CISC 3115 TY2

Custom Exceptions

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Error and error handling
    - Two approaches
  - Exception
  - The throwable class hierarchy
    - System errors and semantics
    - Runtime exceptions and semantics
    - Checked errors and semantics
  - Declaring, throwing, and catching exception
  - Exception, call stack, and stack trace, the finally clause, and rethrowing exceptions
- Custom exceptions

Review: Some Best Practice for Using Exceptions

- Exceptions are expensive, and are for exceptional conditions.
- Exceptions are not abnormal. Organize your code
  - Group exception handling code together
  - Throw exceptions earlier, but catch them later
- Exceptions are commonly used for diagnosing problems in the programs, be specific!
- Be specific when throw exceptions (Throw specific exceptions)
  - Use the exception classes in the API whenever possible.
  - Define *custom exception* classes if the predefined classes are not sufficient.

Custom Exceptions, i.e., Defining Your Own Exceptions?

- Before we proceed, follow the best practice
  - Use the exception classes in the API whenever possible.
  - Define custom exception classes if the predefined classes are not sufficient.

Commonly Reused Exceptions

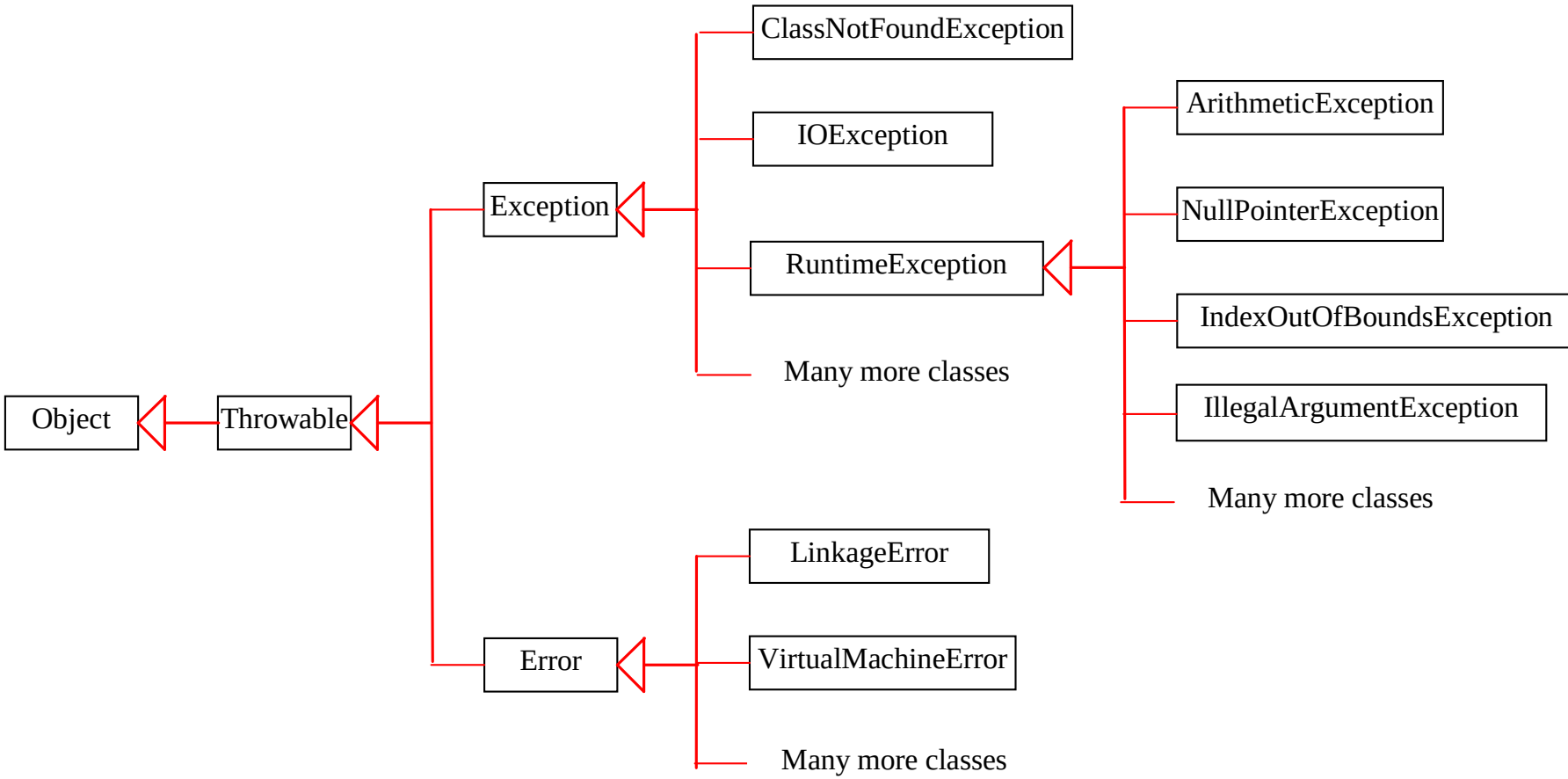
- Use of standard exceptions are generally preferred (Bloch, J., 2008)

Exception	Occasion for Use
IllegalArgumentException	Non-null parameter value is inappropriate
NullPointerException	Parameter value is null where prohibited
IllegalStateException	Object state is inappropriate for method invocation
IndexOutOfBoundsException	Index parameter value is out of range
ConcurrentModificationException	Concurrent modification of an object has been detected where it is prohibited
UnsupportedOperationException	Object does not support method

Defining Your Own Exceptions

- Define custom exception classes if the predefined classes are not sufficient.
- Define custom exception classes by extending Exception or a subclass of Exception.

Recall the Throwable Class Hierarchy



Defining Your Own Exception: Example

- Consider
  - What type of “error” or “exceptional” behavior we are to handle?
  - Which Exception/Error class we extend?
    - Checked vs unchecked?
    - Which subclass?
- Example
  - Define an InvalidRadiusException by extending the selected Exception/Error class (e.g., Exception)
  - Define an InvalidNameException by extending the selected Exception/Error class (e.g., Exception)

Questions?

- One can define her or his own Exception classes by subtyping the Exception class
- When should you use it?
- How do you define it, what's the process, and what are the design considerations?

(Optional) Exercise

- In this exercise, you are to create two custom exceptions, InvalidRadiusException and InvalidNameException
  - Create a directory in your journal
  - Create the following classes
    - Circle, CircleClient, InvalidRadiusException and InvalidationNameException
    - InvalidRadiusException and InvalidationNameException are unchecked exceptions
    - Handle the two exceptions in the main method of the CircleClient class
    - At the top of the CircleClient class, write a comment to compare and contrast the custom exceptions here with the two checked exceptions of the same names demonstrated in the lecture, e.g., advantages or disadvantages of each
  - Submit the work as a journal entry