Day 8

COMP1006/1406
Summer 2016

M. Jason Hinek
Carleton University

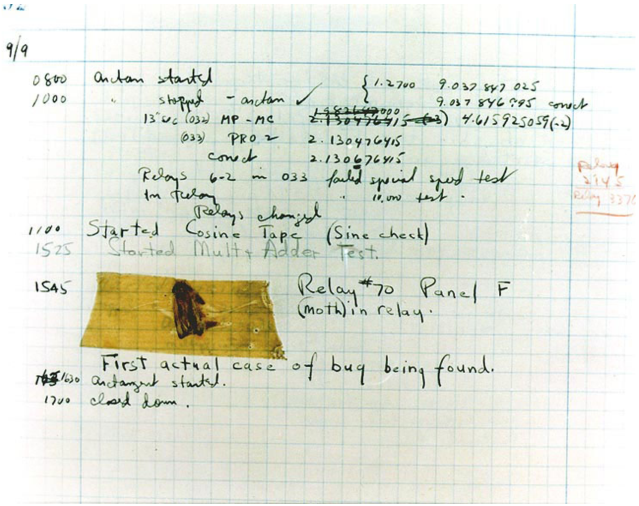
today's agenda

- assignments
    - Assignment 4 is out and due on Tuesday
- Bugs and Exception handling

Bugs...

often use the word `bug` when there is a problem with our program

Bugs...



9/9

0800 Antan started

1000 " stopped - antan ✓ {1.2700 9.037 847 025
{ 9.037 846 995 conect

13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

conect 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 11.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F (moth) in relay.

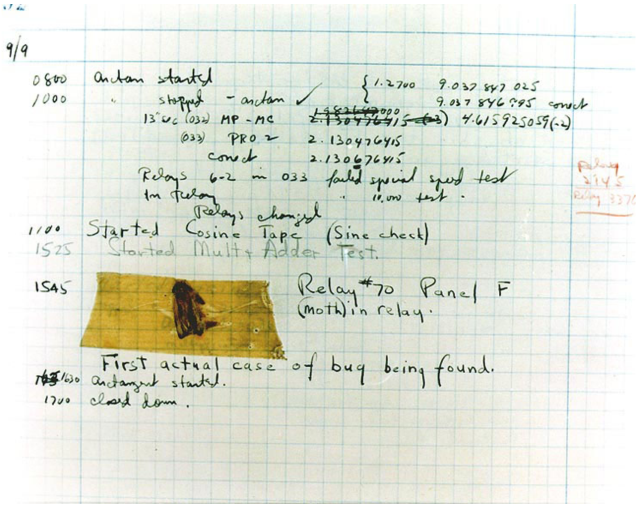
First actual case of bug being found.

~~1630~~ 1630 antangent started.

1700 closed down.

Relay 2145
Relay 3370

Bugs...



9/9

0800 Antan started

1000 " stopped - antan ✓ {1.2700 9.037 847 025

9.037 846 995 conct

13" uc (032) MP - MC ~~2.130476415 (-3)~~ 1.982147000 4.615925059(-2)

(033) PRO 2 2.130476415

conct 2.130676415

Relays 6-2 in 033 failed special speed test

In Relay " 11,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult+ Adder Test.

1545 Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

~~1600~~ 1630 antangent started.

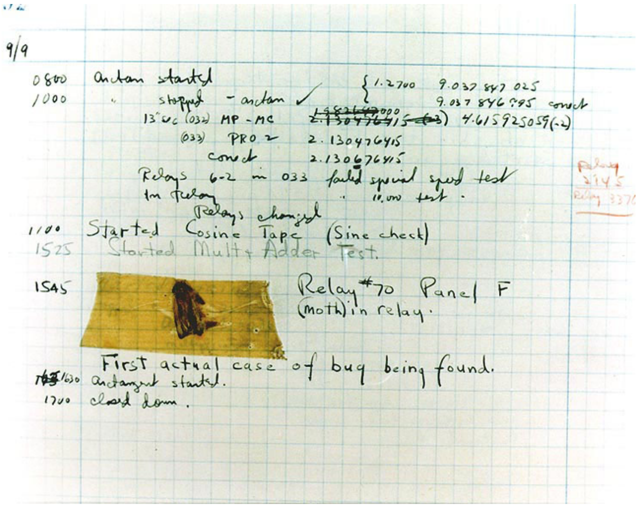
1700 closed down.

Relay 2145

Relay 3370

- bug is an error in our code (or hardware)

Bugs...



9/9

0800 antan started
1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In telay " 11,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~1600~~ 1630 antangent started.
1700 closed down.

Relay 2145
Relay 3370

- bug is an error in our code (or hardware)
- debugging is a methodical process of finding and reducing the number of bugs

Bugs...

essentially 3 types of bugs/errors

Bugs...

essentially 3 types of bugs/errors

- `compile errors`

- `runtime errors`

- `logical errors`

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
- `runtime errors`
- `logical errors`

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
- `runtime errors`
  - cannot be determined at compile time
  - sometimes require re-design of code
- `logical errors`

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
- `runtime errors`
  - cannot be determined at compile time
  - sometimes require re-design of code
- `logical errors`
  - non-syntax mistakes in your code
  - Java cannot detect or explain these errors
  - may be very hard to find...

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
  - `these are your fault`
- `runtime errors`
  - cannot be determined at compile time
  - sometimes require re-design of code
- `logical errors`
  - non-syntax mistakes in your code
  - Java cannot detect or explain these errors
  - may be very hard to find...

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
  - `these are your fault`
- `runtime errors`
  - cannot be determined at compile time
  - sometimes require re-design of code
  - `these may or may not be your fault`
- `logical errors`
  - non-syntax mistakes in your code
  - Java cannot detect or explain these errors
  - may be very hard to find...

Bugs...

essentially 3 types of bugs/errors

- `compile errors`
  - program does not compile but compiler tells you why
  - syntax errors, type mismatch errors, ...
  - `these are your fault`
- `runtime errors`
  - cannot be determined at compile time
  - sometimes require re-design of code
  - `these may or may not be your fault`
- `logical errors`
  - non-syntax mistakes in your code
  - Java cannot detect or explain these errors
  - may be very hard to find...
  - `these are definitely your fault`

Bugs...

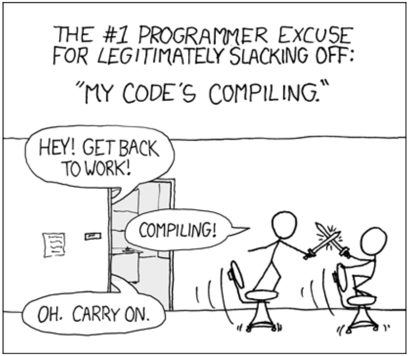
- first we need to discover that a bug exists

Bugs...

- first we need to discover that a bug exists
    - compile errors are found for free*

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*



xkcd - 303

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur

Bugs...

- first we need to discover that a bug exists
    - compile errors are found for free*
    - test, test, test, ...
    - `println` is your friend
- next we need to find in your code where they occur
    - JVM might tell you where code crashed (runtime error)

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free⋆
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur
  - JVM might tell you where code crashed (runtime error)
  - observable error may have been caused elsewhere though...

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur
  - JVM might tell you where code crashed (runtime error)
  - observable error may have been caused elsewhere though...
  - play computer! backtrack...

Bugs...

- first we need to discover that a bug exists
    - compile errors are found for free*
    - test, test, test, ...
    - `println` is your friend
- next we need to find in your code where they occur
    - JVM might tell you where code crashed (runtime error)
    - observable error may have been caused elsewhere though...
    - play computer! backtrack...
    - might be easy or hard

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur
  - JVM might tell you where code crashed (runtime error)
  - observable error may have been caused elsewhere though...
  - play computer! backtrack...
  - might be easy or hard
- next we fix the bug

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur
  - JVM might tell you where code crashed (runtime error)
  - observable error may have been caused elsewhere though...
  - play computer! backtrack...
  - might be easy or hard
- next we fix the bug
  - if we know exactly where the bugs originates, often easy to fix

Bugs...

- first we need to discover that a bug exists
  - compile errors are found for free*
  - test, test, test, ...
  - `println` is your friend
- next we need to find in your code where they occur
  - JVM might tell you where code crashed (runtime error)
  - observable error may have been caused elsewhere though...
  - play computer! backtrack...
  - might be easy or hard
- next we fix the bug
  - if we know exactly where the bugs originates, often easy to fix
  - serious bugs may require rethinking your code/objects

Bugs...

part of the problem is that we don't live in a perfect world...

- people do not follow API specifications (preconditions)
- people do weird things with your code...
- files get corrupted

you have no control over how people use your code
you have no control over the universe your code is running in

- need to write `robust` code
- **robustness** is the ability of the code to cope with errors during execution and cope with erroneous input (from wiki)

Bugs...

we could write code to check for all possibilities in our code

- **error-checking** is code added to your code to look for bad data
- **error-handling** is what you do when bad data is found

  $\Longrightarrow$ everything needs to be an `Object`

in Java we use `Exceptions`...

- an **Exception** is an error that occurs in your code
- **Exception Handling** is what you do when an exception is found

goal of using exceptions

- you want your program to die gracefully
- you want to program to recover from bad data if possible

Exceptions...

in Java, exceptions are `objects`

- the JVM automatically does this, or
- you explicitly do this in your code, or
- another method will explicitly do this

in Java, thrown exceptions are always `caught`

- explicitly caught by your code, or
- delegated to someone else to be caught, or
- caught by the JVM if everyone delegates

Exceptions...

in Java, exceptions are `objects`

- the JVM automatically does this, or
- you explicitly do this in your code, or
- another method will explicitly do this

in Java, thrown exceptions are always `caught`

- explicitly caught by your code, or (graceful)
- delegated to someone else to be caught, or (potentially graceful)
- caught by the JVM if everyone delegates (ugly)

Exceptions...

in Java, exceptions are `objects`

Error Class

- unrecoverable errors
  - `java.lang.StackOverflowError`
  - `java.lang.OutOfMemoryError`
- you do not generally catch these yourself
  (you fix your code so it doesn't happen again!)

Exceptions...

in Java, exceptions are `objects`

Exception Class

- less severe errors (we might be able to recover from these)
- there are **checked** and **unchecked** exceptions

Exceptions...

in Java, exceptions are `objects`

Exception Class

- less severe errors (we might be able to recover from these)
- there are **checked** and **unchecked** exceptions

- **checked** exceptions (`Exception Class`)
    - compiler checks that these are explicitly caught
      (if there is a throw there must be a catch)
    - `IllegalAccessException` (try to violate access modifier)
      `FileNotFoundException` (file is not found)

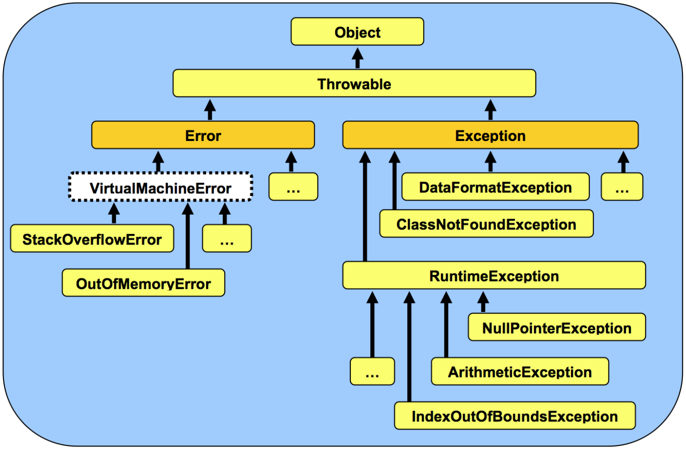
Exceptions...

in Java, exceptions are `objects`

Exception Class

- less severe errors (we might be able to recover from these)
- there are **checked** and **unchecked** exceptions
- **checked** exceptions (`Exception Class`)
  - compiler checks that these are explicitly caught
    (if there is a throw there must be a catch)
  - `IllegalAccessException` (try to violate access modifier)
    `FileNotFoundException` (file is not found)
- **unchecked** exceptions (`RuntimeException` subclass)
  - compiler does not check for a catch
    (typically left for JVM to crash program)
  - you should rethink your code to avoid these...
  - `ArtithmeticException` (divide by zero for example)
    `NullpointerException`
    `IndexOutOfBoundsException`

Exceptions...



Object
Throwable
Error
Exception
VirtualMachineError
...
DataFormatException
...
StackOverflowError
...
ClassNotFoundException
OutOfMemoryError
RuntimeException
NullPointerException
...
ArithmeticException
IndexOutOfBoundsException

let’s take a break...
for 1.2 minutes

Exceptions...

new Java keywords

- `throws`
  - used in method declaration
  - says that this method is delegating this exception
- `try`/`catch`/`finally`
  - used to execute code and handle exceptions thrown
  - "try" to execute this code...
    "catch" any thrown exceptions and handle them
    "finally" execute some code after everything is done
- `throw`
  - explicitly throw an exception

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might
  - explicitly throw a java.io.FileNotFoundException object

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might
  - explicitly throw a java.io.FileNotFoundException object
    or
    calls a method that might

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might
  - explicitly throw a java.io.FileNotFoundException object
    or
    calls a method that might
    - explicitly throw a java.io.FileNotFoundException object

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might
  - explicitly throw a java.io.FileNotFoundException object
    or
    calls a method that might
    - explicitly throw a java.io.FileNotFoundException object
      or
      calls a method that might

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

tells the compiler that this method might

- explicitly throw a java.io.FileNotFoundException object
  or
  calls a method that might
  - explicitly throw a java.io.FileNotFoundException object
    or
    calls a method that might
    - explicitly throw a java.io.FileNotFoundException object
      or
      calls a method that might
        ...
          ...
            ...
              explicitly throw a java.io.FileNotFoundException object

Throws

```
void openFile(String fname) throws java.io.FileNotFoundException{
...
}
```

let’s look at an example

Try/Catch

- use `throws` to delegate exception handling
- use `try` and `catch` to handle the exception yourself

Try/Catch

- use `throws` to delegate exception handling
- use `try` and `catch` to handle the exception yourself

```
try{
  block of code to try
}
catch(Exception e){
  block of code
  to execute if exception is caught
}
```

Try/Catch

- use throws to delegate exception handling
- use try and catch to handle the exception yourself

```
try{
  block of code to try
}
catch(Exception e){
  block of code
  to execute if exception is caught
}
catch(RuntimeException re){
  block of code
  to execute if exception is caught
}
```

Note: this won't compile

Try/Catch

- use `throws` to delegate exception handling
- use `try` and `catch` to handle the exception yourself

let's look at the same example...

Try/Catch

order matters! class hierarchy matters!

$$\begin{tikzcd}
\boxed{\texttt{java.lang.Object}} \\
\boxed{\texttt{java.lang.Throwable}} \arrow[u] \\
\boxed{\texttt{java.lang.Exception}} \arrow[u] \\
\boxed{\texttt{java.io.IOException}} \arrow[u] \\
\boxed{\texttt{java.io.FileNotFoundException}} \arrow[u]
\end{tikzcd}$$

Try/Catch

order matters! class hierarchy matters!

$$\begin{tikzcd}
\boxed{\texttt{Object}} \\
\boxed{\texttt{Throwable}} \arrow[u] \\
\boxed{\texttt{Exception}} \arrow[u] \\
\boxed{\texttt{java.io.IOException}} \arrow[u] \\
\boxed{\texttt{java.io.FileNotFoundException}} \arrow[u]
\end{tikzcd}$$

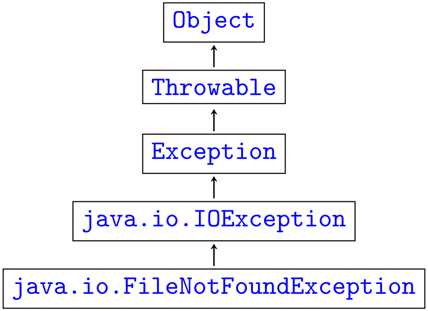
Try/Catch

```
try{
   block of code to try
}
catch(IOException ioe){
   block of code to
   execute if exception
   thrown
}
catch(RuntimeException re){
   block of code to
   execute if exception
   thrown
}
catch(Exception e){
   block of code to
   execute if exception
   thrown
}
```

```
{
   block of code to try
}
if (IOException is thrown){
   block of code to
   execute if exception
   thrown
}
else if (RuntimeException){
   block of code to
   execute if exception
   thrown
}
else if (Exception){
   block of code to
   execute if exception
   thrown
}
```

Try/Catch

order matters! class hierarchy matters!


$$\begin{tikzcd}
\texttt{Object} \\
\texttt{Throwable} \arrow[u] \\
\texttt{Exception} \arrow[u] \\
\texttt{java.io.IOException} \arrow[u] \\
\texttt{java.io.FileNotFoundException} \arrow[u]
\end{tikzcd}$$

Try/Catch/Finaly

- `try` executes a block of code
- if exception thrown, `catch` it and handle it
- after try and possibly catch code executes, we `finally` execute some finishing code

Try/Catch/Finaly

- `try` executes a block of code
- if exception thrown, `catch` it and handle it
- after try and possibly catch code executes, we `finally` execute some finishing code

```
try{
  block of code to try
}
catch(Throwable e){
  block of code
  to execute if exception is caught
}
```

Try/Catch/Finaly

- `try` executes a block of code
- if exception thrown, `catch` it and handle it
- after try and possibly catch code executes, we `finally` execute some finishing code

```
try{
  block of code to try
}
catch(Throwable e){
  block of code
  to execute if exception is caught
}
finally{                                        // like a funny else
  block of code to execute
  REGARDLESS of what happens above
}
```

Throw

`throw` is the mechanism to explicitly throw an exception.

Throw

`throw` is the mechanism to explicitly throw an exception.

- `throw new Exception("something bad here...");`
  creates a new Exception object and throws it

Throw

`throw` is the mechanism to explicitly throw an exception.

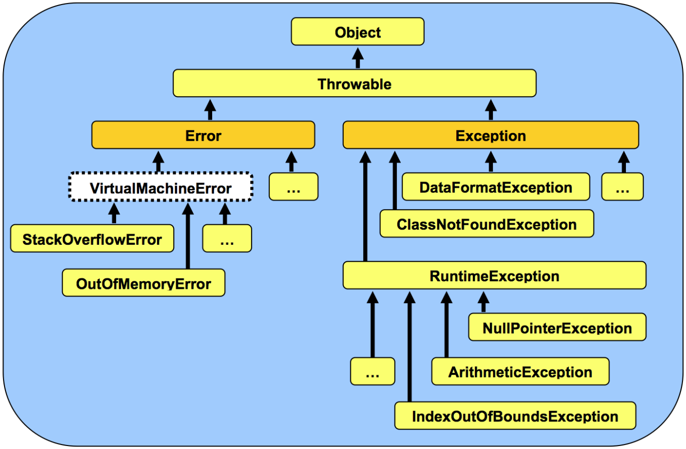
- `throw new Exception("something bad here...");`
  creates a new Exception object and throws it
- `throw e;`
  throws an existing exception
  (`e` must be a `Throwable` object)

Making Your Own Exceptions

you create your own custom exceptions by `extending` an appropriate `Thowable` class

Making Your Own Exceptions

you create your own custom exceptions by extending an appropriate Thowable class



Making Your Own Exceptions

you create your own custom exceptions by `extending` an appropriate `Thowable` class

```
public class BadCardRankException extends Exception {

    public BadCardRankException(){
        super("Rank of card is invalid");
    }
  }
```

Throwable

the `Throwable` class is the root of all exceptions in Java

- `toString()`
  - returns short description of this object
- `getMessage()`
  - returns the message as input with the constructor
- `printStackTrace()`
  - prints the stack trace to standard error