11.4 Building Robust Applications

Key points to remember:

- Thanks to the compiler, checked exceptions are never "unexpected"; they are trapped or acknowledged.
- Unchecked exceptions (often caused by the end user) must be avoided and/or trapped.
- Defensive programming relies on validation to detect invalid inputs.
- Exception-based programming relies on exceptions.
- Both approaches can be employed in the same app.
- Logic errors are minimized through early exposure, e.g. strong typing, assertion, etc.

Building Robust Apps

- correctness: the degree to which software conforms to its specification.
- robustness: the ability of a software product to cope with unusual situations.
  - good coping: graceful, tolerant.
  - bad coping: crash.

- Even an app that never crashes might still be incorrect.
Building Robust Apps

• The goal of robustness means that we don’t want our software to crash.

• We will use all sorts of services, many of which potentially throw exceptions.

• Unhandled exceptions cause apps to crash.

• Crashing app == not robust app.

• Do we rely on our human abilities to track all of these potential sources of exceptions?
  • Humans make mistakes, even with the best of intentions.

Building Robust Apps

• What approach should we use to ensure that our app doesn’t crash?

• **Approach #1** – make sure the exceptions never get thrown in the first place!
  • Need to read all pre-conditions, see which parameter values trigger exceptions, and then avoid such parameter values.
  • Build in a whole bunch of if-then clauses and other ways of validation for parameters, **before** invoking services.

• **Approach #2** – let exceptions happen
  • Make sure all of the necessary handlers are in place.
Analysis: Approach #1

- Suppose our goal is to make sure the exceptions never get thrown in the first place
  - Need to read all pre-conditions, see which parameter values trigger exceptions, and then avoid such parameter values
  - This is prone to error (something can easily be missed)
  - This will be tedious and lengthy (can you imagine how much extra code will be needed? Can you imagine how difficult the code will be to read and understand?)
  - This is not so clever – you are duplicating the functionality that is already implemented in the services

- Conclusion: Don’t use this approach

Analysis: Approach #2

- Suppose our goal is to let exceptions happen and then make sure there are handlers
  - Many exceptions will be checked exceptions, which means the compiler will check that a handler has been added
  - Compiler will not enforce handling of unchecked exceptions, so onus is still on the implementer to ensure that handler has been added
  - Usually more compact to deal with exception rather than to prevent it from happening

- Conclusion: Use this approach