SOFTENG 251: Object-Oriented Software Construction

Lecture 1: Introduction & Java Overture

Ewan Tempero
Department of Computer Science

Potential Assessment Question (ENGGEN 131)

1. Which of the following lines will cause a compilation error in C, assuming `x` is a variable of type `int`?
   (a) `if (x == 1) {`
   (b) `if (x = 1) {`
   (c) `if (x =! 1) {`
   (d) `if (x != 1) {`
   (e) None of the above.
2. What is the output of the following function when called as `q2(2)`?

```
void q2(int x) {
  if (x = 1) {
    printf("One\n");
  } else {
    printf("Not one\n");
  }
}
```

   (a) There will be no output because it will not compile.
   (b) One
   (c) Not one
   (d) The output will be an error message.
   (e) None of the above

Agenda

- Admin
  - *Ewan, have you remembered to start the lecture recording?*
  - SOFTENG 251 details
- A Java Overture — The Hello World program
- Assignment 1
- Week 1 Lab

- Reading:
  - Lesson: The ”Hello World!” Application (Java Tutorials)
  - Lesson: The Java Technology Phenomenon (Java Tutorials)
  - Reading: Chapters 1 & 2

People

Ewan Tempero Course Coordinator, Lecturer
e.tempero@cs.auckland.ac.nz
Office hours: Open door

Ian Warren Lecturer
i.warren@auckland.ac.nz
Office hours: Open door

Tutors Andrew Meads, PhD student
Victor Vix (Vong Vithyea Srey), Part III SOFTENG
Contact via `piazza.com`

Communication

- Meetings — lectures, tutorials, labs. Attendance expected
- Resources
  - Reading: An introduction to Object-Oriented Programming, Timothy Budd
  - Cecil — assignment handouts, possibly other stuff Check out the "Knowledge Map"
- Course marks — Cecil `http://cecil.auckland.ac.nz`
- Email — Electronic Mail is an official and the primary means of communication with students
- Journal
  - Bound book(s)
  - May be taken into tests and exams
  - Can be used to record: lecture notes, thoughts, questions, work log
  - Cannot contain material pasted in
- `piazza.com`
  - general questions and discussion
  - Do not post answers to assignment questions

Meetings

Lectures & Tutorials Monday, Tuesday, Wednesday, Thursday

Lab Room 303S.191 (aka First Floor Science Computer Lab) Wednesday 10-12

Attendance expected for all

Course Information Sheet

- Provisional course topic list
- Assessment details
- Where to find things/people
- How to contact people
- Other information you are expected to know
- Available on Cecil

Goals for First Half

- Learn how to program in Java
- Learn standard object-oriented concepts
- Start to learn object-oriented design
- Learn how to express designs in Java

Assessment for First Half

- Assignment 1, Due Friday 21st March (end of week 3) — 6%

  Write lots (and lots) of little, unrelated, pieces of code in Java — emphasis on learning Java

- Assignment 2, Due Friday 11th April (end of week 6) — 6.5%

  Write lots of related pieces of code in Java to implement some functionality — emphasis on learning object-oriented programming

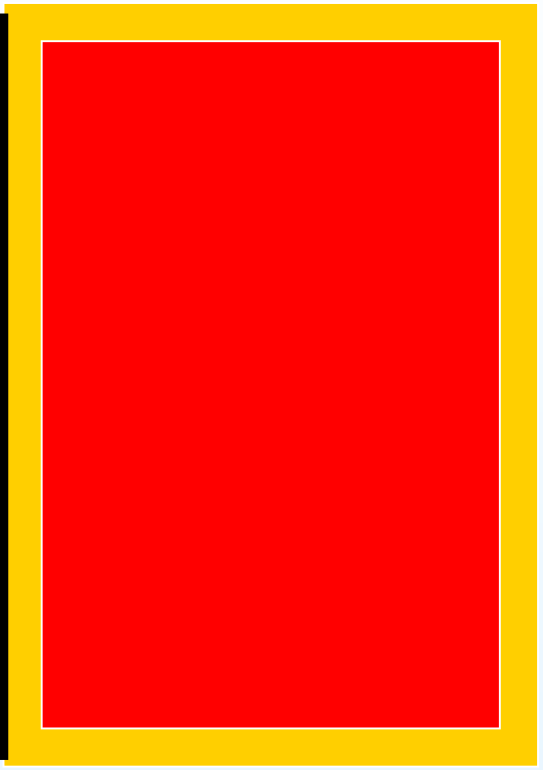
- Test 1, Thursday 10th April (week 6) — 7.5%

  Answer lots (and lots (and lots)) of little questions related to the first half of the course.

Myth #1

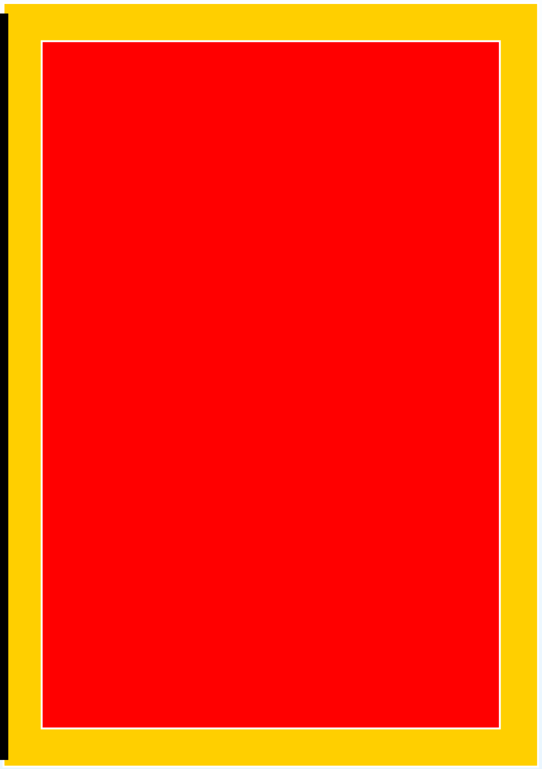
Myth #1

Java is a simple language



Myth # 2

Object-oriented programming is "natural"



Previously in your BE

In file `hello.c`

```
#include <stdio.h>

/* This program displays a welcome message */
int main(void)
{
  printf("hello world\n");
  return 0;
}
```

- C programs are a bunch of files containing functions and variable declarations

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Java programs consist of a bunch of classes (and other similar things), *usually* one per file.
- Classes (and similar things) contain methods and fields, collectively called members
  - C equivalents are (roughly) “functions” and “global variables”

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Java programs consist of a bunch of classes (and other similar things), *usually* one per file.
- Classes (and similar things) contain methods and fields, collectively called members
  - C equivalents are (roughly) “functions” and “global variables”

A Java Overture

In file `Hello.java`

```java
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Methods have a name, parameters, and a return type.
- Methods are *similar but not (always) identical to* functions

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- keywords are reserved — cannot be used as identifiers (names of variables, methods, fields, parameters, and so on)

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- keywords are reserved — cannot be used as identifiers (names of variables, methods, fields, parameters, and so on)
- Some keywords look familiar from C, but *not always identical* in meaning
  - a return “type” of void means do not return any value (as in C)

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- keywords are reserved — cannot be used as identifiers (names of variables, methods, fields, parameters, and so on)
- Some keywords look familiar from C, but *not always identical* in meaning
  - a return “type” of void means do not return any value (as in C)
- Think of `static` as meaning “acts like C function”

A Word on Types

- All names used for memory locations that hold values (variables, parameters, fields) must have a declared type
  - Originally, indication to compiler of which instructions to use (e.g. does "+" mean fixed point — integer — addition or floating point addition?)
  - Now, indication to compiler of what operations are going to be done on values

  ⇒ if try to use other operations, then (Java) compiler will disallow

  ⇒ provides useful **safety check** providing an early warning for certain kinds of errors

Example

```
a = 13;
^
(a) Error: a cannot be unresolved  or
(b) Error: cannot find symbol  or similar‡
```

- **A**lso **K**nown **A**s "undeclared variable"
- Translation: You didn't tell me what operations you planned to do with "a"

(a) The process of figuring out what an identifier means is called *resolution*, so if the identifier has not been declared, then it cannot be resolved.

(b) All declared identifiers (aka "symbols") are stored in a data structure within the compiler known as a Symbol Table. If you don't declare the identifier, then the "symbol" cannot be found in the symbol table.

‡ Different compilers give slightly different error messages (unfortunately)

Example

```
double d = 1.0;
int i = d;
^
```

(a) `Error: Type mismatch: cannot convert from double to int` or

(b) `Error: possible loss of precision` or similar

(a) Translation: You said you wanted to use `int` operations on “`i`” but you are trying to use `double` operations, and I’m not allowed to convert `int`s to `double`s without being told

(b) There are a finite number of different `int` values. There are a finite (but much larger) number of different `double` values. Therefore when assigning doubles to `int`s, some `double`s will be lost (see COMPSYS 201).

Example

```
int i = 1;
double d = i;
```

- Fewer `int`s than `double`s, so no loss of precision assigning an `int` to a `double`
- But, (for example) the binary representation for the `double` `2.0` is not the same as the binary representation for the `int` 2 (see COMPSYS 201).
- BUT, Java knows how to convert from the `int` binary representation to the `double` representation for the same value and will do so without being told to.

Casts — Trust me, I'm a Software Engineer

```
double d = 1.0;
int i = (int)d;
```

- "cast" — tell the compiler to convert values of one type to another and trust you that you know what you're doing
- Can only be done between "compatible" types (e.g. `double` to `int`)
- Can lead to problems (e.g. due to loss of precision), so make sure you know what you are doing!

A Java Overture Continued

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[]args) {
        System.out.println("hello world");
    }
}
```

- *Some* types are “built-in” (from the Java Standard Library, also-known-as “Standard API” “JDK” “JRE”)
  - Most (but not all) such types must be “imported” — similar to C `#include`

A Java Overture Continued

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[]args) {
        System.out.println("hello world");
    }
}
```

- *Some* types are "built-in" (from the Java Standard Library, also-known-as "Standard API" "JDK" "JRE")
  - Most (but not all) such types must be "imported" — similar to C `#include`
- Arrays are *similar, but not identical to* C arrays

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Java has three (!) different kinds of comments:
  - C-style multi-line comments between "`/*`" and "`*/`"
  - "documentation" multi-line comments between "`/**`" and "`*/`"
  - "single-line" comments following "`//`"

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- The main method is the “entry point” for any Java program, just as in C.

A Java Overture

In file `Hello.java`

```
/**
 * This class prints out a welcome message.
 */
public class Hello {
    // This displays a welcome message.
    public static void main(void) {
        System.out.println("hello world");
    }
}
```

- **The return type and parameters must be exactly as stated**

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- No `#include <stdio.h>` is needed to get at *standard* input/output
  - Other kinds of input/output will need something imported
- System.out.println is similar (but not identical) to printf

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- No `#include <stdio.h>` is needed to get at *standard* input/output
  - Other kinds of input/output will need something imported
- System.out.println is similar (but not identical) to printf

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.print("hello world\n");
    }
}
```

- No `#include <stdio.h>` is needed to get at *standard* input/output
  - Other kinds of input/output will need something imported
- System.out.println is similar (but not identical) to printf

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Functions are “called”, Methods are “invoked”

A Java Overture

In file `Hello.java`

```
/**
  * This class prints out a welcome message.
  */
public class Hello {
    // This displays a welcome message.
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

- Functions are “called”, Methods are “invoked”
- Methods are invoked on “objects” using “.”
- Objects are typically identified by name (but not always)

Making Java Programs Run

Lesson: The Java Technology Phenomenon

- A Java program consists of a set of files with suffix `.java`
- The name of the file must be consistent with the class declared within it: `Hello.java` contains class `Hello`
- `.java` files are *compiled* into `.class` files, containing bytecodes

```
se251 prompt> ls
Hello.java
se251 prompt> javac Hello.java
se251 prompt> ls
Hello.class  Hello.java
```

Making Java Programs Run

- Java programs are interpreted by the Java Virtual Machine (JVM). The name of the class is passed to the JVM; the class must have the `public static void main(String[] args)` method

```
se251 prompt> java Hello
hello world
```

- “java Hello”
  - look for the file “Hello.class” in the empty/default package (found in the current working directory)
  - Execute the “main” method in “Hello.class”

Assignment 1

- Implement lots of exercises requiring implementing Java methods
- Use Web-based system, CodeWrite `codewrite.cs.auckland.ac.nz`
- Due: All required exercises must be completed by 5pm Friday 21st March
- Worth 6% of final grade
- Practice exercises in Lab this week
- More details to come

Review

- SOFTENG 251 details
- The `main` method of classes
- Methods — declaration and calling
- Modifiers `static` and `public`
- The `String` class from the JDK
- Arrays
- Comments
- Printing to the display
- Running Java programs