Structured Programming

COMP1110/6710

Australian National University

Image Courtesy NASA/JPL-Caltech.

On U4

- Lots of 0s on tests, because of unchanged static interface methods
- Partial Credit through Code Walk
- But: potential to cure in mark moderation
- For that, we'll allow you to submit a fixed version of the interface (see rules on next slide)

- NO GUARANTEES, but we'll have your back if the difference would make you fail the course

Curing U4 - Rules

- DO NOT touch a single character outside the static methods that make up the testing interface
- DO NOT put any important logic into the static methods – they need to be calling existing methods/constructors in the code you already submitted, module constants/minor expressions (such as x+1)
- How the fix is scored is decided in mark moderation, but if you violate the rules, your assignment marks become 0 (otherwise, you'll get the better mark)
- How to submit this – to be announced

Quiz time!

pollev.com/fabianm
Register for Engagement

Log in with ANU Account!

Exceptions

Not the ones where you get more marks

Australian National University

From Week 6: Reading (Text) Files

```
try(var reader = new BufferedReader(new FileReader(file))) {
   for(String line = reader.readLine(); line != null;
               line = reader.readLine()) {
       … [do something with line] …
   }
} catch(Exception e) {
  throw new RuntimeException(e);
}
```


Exceptions & Control Flow

Normal Control Flow

- `return` statements/end of method go back to current position in previous method on stack
- returned value has to match return type

Exceptional Control Flow

- `throw` statements go to closest matching catch-block on stack (which may be in current method)
- thrown value may have to be caught or match `throws` clause

Throwing Exceptions

```
throw new RuntimeException();
```


Throwing Exceptions

```
Type method([args] …) {
  …
  try {
   …
   throw new RuntimeException();
   …
  } catch(…) { …
  } catch(RuntimeException e) {
      …
  }
  …
}
```

Matching catch-block
e refers to thrown exception object

Continue after last catch-block

```
Type method([args] …) {
  …
  try {
   …
   throw new RuntimeException();
   …
  } catch(…) {
      …
  }
  …
}
```

zero or more try-blocks

no matching catch-blocks

Look further down the call stack

Throwing Exceptions

```
Type method2([args] …) {
  …
  try {
   …
   ….method(…);
   …
  } catch(…) { …
  } catch(RuntimeException e) {
      …
  }
  …
}
```

Matching catch-block
e refers to thrown exception object

```
Type method2([args] …) {
  …
  try {
   …
   ….method(…);
   …
  } catch(…) {
      …
  }
  …
}
```

zero or more try-blocks

no matching catch-blocks

Keep looking further down the stack

Throwing Exceptions

```
public static void main(String[] args) {
  …
  try {
   …
   ….methodN(…);
   …
  } catch(…) {
      …
  }
  …
}
```

zero or more try-blocks

no matching catch-blocks

Crash the program, print stack trace

Try/Catch

```
try {
    …
} catch(ExceptionType1 e) {
    …
} … {  …
} catch(ExceptionTypeN e) {
    …
}
```

If an exception is thrown in here, and not caught earlier …

Some number of catch clauses

Must be subtypes of `Throwable`

… we try to match its (run-time) type against the catch-clause, using the first matching one.
If no match, look for matching try-catch outside of this one, possibly further down the call stack.

Throw

`throw e;` Must be instance of `Throwable`

```
throw new MyExceptionClass(…);
```

```
class MyExceptionClass
  extends Exception { … }
```


Throws

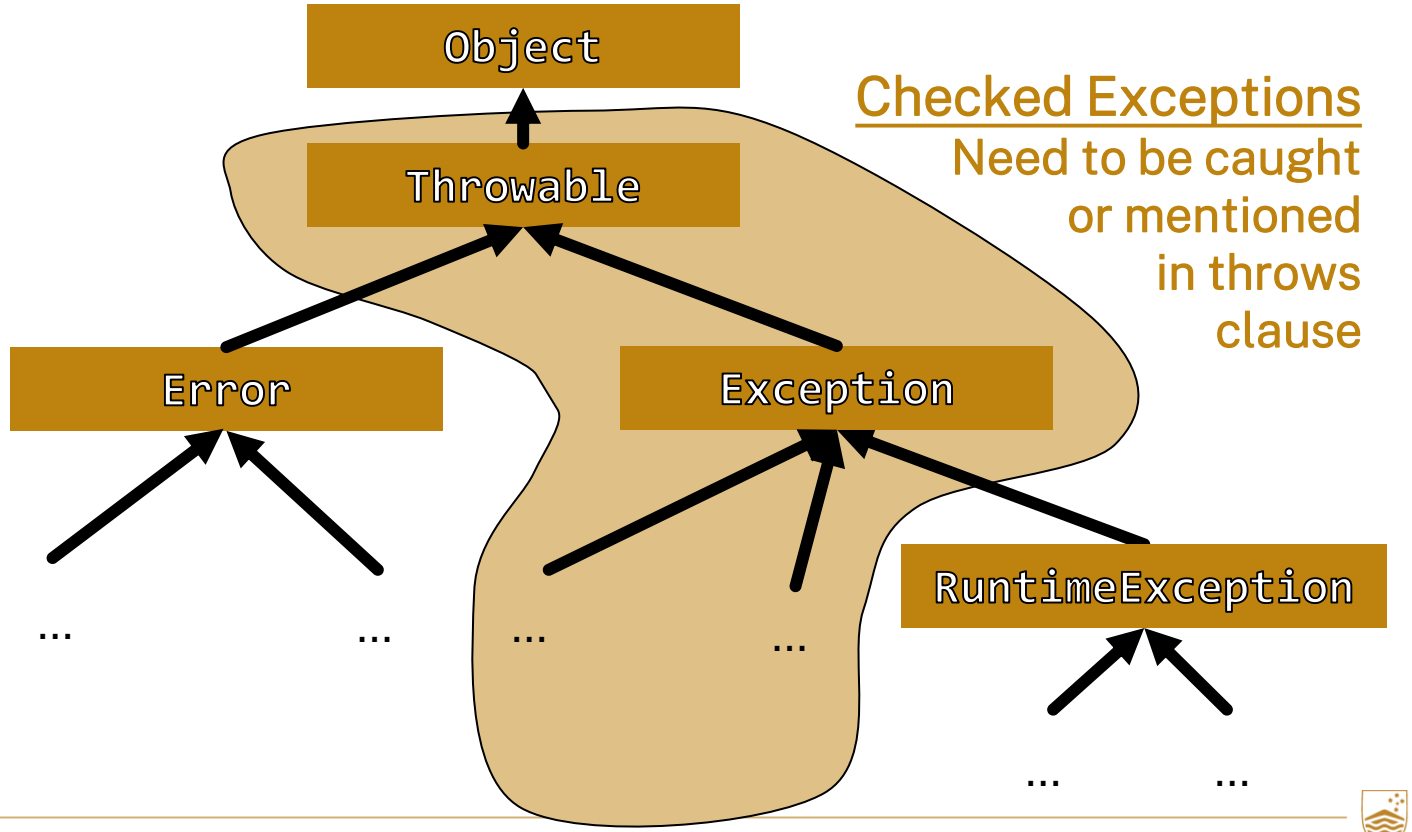
```
/** …
 * @throws MyException [reasons for MyException]
 * @throws MyOtherException [reasons for MyOtherException]
 * … */
Type myMethod([args]…) throws MyException, MyOtherException {
    …
}
```

Needs to be mirrored in human signature

Must be subtypes of `Throwable`

Part of the Java signature of a method. Says that the method may throw exceptions of those types, and callers should prepare to handle them.

Exception Hierarchy



Checked Exceptions
Need to be caught or mentioned in throws clause

Exception Hierarchy

Checked Exceptions

You should be able to handle these. Need to be caught or declared in `throws` clause – part of the Java signature.

Most common: `IOException`, `SQLException`

`RuntimeException`

You may not be able to handle these.
Only part of human signature , if at all.

Typically, these are excluded by pre-/postconditions/invariants.
E.g. `NullPointerException`, `IndexOutOfBoundsException`

`Error`

You may not be able to handle these.
Only part of human signature, if at all.

Caused by things outside of the program.
E.g. `IOError`, `VirtualMachineError`

finally-Blocks

```
try {
    …
} catch ( … ) {
    …
} finally {
    //cleanup
}
```

finally-blocks are an optional last part of try-blocks.

They are always executed when the try-block is left:

- through regular control flow (return, continue, break, reaching the end of normal execution or a catch-block)
- By throwing an exception

finally-Blocks

```
Type methodN([args] …) {
  …
  try {
   …
   …
  } catch(…) {
      …
  }
  …
}
```

zero or more try-blocks

no matching catch-blocks

Keep looking further down the stack
Run any finally-blocks belonging to try-blocks we are leaving

finally-blocks are an optional last part of try-blocks.

They are always executed when the try-block is left:

- through regular control flow (return, continue, break, reaching the end of normal execution or a catch-block)
- By throwing an exception

try-with-resources

Must be instance of `java.lang.AutoCloseable`

```
try(var reader = new BufferedReader(new FileReader(file))) {
   for(String line = reader.readLine(); line != null;
               line = reader.readLine()) {
       … [do something with line] …
   }
} catch(Exception e) {
  throw new RuntimeException(e);
}
```

A safer version of calling `reader.close()` in finally-block

Chained Exceptions

```
try(var reader = new BufferedReader(new FileReader(file))) {
   for(String line = reader.readLine(); line != null;
               line = reader.readLine()) {
       … [do something with line] …
   }
} catch(Exception e) {
  throw new RuntimeException(e);
}
```

Wrap current exception (which may be a checked exception) in another exception that works with the interface of the current code (i.e. an unchecked exception, or one that is declared in throws clause, or caught somewhere in the context).

Quiz time!

pollev.com/fabianm
Register for Engagement

Log in with ANU Account!

Packages and the Class Path

Organizing your code and avoiding name clashes

Australian National University

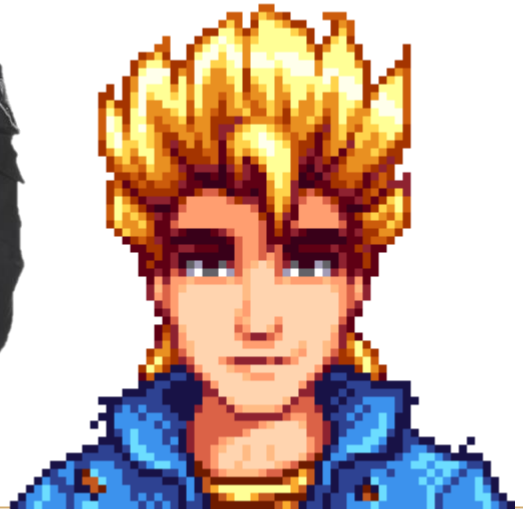
Java Packages

“Sam” – but which one?

stargate.Sam

© MGM

stardewValley.Sam



© ConcernedApe

dannyPhantom.Sam

© Nickelodeon

lordOfTheRings.Sam

© New Line Cinema

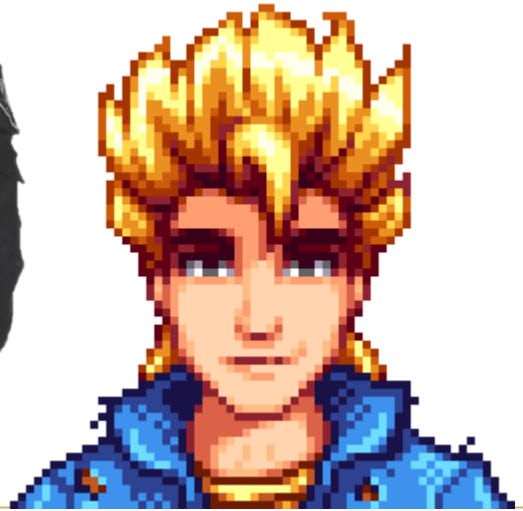
Java Packages

Namespaces in which "Sam" is well defined.

stargate.Sam

© MGM

stardewValley.Sam



© ConcernedApe

dannyPhantom.Sam

© Nickelodeon

lordOfTheRings.Sam

© New Line Cinema

Java Packages

- Mirror directory structure of source and class files
- Source files need to declare:
  `package path.to.package.folder;`
- Packages can be imported with:
  `import path.to.package.folder;`

  (Folder paths are relative to class path, coming up)

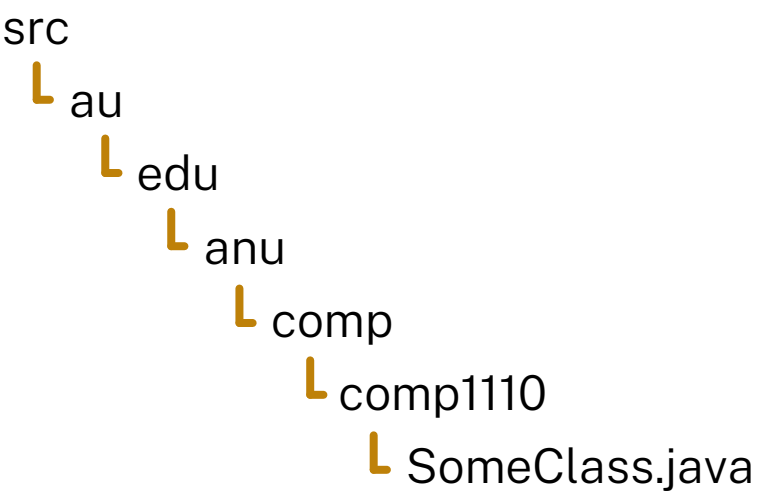
Package Conventions

If you write a library, use an internet domain as package prefix, in reverse order.

E.g.

package au.edu.anu.comp.comp1110;

Folder structure:



The Default Package

For simplicity, the root source folder is the “default” package.

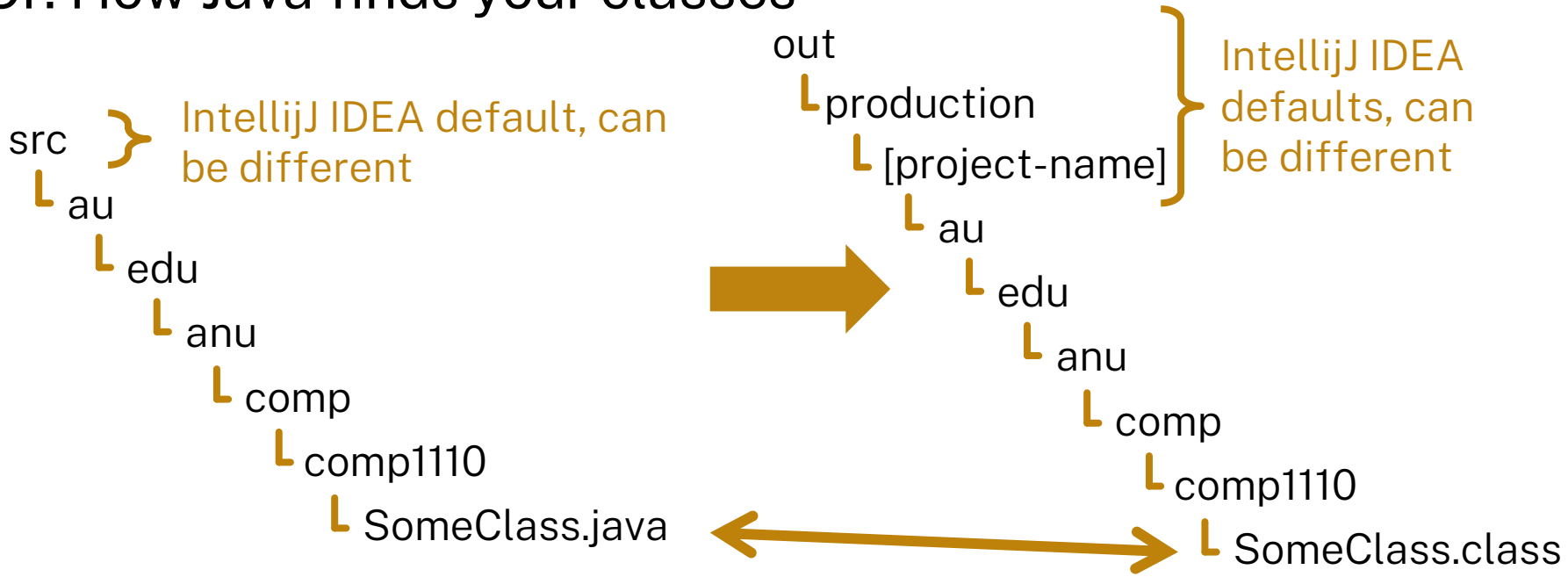
There is no package declaration for it, but you also cannot import it.

Things in actual packages cannot refer to things in the default package, but things in the default package can refer to anything.

We used this in all assignments so far.

The Class Path

Or: How Java finds your classes



The Class Path

Or: How Java finds your classes

Like the PATH environment variable, the Class Path is a collection of “roots”.

E.g.: “out/production/[project-name]/:lib/comp1110lib.jar”

Path separator on Linux/Mac – on Windows, use “;”

The Class Path

Or: How Java finds your classes

Like the PATH environment variable, the Class Path is a collection of “roots”.

E.g.: “out/production/[project-name]/:lib/comp1110lib.jar”

Essentially a Zip-file containing a folder structure with class files, like a regular out/production/[project-name].

The Class Path

Or: How Java finds your classes

Like the PATH environment variable, the Class Path is a collection of “roots”.

E.g.: “out/production/[project-name]/:lib/comp1110lib.jar”

When you `import au.edu.anu.comp.comp1110.SomeClass;`
Java will look to find the corresponding file at the corresponding path starting from any component of the class path.

The Class Path

Or: How Java finds your classes

Like the PATH environment variable, the Class Path is a collection of “roots”.

E.g.: “out/production/[project-name]/:lib/comp1110lib.jar”

On the command line, write:

```
java –cp “out/production/[project-name]/:lib/comp1110lib.jar” au.edu.anu.comp.comp1110.SomeClass
```

to find that class starting from one of the class path components and try to run its main method.

Quiz time!

pollev.com/fabianm
Register for Engagement

Log in with ANU Account!