CSC148H
Lecture 2

Exceptions
Object Oriented Analysis

Announcements

- Python Ramp-up session Monday May 26th BA3175, 1-5 pm.
  - Open to anyone enrolled in this course who feels that they can benefit from it
  - Attendance is completely optional

Last lecture...

- Talked about what computer science is
- Talked about ADTs
- Left off talking about the Priority Queue ADT
  - the last exercise in the lab this week was about using a priority queue

This lecture

- Exceptions
- Object Oriented Analysis & Design
- More examples with stacks and queues

The World Before Exceptions

- Not a pretty place
- Have to check return values from a function
- This is done as part of the regular flow of your program
- Can lead to spaghetti code that's hard to read and doesn't flow naturally

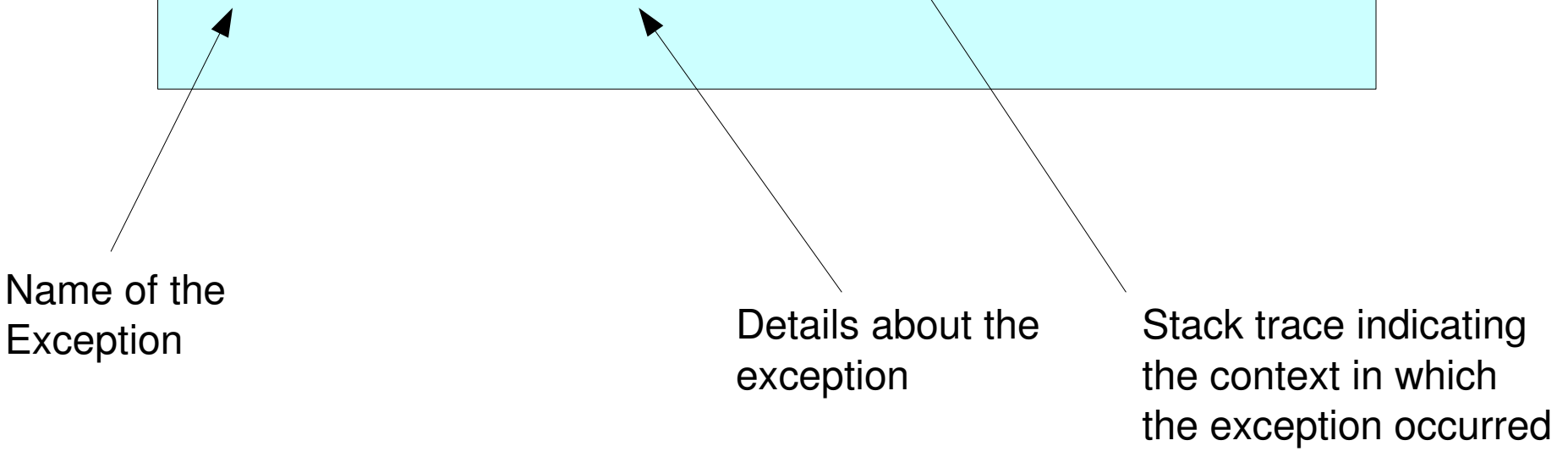
What are Exceptions?

- Exceptions allow you to structure code in a more natural way so that error handling and recovery is isolated from the regular flow of your program
- An exception is an object that indicates a problem
- An exception gets **raised** during program execution
  - Interrupt regular program flow
  - If you don't handle an exception, your program will

What are Exceptions?

- Example exceptions

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
```



Syntax Errors vs. Exceptions

- You may see the following:

```
>>> while True print 'Hello world'
  File "<stdin>", line 1, in ?
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

```
>>> while True print 'Hello world'
...
Traceback (most recent call last):
  File "<string>", line 1, in <string>
invalid syntax: <string>, line 1, pos 16
```

- This is not an exception, but rather a syntax error
  - It occurs while the interpreter is trying to parse your code and encounters code that is not proper Python

What are Exceptions?

- Lets create some examples in Wing!

Catching Exceptions

- **Try** to execute some code. It will run, unless an exception occurs
- If an exception occurs, you can catch and handle it

execute this code

handle the named exception if it happens

```
import sys
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError:
    print "Input/Output error."
except ValueError:
    print "Could not convert data to an integer."
```

Possible Execution Path #1

```
import sys

try:
    f = open('myfile.txt')
                                        No exceptions occur
    s = f.readline()

    i = int(s.strip())
except IOError:
    print "Input/Output error."

except ValueError:
    print "Could not convert data to an integer."

print "Hello World"        Execution resumes after the try clause
```

Possible Execution Path #2

```
import sys

try:
    f = open('myfile.txt')

    s = f.readline()

    i = int(s.strip())
except IOError:
    print "Input/Output error."

except ValueError:
    print "Could not convert data to an integer."

print "Hello World"
```

IOError exception occurs when opening a file

Exception is handled

After exception is handled, execution resumes after the try clause

Possible Execution Path #3

```
import sys

try:
    f = open('myfile.txt')

    s = f.readline()

    i = int(s.strip())
except IOError:
    print "Input/Output error."

except ValueError:
    print "Could not convert data to an integer."

print “Hello World”
```

Exception occurs trying to convert s to an integer

Exception is handled

After exception is handled, execution resumes after the try clause

Possible Execution Path #4

```
import sys

try:
    f = open('myfile.txt')

    s = f.readline()

    i = int(s.strip())
except ValueError:
    print "Could not convert data to an integer."

print “Hello World”
```

IOError exception occurs when opening a file

IOError is **not** handled and program crashes (** see example with nested try)

Catching Exceptions

- Lets see some examples in Wing!

Other nifty features of try/except clauses

- You can name multiple exceptions as a tuple in an except clause:
  - except (IOError, ValueError):
- You can use the “except” keyword without naming any exceptions – this means catch **any** exception. It must always appear after all other except clauses.

```
try:
   # do something
except (IOError, ValueError):
   # handle exception
except:
   # handle all other types of exceptions
```

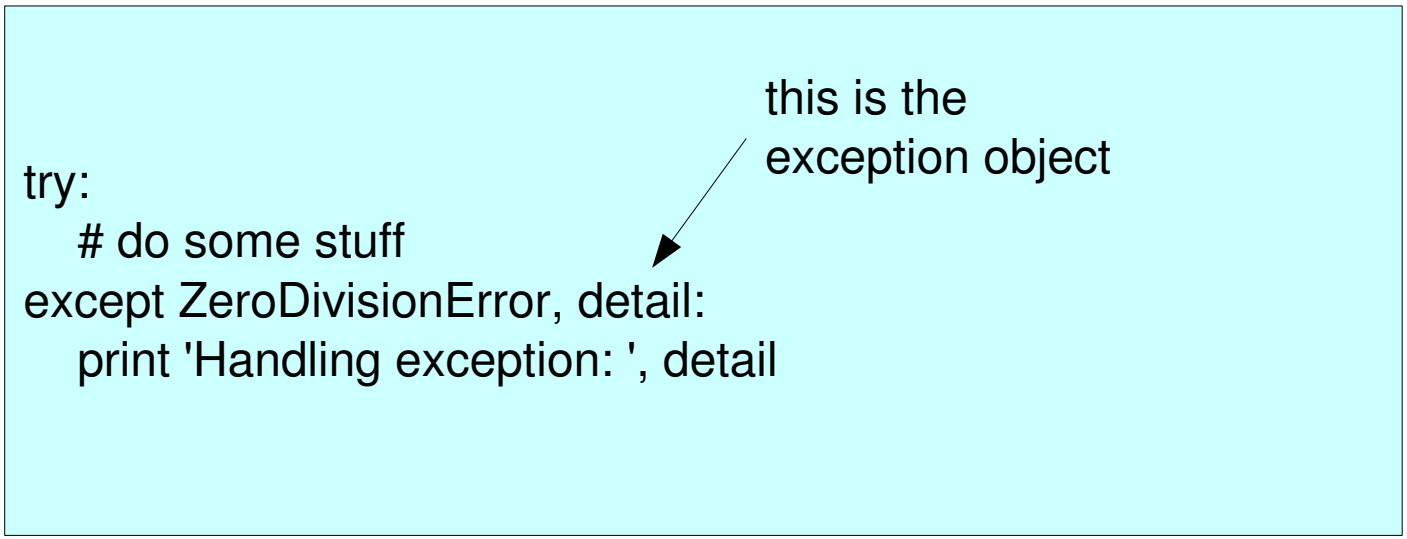
The finally clause

- If you want some code to be executed regardless of whether an exception occurs or not, and regardless of whether exception is handled or not, use a **finally** clause

```
try:
   #do some stuff
except IOError:
   #handle exception
finally:
   # do some cleanup
```

Exception variables

- You can get access to the exception object when handling an exception by adding it after the exception in the **except** statement:



Defining your own exceptions

- A user-defined exception has **Exception** as a superclass
- Can define it to have any instance variables or methods you want, just like a regular class
- Should include a __str__ method

```
class MyException(Exception):
        def __init__(self, value):
                self.value = value

        def __str__(self):
                return str(self.value)
```

Raising Exceptions

- Use the **raise** keyword

```
>>> raise NameError, 'my error message'
Traceback (most recent call last):
  File "<string>", line 1, in <string>
NameError: my error message
```

Re-raising Exceptions

- You can use the raise keyword to re-raise exceptions that you've already caught and handled

```
try:
   x = 1/0
except ZeroDivisionError, detail:
   print 'Caught the following error: ', detail
   raise
```

Examples

- Examples in Wing of
  - Exception variables
  - defining your own exceptions
  - raising/re-raising exceptions

More Reading About Exceptions

http://docs.python.org/tut/node10.html

Object Oriented Analysis and Design

"In OOA, we seek to model the world by identifying the classes and objects that form the vocabulary of the problem domain, and in OOD, we invent the abstractions and mechanisms that provide the behavior that this model requires." - Grady Booch. “Object-Oriented Design With Applications”

Object Oriented Analysis & Design - Terminology

OOA	Object-Oriented Analysis - analyzing your problem by decomposition into objects.
OOD	Object-Oriented Design – designing your code into objects
OOP	Object-Oriented Programming – programming using the concepts of object orientation
Domain	A formal boundary that defines a particular subject or area of interest
Abstraction	A clear separation between the abstract properties of a data type and the concrete details of its implementation
Class	A description of an object that contains data and methods; a new type
Object	An instance of a class
Attribute	A named property of an object capable of holding state (i.e., instance variables or data members)
Method	An operation on an object

http://www.fincher.org/tips/General/SoftwareEngineering/ObjectOrientedDesign.shtml

Real world vs. Domain Models

- The real world is impenetrably complex
  - e.g. You are made up of DNA, parents, history, etc.
  - For a particular problem, you can be abstracted (or **modeled)** as follows:
    - last name, first name, student number, course, final grade
- The object-oriented paradigm is one approach for simplifying the world
- First step: figure out the abstractions

OOA

- Find out what the client wants, then start turning it into programming speak
- Figure out what is irrelevant to your program
- Analyze your problem by decomposition into objects
- Look for nouns: those are candidate classes
- Look for verbs: those are candidates for methods

Example

- We are asked to build a system for keeping track of the time a client's workers spend working on customer projects. Projects have a set of tasks. Each task is assigned to a worker.

Analysis Step

- Analyze the written requirements to find the nouns and the verbs
  - Extract the nouns and make them classes
  - Determine attributes
  - Extract verbs and make them methods (be careful about which class owns them)

Example

- We are asked to build a system for keeping track of the **time** a client's **workers** spend working on **customer projects**. Projects have a set of tasks. Each **task** is assigned to a worker.
- Nouns: time, worker, project, customer, task
- Verbs: spend, have, assign; "have" and "assign" imply set a value.

Need to clarify the problem

- What information does the client want tracked for each noun?
- For example, is a worker's name enough, or does he/she have an employee id?
- In other words, we need to determine the attributes

Example – Determining Attributes

- Customer: has a list of projects (and probably contact information)
- Project: has a list of tasks (and probably a description)
- Task: has a time spent (and probably a description; possibly a project and a worker)
- Time: has a number of hours (and probably a start date/end date – or should that be part of the task?)
- Worker: has a list of tasks

Example

- Lots of decisions to make
- For example, a project has a set of tasks; does each task also know its project?
- Some decisions are arbitrary, and for complex requirements several different reasonable designs are possible
- There may be trade-offs that need to be considered when picking a design

Example

- Warning: your design may have to change as you program
- There are probably other details, but this is enough to start coding

Name	Customer	Project	Task	Worker	Time
attributes	- list of projects	-name -customer -list of tasks	-description -time spent	-name -task list	-start date -end date -hours
methods	- add project	- add task - remove task	-set time -add to time	-add task -remove task -spend time on task	-set hours

OOA – Another Example

- Your client has asked you to develop an address book application for storing various pieces of information about contacts. A contact has a name, and may be either a company or a person. Both kinds have an address and a main phone number. They may also have a secondary phone number. Contacts can be added, renamed, and removed; their address and phone number(s) can also change. It should be possible to search by name, address, and phone number.

Example

- How do we determine what the classes will be?
- How do we determine what the attributes will be?
- How do we determine what the methods will be?

Example

- Make a list of all the nouns and figure out which ones should be classes and which ones attributes.
- Figure out which attributes belong with which classes, recognizing that some may be attributes of other classes
- Determine the verbs and how those verbs affect the nouns – this will help you figure out what methods to create

OOA – Another Example

- Your client has asked you to develop an **address book** application for storing various pieces of information about **contacts**. A contact has a **name,** and may be either a **company** or a **person**. Both kinds have an **address** and a **main phone number**. They may also have a **secondary phone number**. Contacts can be added, renamed, and removed; their address and phone number(s) can also change. It should be possible to search by name, address, and phone number.

Nouns in the problem statement

- Address book: a collection of contacts
- Contact: has a name, address, phone #
- Name: just a string?
- Address: just a string?
- Main phone number: just a string?
- Secondary Phone Number: just a string?
- Company: marks whether contact is a company
- Person: marks whether contact is a person

Address book organization

- Address book
  - attributes: a collection of contacts
- Contact
  - attributes: name, address, main phone number, secondary phone number, company/person

Address Book Operations

- Address Book
  - operations: add contact, remove contact, find contact (by name/address/phone #)
- Contact
  - rename, set or change address, set or change main phone number, set or change secondary phone number.

Example

- Lets translate this into (the start of) a python program