# CSE1710

Week 05, Lecture 09

Fall 2013 ◆ Tuesday, Oct 8, 2013

YORK UNIVERSITÉ UNIVERSITY

## Big Picture

The assigned reading was for today:

- Software Engineering sec 2.3
- Risk Mitigation Early Exposure; sec 2.3.1, pp. 71
- Handling Constants; sec 2.3.2, pp. 71-72
- **Contracts**; sec 2.3.3, pp. 73-77



YORK UNIVERSITÉ UNIVERSITY

# Checklist (for Today)

What we are reinforcing with the exercises this class…

- ❑ ability to successfully complete Ch 2 RQ's 1-35
- ❑ ability to successfully complete Ch 2 Ex's 2.1-2.22

This is the final lecture on Chapter 2!



YORK UNIVERSITÉ UNIVERSITY

# Checklist (for next time, Lecture 10)

What you should be doing to prepare for what comes next…

- ❑ read section 3.1 "Anatomy of an API"
- ❑ review Ch 3 KC's 1-6
- ❑ do Ch 3 RQ's 1-13
- ❑ do Ch 3 Ex's 3.1-3.11

The next **three lectures** are on topics related to Chapter 3.
Final lecture on Chapter 3 is Thursday, Oct 17.
Written Term Test on Chapters 1-3 is scheduled for **Tues Oct 29**



YORK UNIVERSITÉ UNIVERSITY

Here is a question from last year's final exam



**Question Q1: Contracts** *<8 marks>*

This question makes use of the class `FinalExamServices`. The API for this class can be found in the appendix, which is provided in a separate handout.

**Q1–1. [2 marks]**

```
1 public class Question1App1 {
2   public static void main(String[] args) {
3     String theString = "Parker Bowles, Camilla Rosemary, rottweiler@yorku.ca";
4     String x = FinalExamServices.extractFirstNameStrict(theString);
5     System.out.println(x);
6   }
```

**Scenario:** Suppose the app runs and the following is printed to `System.out` (as resulting from the method invocation in line number 5). (The console output is shown between the two horizontal lines below).

```
rottweiler
```

(a) Which of the following applies: (check one)

- ☐ The contract was fulfilled.
- ☐ The contract was not fulfilled; the client did not fulfill its part.
- ☐ The contract was not fulfilled; the implementer did not fulfill its part.

5

## extractFirstNameStrict

```
public static java.lang.String extractFirstNameStrict(java.lang.String rowFromClassList)
```

This method extracts the *First Name* element from a String that represents **one row** of a York University class list. Class lists typically look like this:

```
911002999, Mountbatten-Windsor, William, wills@yorku.ca
911003999, Mountbatten-Windsor, Henry, prdisaster@yorku.ca
912002888, Middleton, Catherine Elizabeth, publiclypreggers@yorku.ca
```

This method is denoted "Strict" because of its strict precondition about the format of the String parameter.

**Parameters:**

`rowFromClassList` - a String that adheres to the following formatting convention: Each student is represented by one string that has four *comma-delimited* elements: Student Number, Last Name, First Name, and Email Address. Rules about the elements:

- The First Name and Last Name element must each be **at least** one character long. Names must consist of only **alphabetic** characters (A-Z), in either lower or upper case, with no special characters. The First Name and Last Name element may consist of one or more names, provided the names are separated by a single space, hyphen, or single apostrophe.
- The Student Number element must be 9 characters long and must contain **only** digits.
- The Email element must be a string that contains exactly one character @, with at least one character before and after the @ character.

**Returns:**

a String containing the First Name, as described above

6

**RQ2.28** Can the main class be the culprit when a runtime error occurs in a component?

Before answering this, let's discuss, more generally, how do we determine who is the culprit when runtime errors occur.

And while we're at it, let's review the concept of a runtime error.

7

YORK UNIVERSITÉ UNIVERSITY

**RQ** What are the three categories of errors?
Briefly describe each one and provide an example.
Identify how each type of error is triggered.

1. _____

2. _____

3. _____

From this exercise, you should now have a clear idea of what is meant by a runtime error.

8

YORK UNIVERSITÉ UNIVERSITY

RQ2.25

**What does the VM do when a program crashes or has a bug?**

- for crashes
  - VM identifies where the problem occurs in the stack trace
- for bugs
  - VM will not realize that there is a bug, so it cannot possibly flag them
- debugging
  - you (not the VM) need to determine why the program produced an incorrect result
  - may need to trace the entire program



RQ2.26

**So what is the difference between a bug and other types of errors?**

- a bug
  - depends on some notion of what correct output looks like
- compile-time error
  - compiler has a problem with the syntax
  - need to understand compiler's error message
- run-time error
  - VM had a problem running the byte code
  - need to understand stack trace

# Brief Recap of Runtime Errors

- Runtime errors happen while the program is running. The program abruptly and unexpectedly stops (aka **CRASH**).
- The VM triggers runtime errors
- Runtime errors are different from compilation errors and from logic errors.
- Just because an app is free of runtime errors, it still might have errors.
  - runtime-error free ≠ error free
  - there still may be logic errors!

YORK UNIVERSITÉ UNIVERSITY



**RQ** Who is the culprit when runtime errors occur?

**To answer this, you might ask**
**who are the players?**
**what are their responsibilities and expectations?**

YORK UNIVERSITÉ UNIVERSITY

# Responsibilities and Expectations

**!!! KEY THING TO KEEP IN MIND !!!**

Responsibilities and expectations in the client-implementer software domain

are NOT THE SAME as

Responsibilities and expectations in everyday life (as encoded into ethics, law)

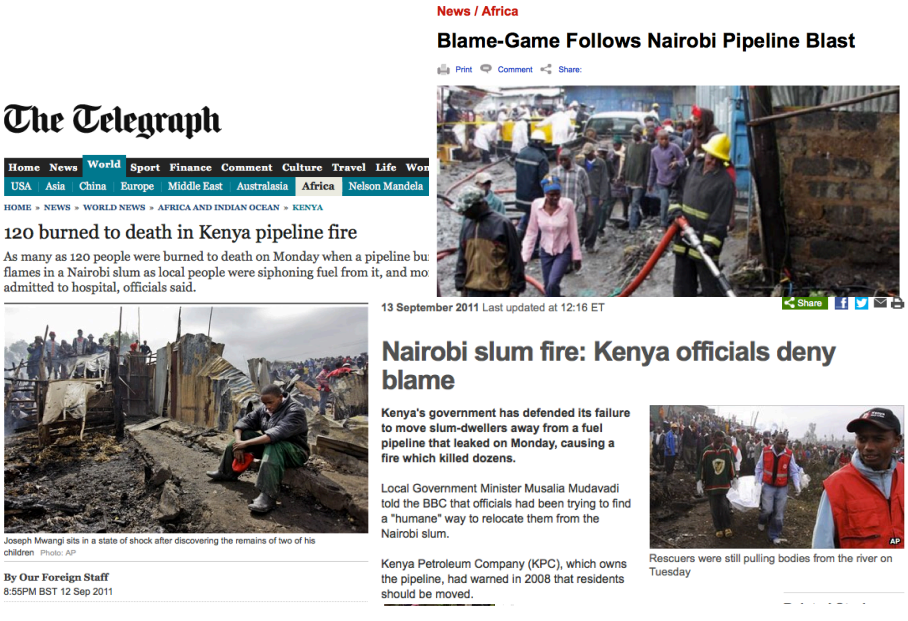
You need to "suspend" your intrinsically-held ideas about justice in order to "get" the concepts about contracts



YORK UNIVERSITÉ UNIVERSITY

# Illustration

*Who is to blame? Kenya Petroleum Company? The Kenyan authorities? The slum dwellers? Shared blame?*



The Telegraph

Home News World Sport Finance Comment Culture Travel Life Wor
USA Asia China Europe Middle East Australasia Africa Nelson Mandela

HOME » NEWS » WORLD NEWS » AFRICA AND INDIAN OCEAN » KENYA

120 burned to death in Kenya pipeline fire

As many as 120 people were burned to death on Monday when a pipeline bu flames in a Nairobi slum as local people were siphoning fuel from it, and mo admitted to hospital, officials said.

Joseph Mwangi sits in a state of shock after discovering the remains of two of his children Photo: AP

By Our Foreign Staff
8:55PM BST 12 Sep 2011

News / Africa

Blame-Game Follows Nairobi Pipeline Blast

Print Comment Share:

13 September 2011 Last updated at 12:16 ET

Share

Nairobi slum fire: Kenya officials deny blame

**Kenya's government has defended its failure to move slum-dwellers away from a fuel pipeline that leaked on Monday, causing a fire which killed dozens.**

Local Government Minister Musalia Mudavadi told the BBC that officials had been trying to find a "humane" way to relocate them from the Nairobi slum.

Kenya Petroleum Company (KPC), which owns the pipeline, had warned in 2008 that residents should be moved.

Rescuers were still pulling bodies from the river on Tuesday

# Responsibilities and Expectations

Expectations:

- the client
  - needs to ensure the precondition is met
- the implementer
  - needs to ensure the postcondition is met

Responsibilities:

- are **all-or-nothing**
- there is no such thing as **sharing the blame**
- the culprit is either the client **OR** the implementer



YORK UNIVERSITÉ UNIVERSITY

# Expectations

**getBMI**

```
public static double getBMI(double weight,
                            java.lang.String height)
```

Compute the body mass index.

**Parameters:**

`weight` - the weight in pounds. To be valid, weight must be positive.

`height` - the height in feet'inches. To be valid, height must have a feet component (a positive integer) optionally followed by an inches component (a non-negative integer less than 12). And if both components are present then they must be separated by a single quote.

**Returns:**

the body mass index (BMI) for the given weight and height

**Throws:**

`java.lang.RuntimeException` - if either weight or height is not valid as defined above.

potentially throws an exception!!!

**repeat**

```
public static java.lang.String repeat(int count,
                                      char c)
```

Create a string containing the passed character repeated as many times as specified. If count is less than 1, a zero-length string is returned.

**Parameters:**

`count` - the number of repetitions.

`c` - the character to be repeated.

**Returns:**

a string containing `count` repetitions of `c`.

does not throw an exception!!!



YORK UNIVERSITÉ UNIVERSITY

# Responsibilities

Case 1: the implementer **does not throw an exception**
e.g., `ToolBox.repeat(10, 'c');`

| Outcome | Who is the Culprit? |
|---|---|
| No crash<br>[component behaviour is *as specified*] | **client met the preconditions?**<br>n/a – correct outcome, no one is a culprit<br>**client did not meet the preconditions?**<br>just dumb luck – client should not expect this |
| No crash<br>[component behaviour is *NOT as specified*] | **client met the preconditions?**<br>implementer's fault<br>**client did not meet the preconditions?**<br>client's fault* |
| Crash!<br>(component behaviour is *NOT as specified*; the implementer threw an exception) | **client met the preconditions?**<br>implementer's fault<br>**client did not meet the preconditions?**<br>client's fault* |

17 *note: the implementer did not meet its responsibilities, but it is excused because of the client

YORK UNIVERSITÉ UNIVERSITY

# Responsibilities

Case 2: the implementer **may potentially throw an exception**
e.g., `ToolBox.getBMI(200,"6'1");`

| Outcome | Who is the Culprit? |
|---|---|
| No crash<br>[component behaviour is *as specified*] | **client met the preconditions?**<br>n/a – correct outcome, no one is a culprit<br>**client did not meet the preconditions?**<br>just dumb luck – client should not expect this |
| **Crash!**<br>**[component behaviour is *as specified*]** | **client met the preconditions?**<br>n/a – correct outcome, no one is a culprit<br>**client did not meet the preconditions?**<br>just dumb luck – client should not expect this |
| No crash<br>[component behaviour is *NOT as specified*] | **client met the preconditions?**<br>implementer's fault<br>**client did not meet the preconditions?**<br>client's fault* |
| Crash!<br>[component behaviour is *NOT as specified*] | **client met the preconditions?**<br>implementer's fault<br>**client did not meet the preconditions?**<br>client's fault* |

18

YORK UNIVERSITÉ UNIVERSITY

# Contracts in Java

**Methods in the Java standard library specify their pre and post as follows:**

- **pre is always true unless stated otherwise**
- **post is specified under Returns and Throws**

### *Generic-style contract*

`double squareRoot(double x)`

Returns the square root of the given argument.

**Parameters**

`x` – an argument

**Pre**

`true`

**Returns**

The positive square root of x

**Post**

The return as stated under "Returns".

### *Java-style contract*

`double squareRoot(double x)`
Returns the square root of the given argument.

**Parameters:**
`x` – an argument.
**Returns:**
the positive square root of `x`.
**Throws:**
an exception if `x < 0`.




**RQ2.28** Can the main class be the culprit when a runtime error occurs in a component?

After all of the background discussion, now answer!!!



YORK UNIVERSITÉ UNIVERSITY

## Though Questions

Give an example of a scenario where a false precondition does not cause the program to crash.

Why is this scenario dangerous?



YORK UNIVERSITÉ UNIVERSITY

RQ2.30-2.31

**What are the key concepts about Software Engineering?**

- it is study of software projects and their progress; guidelines for creating sw
  - mitigating risk
  - enhancing readability
  - defining software correctness
    - *long-term objective: to validate correctness automatically*
- Software: Development vs Production
  - development: used in controlled environment (developers, testers, QA)
    - revenue possibly down the line, problems identified and fixed
  - production: sw goes “live” (released to public, paying customers)
    - revenue now, problems are costly (money, reputation)
- “Risk Mitigation by Early Exposure” is a key principle
  - we cannot avoid risk altogether, but we can manage it
  - If there is a risk in doing something during software development, confront it as early as possible.





YORK UNIVERSITÉ UNIVERSITY

RQ2.30-2.31

**Give a concrete example of RMBEE.**

- converting the type of a value is risky
  - e.g., converting a `double` to an `int` may result in data loss
  - strategy for this: check for (and disallow) this at compilation time rather than at runtime
- Java mitigates this risk by checking type compatibility at compile time (which refuse to compile if there is a violation) rather than leaving it for run time
- the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error.
- The risk of truncating the real value is exposed early.





YORK UNIVERSITÉ UNIVERSITY

**RQ** What are magic numbers? Should they be avoided?



YORK UNIVERSITÉ UNIVERSITY

**RQ** If magic numbers are to be avoided, why shouldn't I just use a variable?



YORK UNIVERSITÉ UNIVERSITY

**RQ2.32**
Explain how the `final` keyword enables us to eliminate magic numbers.



YORK UNIVERSITÉ UNIVERSITY

# 2.3.2 Handling Constants

## Replace all magic numbers (literals) in your program with finals.

**Instead of:**

```
width = width / 12;
```

**Write:**

```
final int INCH_PER_FOOT = 12;
width = width / INCH_PER_FOOT;
```

### Advantages of finals versus literals:

The literal has a name and, thus, is self-documenting

The compiler prevents you from inadvertently change its value




RQ2.32

**Explain how the final keyword enables us to eliminate magic numbers?**

- literals embedded in expressions or as parameters are magic numbers
  - you used a literal because:
    - some particular value is needed
    - that particular value is pre-defined and unchanging
- magic numbers should be avoided
- use variables instead of magic numbers? somewhat dangerous! some other person can come along and futz with your code! can introduce logic errors.
- how do you enforce that the value is predefined and not able to change?
  - use the keyword `final` before the declaration.