name) {
        System.out.printf("Merry Christmas, %s!\n", name);
    }
}
```

You are asked to use it for whishing everyone in a list of command line arguments a happy birthday.
```java
class Mailer {
    public static void main(String[] args) {
        for(String name : args)
            Greet.birthday(name);
    }
}
```

Working with Existing Code - Skeleton Example

You are given a class `Calculator.java`
```java
class Calculator {
    private static int[] parseArgs(String[] args) {
        int[] nums = new int[args.length];
        for(int i=0; i<args.length; i++)
            nums[i] = Integer.parseInt(args[i]);
        return nums;
    }
    private static int sum(int[] nums) {
        int result = 0;
        for(int n : nums) result += n;
        return result;
    }
    public static void main(String[] args) {
        int[] nums = parseArgs(args);
        int sum = sum(nums);
        System.out.println("The total is: " + sum);
    }
}
```

You are asked to fill in the empty methods with the proper implementations.
Exam Machine Setup

- Likely one of the computer labs in AT
- You will get a DICE machine in Exam-Lock-Down mode
  - no access to your university home directory
  - no internet access
  - java 8 API docs, Oracle tutorials and lecture slides are available
- Instruction Sheet
  - how to get the exam paper and template files
  - how to submit your solution files

Exam Format

- Exam Paper
  - You will get two questions in the exam which you are required to solve. Each question has roughly the following structure:
    1. Setting the scene (define the problem, give some background, show examples)
    2. Which resources should you use for your solution (java files, data files, basic tests)
    3. Subquestions to be answered
    4. Remark on which file to submit as solution

What Type of Questions to Expect

- 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 Java documentation
Golden Rules

Three golden rules for submitting exam solutions

1. Submit code that compiles!
2. Submit code that passes the basic unit tests!
3. Do not define your own packages!

If you violate any of those three rules for one of the two questions in the exam, you will get ZERO marks for that question!

Submit code that passes the basic unit tests!

You will be given some extremely basic JUnit tests.

- They will help you to evaluate if at least the start of your answer is correct but do not fully evaluate your solution.
- Do not modify the basic tests in any way.
- If not ALL of the basic tests for one of the two main questions executes successfully, you won't get any marks for that question.
- But you may well get partial credit for code that passes these basic tests, but fails some of the tests we use for automarking.

Do not define your own packages

Using Packages

To use a class from the JDK, you need to import its package, for example:

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

This is allowed and required in the exam.

Defining Packages

To manage your code base you can put multiple classes into different packages by copying the files into subfolders and adding a `package mypackage;` statement to each class you write.

This is NOT allowed in the exam.
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!!

Practice this with your tutor using old exam questions.

Example - The Task

Implement a Calculator class which has three static methods:

▶ A public static method add which gets two integer parameters, adds them together and returns the result as an integer.
▶ A public static method mul which gets two integer parameters, multiplies them with each other and returns the result as an integer.
▶ A public static method incrementAll which gets, in that order, an integer array nums and an integer inc parameter and increments each entry in nums by inc. It returns nothing.

Example - The Tests

```java
import org.junit.Test;
public class CalculatorBasicTest {
    @Test
    public void testAdd() {
        int res = Calculator.add(1, 2);
    }
    @Test
    public void testMul() {
        int res = Calculator.mul(1, 2);
    }
    @Test
    public void testIncrementAll() {
        int[] seq = {1, 2, 3};
        Calculator.incrementAll(seq, 5);
    }
}
```

Example - First Step

Write method stumps.

```java
public class Calculator {
    public static int add(int a, int b) {
        return 0;
    }
    public static int mul(int a, int b) {
        return 0;
    }
    public static void incrementAll(int[] nums, int inc) {
        // Method implementation goes here.
    }
}
```
Fill in solutions.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        for(int i = 0; i < nums.length; i++)
            nums[i] += inc;
    }
}

Example - Comment out Code

Comment out code for your submission that does not compile.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        // for(i = 0; i < nums.length; i++)
        //     nums[i] += inc;
        // }
    }
}

Example - Fill in Solutions

Fill in solutions.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        for(int i = 0; i < nums.length; i++)
            nums[i] += inc;
    }
}

Example - Comment out Code

Comment out code for your submission that does not compile.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        // for(i = 0; i < nums.length; i++)
        //     nums[i] += inc;
        // }
    }
}

Example - Fill in Solutions

Fill in solutions.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        for(int i = 0; i < nums.length; i++)
            nums[i] += inc;
    }
}

Do this!

Example - Comment out Code

Comment out code for your submission that does not compile.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        // for(i = 0; i < nums.length; i++)
        //     nums[i] += inc;
        // }
    }
}

Example - Fill in Solutions

Fill in solutions.

class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int mul(int a, int b) {
        return a * b;
    }

    public static void incrementAll(int[] nums, int inc) {
        for(int i = 0; i < nums.length; i++)
            nums[i] += inc;
    }
}

Don't do that!
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.
▶ You may NOT bring a USB flash drive or anything else electronic.

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.
How do I practice for the Exam?

- Finish all the basic labs (you hopefully have done that by the end of next week!)
- Implement the solutions to old exam questions
- Use the online automarking service to test your submissions for old exams
- Simulate the exam environment: no internet, only your notes
- **GO TO THE MOCK**
  - I will keep answering questions on Piazza or see you in my office to help you out

Main Exam

- Check announcements on UoE Registry page
- definitive source of information, especially if there are sudden unforeseen circumstances

I Want Your Feedback!

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