Dealing with errors

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 7

When things go wrong

- Our code could encounter many types of errors
  - *User input* — enter invalid filenames or URLs
  - *Device errors* — printer jam, network connection drops
  - *Resource limitations* — disk full
  - *Code errors* — invalid array index, key not present in hash table, refer to a variable that is `null`, divide by zero, . . .

When things go wrong

- Our code could encounter many types of errors
  - *User input* — enter invalid filenames or URLs
  - *Device errors* — printer jam, network connection drops
  - *Resource limitations* — disk full
  - *Code errors* — invalid array index, key not present in hash table, refer to a variable that is `null`, divide by zero, ...
- Signalling errors
  - Return an invalid value: -1 at end of file, `null`
  - What if there is no obvious invalid value?

Exception handling

- Code that generates error raises or throws an exception

Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object

Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash

Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash
- . . . or passes the exception back up the calling chain

Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash
- . . . or passes the exception back up the calling chain
- Declare if a method can throw an exception
  - Compiler can check whether calling code has made a provision to handle the exception

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions
- `RunTimeException` — programming errors that should have been caught by code
  - Array index out of bounds, invalid hash key, . . .

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions
- `RunTimeException` — programming errors that should have been caught by code
  - Array index out of bounds, invalid hash key, ...
- Checked exceptions
  - Typically user-defined, code assumptions violated
    - In a list of orders, quantities should be positive integers

Summary

- Exception handling — gracefully recover from errors that occur when running code
- Throw an exception — generate an object encapsulating information about the error
- Catch an exception — decode the nature of the error and take corrective action
- Java organizes exceptions in a hierarchy, by type
  - `Error` — internal errors within JVM, “not the programmer’s fault”
  - `RunTimeException` — coding errors, could have been avoided by runtime checks in code
  - Checked exceptions — user-defined, violations of assumptions made by code
    - To contrast, `Error` and `RunTimeException` are called unchecked exceptions