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.**

YORK UNIVERSITÉ UNIVERSITY

3

Building Robust Apps

- correctness : the degree to which software conforms to its specification
- robustness : the ability of a software product to cope with unusual situation
  - good coping – graceful, tolerant
  - bad coping : crash
- Even an app that never crashes might still be incorrect

YORK UNIVERSITÉ UNIVERSITY

4

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.

5

YORK UNIVERSITÉ UNIVERSITY

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

6

YORK UNIVERSITÉ UNIVERSITY

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 image 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

YORK UNIVERSITÉ UNIVERSITY

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

YORK UNIVERSITÉ UNIVERSITY