

CISC 3115 TY2

# Exception and Some Best Practice

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Discussed
  - Approaches to handle errors (what-if and exceptions)
  - Concept of Exception
  - The Java throwable class hierarchy
    - system errors, runtime exceptions, checked errors, unchecked errors
  - Methods of declaring, throwing, catching exception, and rethrowing exceptions
  - Exception, call stack, stack frame, and stack trace
- Exception and some best practice

# Exceptions are for Exceptional Conditions

- Exception handling usually requires time and resources because it requires
  - instantiating a new exception object,
  - rolling back the call stack, and
  - propagating the errors to the calling methods.

# Some Best Practices

- Do throw specific Exceptions

```
throw new RuntimeException("Exception at runtime");
```



- Throw early, catch late.

- better to throw a checked exception than to handle the exception poorly.

- Use exception only for exceptional situations

```
if (args.length != 3) {  
    System.out.println("Usage ...");  
}
```



```
try {  
    d1 = Integer.parseInt(args[2]);  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Usage ...");  
}
```



# Throw Specific Exceptions?

- Use the exception classes in the API whenever possible.
- Define *custom exception* classes if the predefined classes are not sufficient.
  - How to define custom exception?

# Questions

- Exceptions are expensive, and are for exceptional conditions.
  - Use the exception classes in the API whenever possible.
  - Define *custom exception* classes if the predefined classes are not sufficient.
- Exceptions are commonly used for diagnosing problems in the programs, be specific!
- Exceptions are not abnormal. Organize your code.