Programming Languages and Techniques (CIS120)

Lecture 28

Mar 26, 2012

Queue Iterators and Exceptions

Announcements

- HW08 is due *tonight* at 11:59:59pm
- HW09 will be available next Monday, Due Apr2.
- Midterm 2 is Friday, Mar 30th
  - Location is across campus: **FAGN AUD**
  - Review session: Wed 8-10PM in Levine 101
  - Lab this week is review (bring questions!)
- Final exam date is confirmed
  - Tuesday, May 8th 9-11AM

Queue Iterator

Exceptions

Dealing with the unexpected.

Sources of method Failure

- Some methods may require that their arguments satisfy certain preconditions
  - Input to max is a nonempty list, Item is non-null, no more elements for next
- Interfaces may be imprecise
  - Some Iterators don't support the "remove" operation
- External components might fail
  - Try to open a file that doesn't exist
- Resources might be exhausted
  - Program uses all of the computer's disk space
- These are all *exceptional circumstances…*
  - how do we deal with them?

Ways to handle failure

- Return an error value (or default value)
  - e.g. Math.sqrt returns NaN ("not a number") if given input < 0
  - e.g. Many Java libraries return null
  - e.g. file reading method returns -1 if no more input available
  - *Caller must check return value*
  - *Use with caution – easy to introduce hard to find bugs*
- Use an informative result
  - e.g. in OCaml we used options to signal potential failure
  - e.g. in Java, create a special class like option
  - *Passes responsibility to caller, but caller must do the proper check*
- Use exceptions
  - Available both in OCaml and Java
  - Any caller can handle the situation
  - If exceptions are uncaught, the program terminates

Exceptions

- An exception is an *object* representing abnormal conditions.
  - Its internal state describes what went wrong
  - e.g. NullPointerException, IllegalArgumentException, IOException
  - Can define your own exception classes
- *Throwing* an exception is an *emergency exit* from the current method.
  - The exception propagates up the invocation stack until it either reaches the top and the stack, in which case the program aborts with the error, or the exception is caught
- *Catching* an exception lets callers take appropriate actions to handle the abnormal circumstances

Example

```
void loadImage (String fileName) {
  try {
      Picture p = new Picture(fileName);     // could fail

      // ... code to display the new picture in the window
      // executes only if the picture is successfully created.

   } catch (IOException ex) {

      // Use the GUI to send an error message to the user
      // using a dialog window
      JOptionPane.showMessageDialog(
          frame,                   // parent of dialog window
                                   // error message to display
          "Cannot load file\n" + ex.getMessage(),
          "Alert",                         // title of dialog
          JOptionPane.ERROR_MESSAGE    // type of dialog
         );
  }
```