Wednesday, November 18

Special Session – Narration in Writing

Hello, everyone! We do not meet in the classroom today. This presentation file will guide you to complete the lecture in self-paced mode. I will be available on zoom for lecture explanation (see next slide for zoom availability.) I've added a bonus assignment based on this lecture. You may find the bonus questions in the following slides.

Today's Lecture Mode

- This is a silent/annotated lecture.
  - Annotations:

I've put whatever I'd talk during the face-to-face meeting in writing and placed in curve-shaped blocks or highlighted.

- If you wish to speak to me or have the lecture via zoom, I'll be available via zoom (zoom id: 9098694052) 10-11:50am and your regular class meeting time.
  - Appreciate it if you could drop me an email before joining the zoom as I may be on another zoom with student appointments etc.

With your strong background in Java, I am confident that you're familiar with exception handling. This lecture focuses on comparisons of language constructs in supporting exception handling.

Lecture 10: Exception Handling

Basic concepts
and
summary of features

Basic Concepts

- Exception vs. Errors
  - Error: unrecoverable, e.g. OutOfMemoryError
  - Exception: unexpected event that can be handled gracefully, e.g. FileNotFoundException

– Are exceptions all bad? (not necessary)

We can apply exception handling constructs to check and prevent index bounds or zero-division case (similar to if-stmt does in these cases but in better structured way or with better readability.)

Basic Concepts

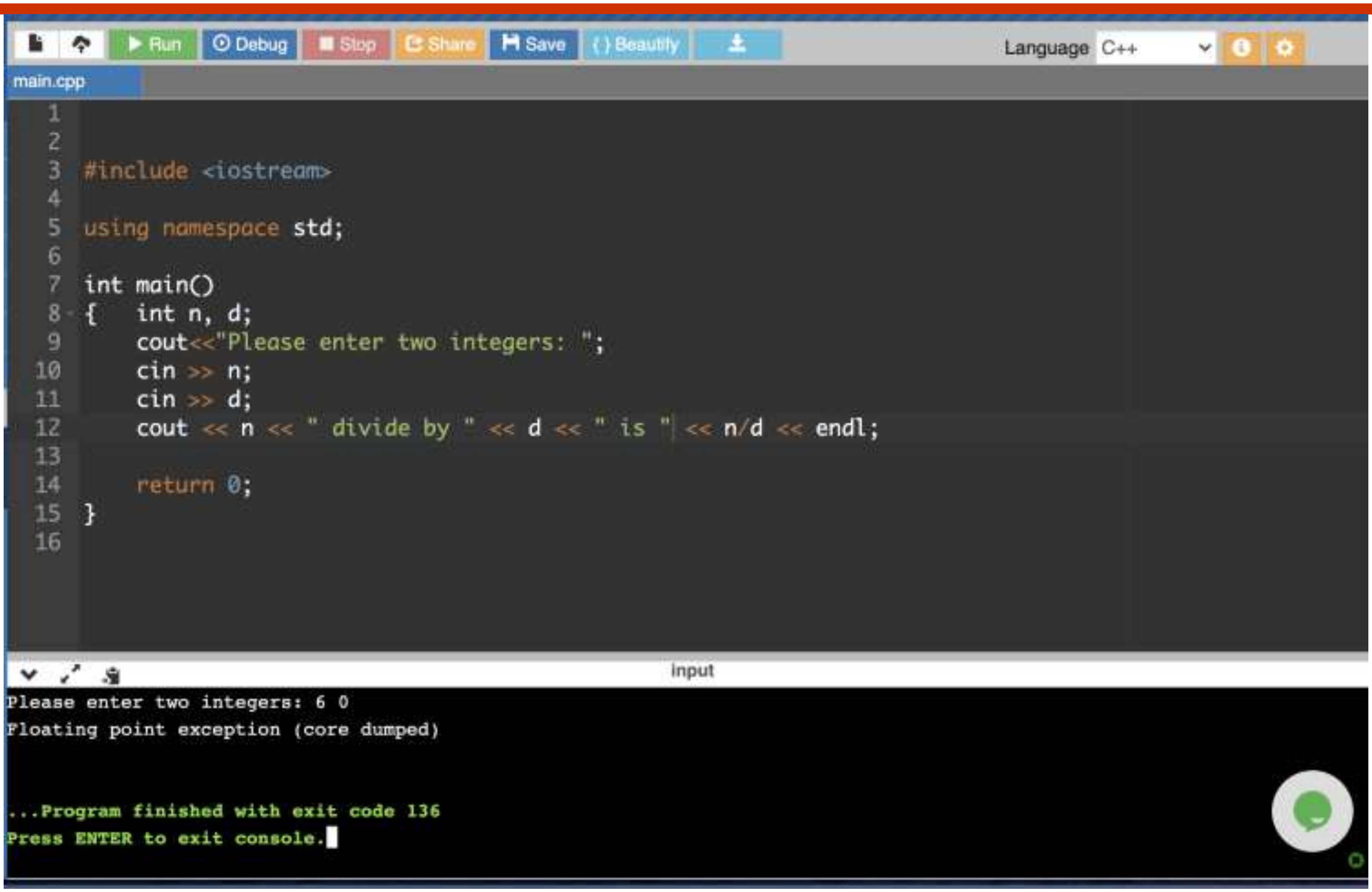
- Exception handling vs. Event handling
  - Exceptions: entities that are generated by a condition or event and that might require additional information or investigation.
  - Events: completion or failure of some actions, indicating a user is interacting with a program.
    - Example: an event can be a mouse click or pressing a key on the keyboard.
    - Event handler: what to do when an event occurs

Event handling is a newer (exciting, useful) feature, but we won't cover it in this class.

How did programmers survive at early days when there wasn't any language supported exception handling mechanism?

(The following slides are based on my personal experience.)

Scenario 1: When an exception occurs, control goes to OS where error msg displayed and program terminated



```cpp


#include <iostream>

using namespace std;

int main()
{   int n, d;
    cout<<"Please enter two integers: ";
    cin >> n;
    cin >> d;
    cout << n << " divide by " << d << " is " << n/d << endl;

    return 0;
}

```

```
Please enter two integers: 6 0
Floating point exception (core dumped)


...Program finished with exit code 136
Press ENTER to exit console.
```

C++: Divide by zero exception(core dumped was a puzzled error message.)

Scenario 2: Programmer adds protection

```
if (d == 0 )
            cout << “divide by zero error\n”;
else
            cout << “quotient is: “ << n/d << “\n”;
cout << “now we continue”;
```

Compare with previous case, now in case d is 0, a message is displayed, and the program will continue to execute.

However, in this case the coding writability suffers (as well as readability) due to adding a lot of if statements to prevent exceptional cases.

Scenario 3: Aided with built-in function

```
//fin: a file stream opened successfully
while (!fin.eof()) {  //while not end of file
        data = fin.read();
        ...
}
```

!eof() protects read() from hitting an exception when no more data to read. Similar strategy could be used for preventing other exceptions such as file not exist when opening.

Without built-in functions such as eof(), programmers are not able to code the protection. – now these features are automatically built into exception handling mechanisms.

Scenario 4: errors occurred in subprograms

```
double my_div (double n, double d, int & flag) {
          if (d == 0) {flag = 1; return 0.0;  }
          else { flag = 0; return n/d;}
}

int main () {
 int error;
 …
 result = my_div (5, 0, error);    //a status flag from the subprogram
 if (error) { return 1; } else { … display result …}
 …
}
```

A pass-by-reference parameter is used to keep track of whether an error every occurred or not.

Now, almost every newer language comes with language supported exception handling construct.

Language supported Exception handling mechanism became popular in 1980s. Java introduced it in its early version Java 1.0

Bonus Assignment

Question 1

True or False: If there is no exception handling mechanism in a language (say, there isn’t a try-catch block in Java), we have no way to protect our program from errors such as divide-by-zero or index out of bound.

Basic Concepts

- What?
  - What is an exception?
  - What is an exception handler?
- Why?
  - Reliability (pros)
  - (cons) ?
- How?
  - How to raise an exception?
  - How to catch an exception?
    - What if not caught?
  - How to handle an exception?
    - Where to go once an exception is handled?

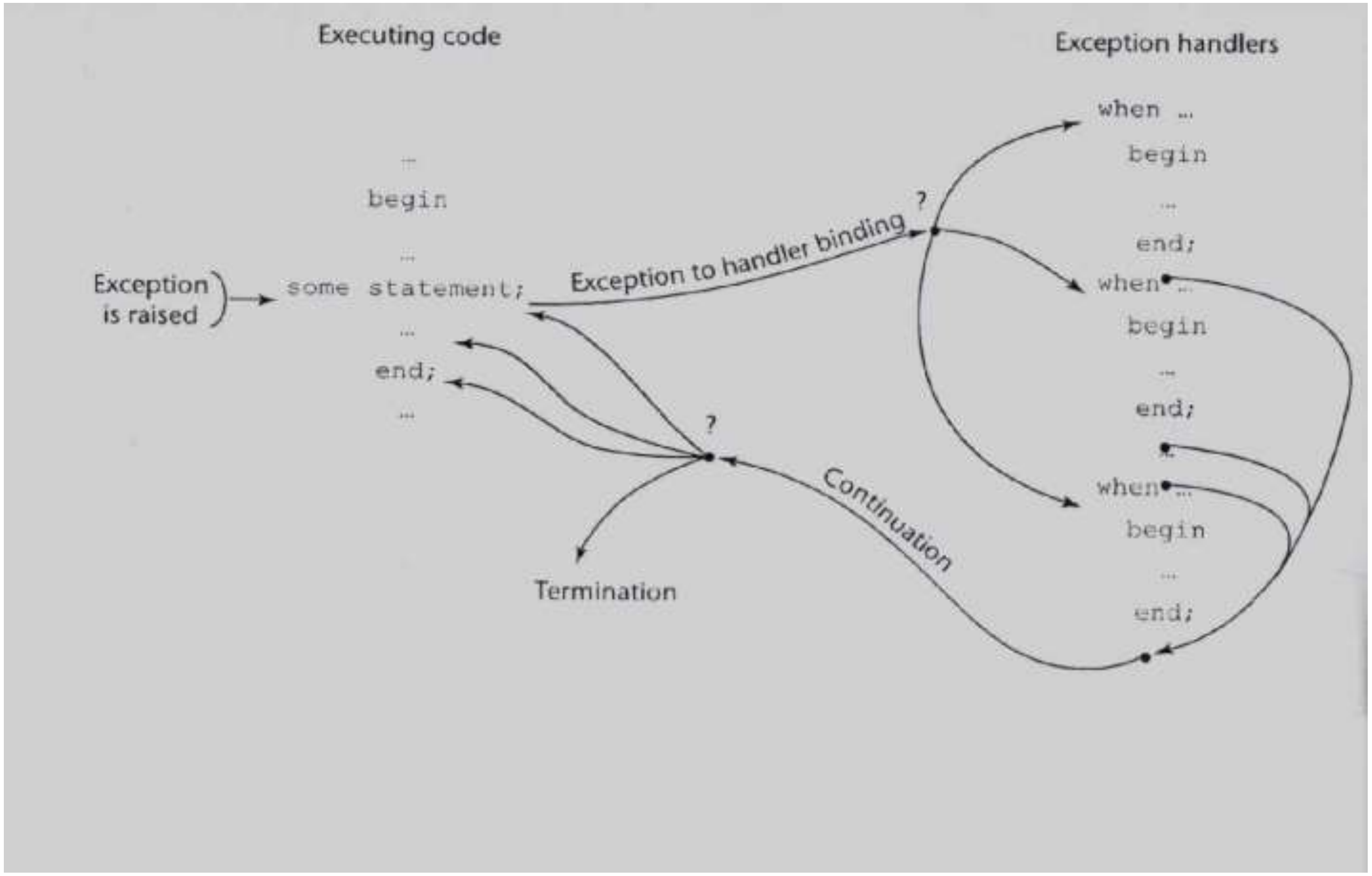
Terms and fundamental knowledge on exception handling, applying to any programming language.

Exception Handling: Advantages

- Traditionally error detection code is tedious to write
- Exception handling encourages programmers to consider many different possible errors
- Exception propagation allows a high level of reuse of exception handling code

Advantages of exception handling mechanism:
improved writability (including reuse)
improved reliability

Exception Handling Control Flow



Bonus

Question 2

Why we introduce exception handling constructs to programming languages?

A. To enhance program’s readability
B. To enhance program’s reliability
C. To enhance program’s portability
D. To improve the run-time efficiency

Question 3

True or False

(1) Most of the contemporary languages now included mechanisms for exception handling.
(2) One reason that some languages do not include exception handling is the complexity it adds to the language.

Early Language Support: Ada

```
declare
   A : Matrix (1 .. M, 1 .. N);
 begin
   for I in 1 .. M loop
      for J in 1 .. N loop
          begin
             Get (A(I,J));
          exception
             when Data_Error =>
                Put ("Ill-formed matrix element");
                A(I,J) := 0.0;
          end;
       end loop;
   end loop;
 exception
   when End_Error =>
      Put ("Matrix element(s) missing");
 end;
```

Ada is an early language with strong support to exception handling. Usage:
exception => when

Ada vs. C++ vs. Java

https://learn.adacore.com/courses/Ada_For_The_CPP_Java_Developer/chapters/10_Exceptions.html

For your reference only.

Exception Handling in C++

- Added to C++ in 1990
  - Design is based on that of CLU, Ada, and ML
- Exception Handlers Form:

```
try {
-- code that is expected to raise an exception
}
catch (formal parameter) {
-- handler code
}
...
catch (formal parameter) {
-- handler code
}
```

Ada’s except => when vs. C++’s try-catch, which design you like?

```
double division(int a, int b) {
  if( b == 0 ) {
    throw ”Div by zero";
  }
  if (a == 0) {
    throw 0.0;
  return (a/b);
}
```

C++:
(1) any predefined exceptions ?
(2) throw exception, what type of values can be thrown?

```
int main () {
   …
   cin >> a;
   cin >> b;

  try {
    z = division(x, y);
    cout << z << endl;
  }
  catch (const char* msg) {
   cerr << msg << endl;
  }
  catch (double d) {
   cerr << “not like” << d << ”as result”;
  }
  catch (…) { cerr << “other errors”; }
              //e,g. error in cin >> a;
  return 0;
}
```

Control Flow

- If a throw statement is executed, control goes to the catch block
  - If a handler is matched, after a handler completes its execution, control flows to the first statement after the last handler in the sequence of handlers;
  - If no handler is matched, the exception is unhandled
- Unhandled exceptions
  - An unhandled exception is propagated to the caller of the function in which it is raised
  - This propagation continues to the main function
  - If no handler is found, control goes to the OS (to call default exception handler provided by the OS)

Bonus

Question 4

The following function may have divide-by-zero errors during execution. However, within the function there is no exception handling code. What happens if y happens to be 0?

double divNum (double x, double y) {return x/y; }

A. The program immediately crashes at the execution of x/y.

B. The exception will be passed to the function that calls divNum().

C. The function divNum will print out an error message before switching control flow to OS.

D. Correct answer not found in given choices.

Control Flow: Python vs. C++

Python: (1) advocate the use of predefined exceptions; (2) "else" feature in exception handling

```
try :
        A
except myException :
        B
else :
        C
finally :
        D
```

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

else block: executed when no exception occurs
finally block: always executed (popular in many languages)

Default handler: Python vs. C++

Python:

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    except:
        print(“Catch’em all!”)
    else:
        print("result is", result)
    finally:
        print("finally clause")
```

C++:

```
try {
    z = division(x, y);
  }

  catch (…) { cerr << “other errors”; }

  return 0;
}
```

What is Java’s default handler?

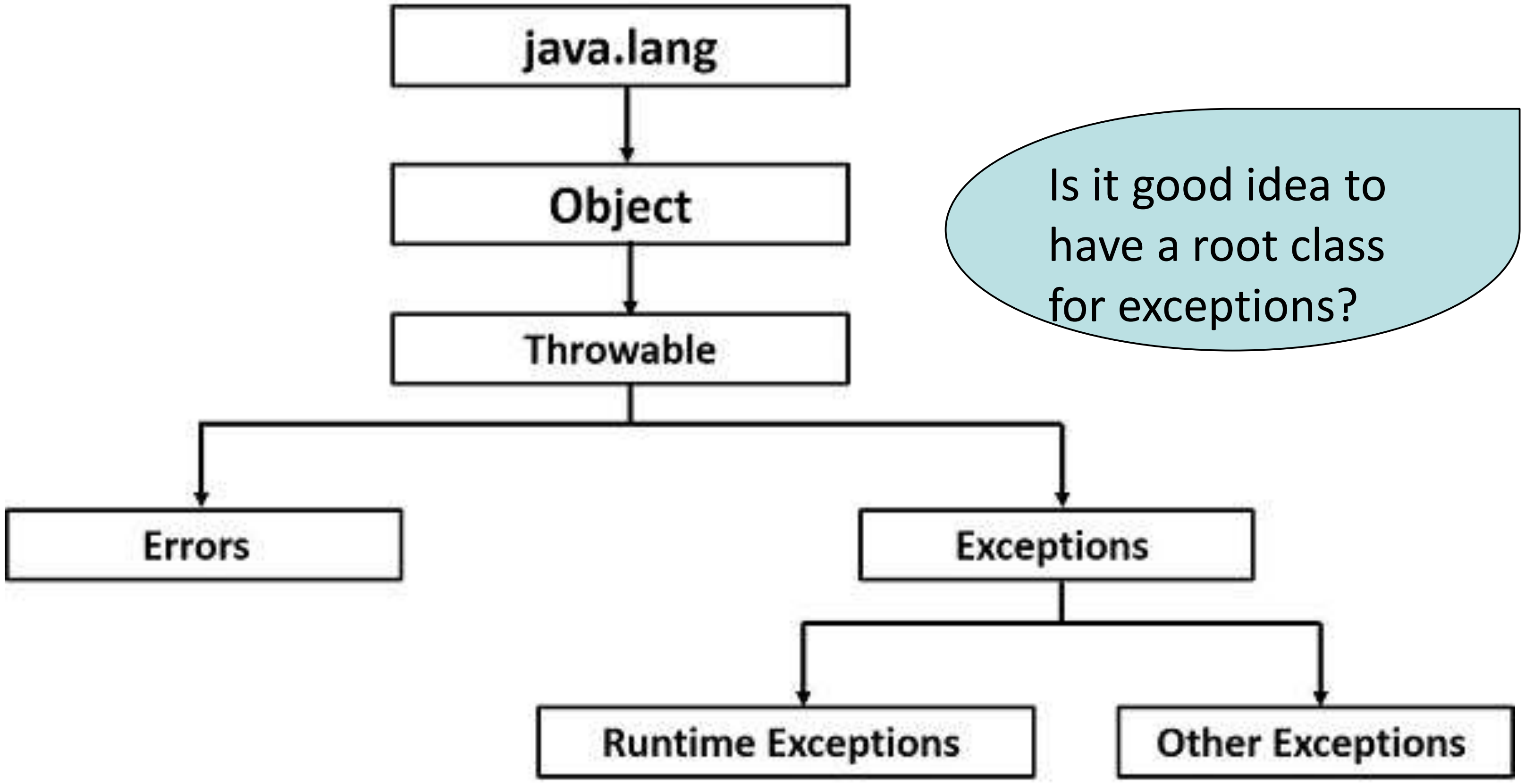
Bonus

Question 5

Which of the following is C++'s default exception handler (i.e. catch all handler)?

A. catch (Exception e)
B. catch ()
C. catch (…)
D. catch (_)
E. None of above

Java’s exception handling hierarchy



Is it good idea to have a root class for exceptions?

Reference

Checked and Unchecked Exceptions

- Unchecked Exceptions
  - Exceptions of class Error and RunTimeException and all of their descendants (all others are checked exceptions)
- Checked Exceptions
  - Checked exceptions are checked at compile-time.
    - It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.
- throw vs. throws (Example next slide)
  - A method that calls a method with a particular checked exception in its throws clause has three alternatives:
    - Catch and handle the exception
    - Catch the exception and re-throw an exception (could be a different exception) listed in its own throws clause
    - Declare the same exception in its throws clause and do not handle it

Checked and Unchecked Exceptions

- Unchecked Exceptions
  - Exceptions of class Error and RunTimeException and all of their descendants (all others are
- Checked Exceptions
  - Checked exceptions
    - It means if a meth should handle the declare the excep program will give
- throw vs. throws (
  - A method that calls a m exception in its throws cla
    - Catch and handle the exceptio
    - Catch the exception and re-throw an exception (could be a different exception) listed in its own throws clause
    - Declare the same exception in its throws clause and do not handle it

Are these good language features? Would you promote them (i.e. checked and unchecked exceptions and/or the usage of throw and throws) to future/new language design?

throw vs. throws

```
void myMethod() {
   try {
      //throwing arithmetic exception using throw
      throw new ArithmeticException("Something went wrong!!");
   }
   catch (Exception exp) {
      System.out.println("Error: "+exp.getMessage());
   }
}

class Example {
   public static void main(String args[]) throws IOException
   {                                    //IOException will not be handled in main()
      FileInputStream fis = null;
      fis = new FileInputStream("B:/myfile.txt");
       …
      fis.close();
   }
```

Bonus

Question 6

True or False

(1) In Java, index out of bound exception is a checked exception

(2) In Java, file not found exception is a unchecked exception.

Question 7

In your opinion, what are pros and cons of Java's design of checked/unchecked exceptions. (use your own words to describe.)

Bonus

Question 8

True or False

(1) In C++, a user-defined exception type must inherit from the Exception class.

(2) In Java, any user-defined exception type is a descent of the Throwable class.

The finally Clause

- Can appear at the end of a try construct

```
finally {
...
}
```

- Purpose: To specify code that is to be executed, regardless of what happens in the try construct

newer languages all support the finally feature

```
class TestFinallyBlock{
  public static void main(String args[]) {
  try {
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e) {
        System.out.println(e);
   }
   finally {
        System.out.println(”always executed");
   }
   System.out.println("rest of the code...");
  } //end of main
} //end of class
```

Bonus

Question 9

Any problem with the use of finally in the pseudo code below?

```
try {
    fn = … open a file
    … read in data from the file …
}
catch ( … file not found error… ) {
            … display the message
}
finally {
      …close the file …
}
```

Bonus

Question 10

In the following Python's code, what is the rule of the else?

```
if x > y :
    try:
        result = x // y
    except ZeroDivisionError:
        print("Sorry ! You are dividing by zero ")
    else:
        print("Yeah ! ")
```

Summary

Instead of looking back, I'd suggest that you look at the exception handling mechanism supported by your group language.

- History
  - Ada, one of the first languages with comprehensive exception handling features, provides extensive exception-handling facilities with a large set of built-in exceptions.
- C++ includes no predefined exceptions

My believe: Python well illustrated the convenience of using pre-defined exceptions

- Exceptions are bound to handlers by connecting the type of expression in the throw statement to that of the formal parameter of the catch function

Similar to the switch statement's matching

- Other features: the use of finally (and Python's else), default handler, user-defined exceptions, etc.

Well, we've finished Lecture 10: Exception Handling.

To earn extra credit, please complete the Bonus assignment on Canvas.

For the remaining time:

Take Quiz 5 – Coverage: OOPs (Lectures 8 and 9)

Work with group on Activity 5 and/or Capstone.

Thank you! Any question email me.

For zoom appointment please email me first before joining – zoom available today 10-11:50am as well as regular class meeting time.