CMSC 132: Object-Oriented Programming II

Program Correctness

Department of Computer Science
University of Maryland, College Park
Overview

Program correctness is determined by the presence / absence of program defects (errors)

Issues

- Types of errors
- Exceptions
- Testing
- Debugging
Program Errors

Types of errors
- Compile-time (syntax) errors
- Run-time errors
- Logic errors
Program Errors – Compile Time

- Compile-time (syntax) errors
  - Errors in code construction
    - Lexical (typographical), grammatical, types
  - Detected during compilation
  - Usually easy to correct quickly

Examples
- Misspelled keyword
- Missing or misplaced symbol
- Incorrect operator for variable type
Program Errors – Run Time

- Run-time errors
  - Operations illegal / impossible to execute
  - Detected during program execution
    - But not detectable at compile time
  - Treated as exceptions in Java

- Example
  - Division by zero
  - Array index out of bounds
  - Using null pointer
  - Illegal format conversion
Program Errors – Logic

- Logic errors
  - Operations leading to incorrect program state
  - May (or may not) lead to run-time errors
  - Problem in design or implementation of algorithm

- Examples
  - Computing incorrect arithmetic value
  - Ignoring illegal input

- Hardest error to handle
  - Detect by testing
  - Fix by debugging
Exceptions

- Rare event outside normal behavior of code
  - Usually a *run-time error*

Examples

- Division by zero
- Access past end of array
- Out of memory
- Number input in wrong format (float vs. integer)
- Unable to write output to file
- Missing input file
Exception Handling – Throw Exception

Approach
- Throw exception

Example

```java
A( ) {
    if (error) throw new ExceptionType();
}
B( ) {
    try {
        try {
            A( );
        } catch (ExceptionType e) { ...action... }
    }
}
```

Java exception backtracks to caller(s) until matching catch block found
Representing Exceptions in Java

- Exceptions represented as
  - Objects derived from class Throwable

- Code

  ```java
  public class Throwable() extends Object {
    Throwable() // No error message
    Throwable(String mesg) // Error message
    String getMessage() // Return error mesg
    void printStackTrace() { … } // Record methods
    … // called & location
  }
  ```
Generating & Handling Exceptions

Java primitives
- Try
- Throw
- Catch
- Finally

Procedure for using exceptions
1. Enclose code generating exceptions in try block
2. Use throw to actually generate exception
3. Use catch to specify exception handlers
4. Use finally to specify actions after exception
try { // try block encloses throws
    throw new eType1(); // throw jumps to catch
}
catch (eType1 e) { // catch block 1
    ...action... // run if type match
}
catch (eType2 e) { // catch block 2
    ...action... // run if type match
}
finally { // final block
    ...action... // always executes
}
Exceptions – Examples

- `FileNotFoundException` (java.io)
  - Request to open file fails

- `IllegalArgumentException` (java.lang)
  - Method passed illegal / inappropriate argument

- `IOException` (java.io)
  - Generic I/O error

- `NullPointerException` (java.lang)
  - Attend to access object using null reference

- `UnsupportedOperationException` (java.lang)
  - Object does not provide requested operation
Exceptions – Examples

- **Used in programming project**
  ```java
  public void MethodRequiredForProject() {
    throw new UnsupportedOperationException(
        "You must implement this method.");
  }
  ```

- **Behavior**
  - If method is invoked during program execution
  - Exception is thrown
    - Of type `UnsupportedOperationException`
    - Message string is displayed
  - Program execution stops unless exception caught
Testing

Run program (or part of program) under controlled conditions to verify behavior
- Detects run-time error if exception thrown
- Detects logic error if behavior is incorrect

Issues
- Selecting test cases
- Testing different parts of program
- Visibility of program code

Covered in more detail later…
Debugging

- Process of finding and fixing software errors
  - After testing detects error

Goal

- Determine cause of run-time & logic errors
- Correct errors (without introducing new errors)

Similar to detective work

- Carefully inspect information in program
  - Code
  - Values of variables
  - Program behavior
Debugging – Approaches

Classic
- Insert debugging statements
- Trace program control flow
- Display value of variables

Modern
- IDE (integrated development environment)
- Interactive debugger
Interactive Debugger

Capabilities

- Provides trace of program execution
- Shows location in code where error encountered
- Interactive program execution
  - Single step through code
  - Run to breakpoints
- Displays values of variables
  - For current state of program
Interactive Debugger

Single step
- Execute single line of code at a time
- When executing method, can
  - Finish entire method
  - Execute first line in method
- Tedious (or impractical) for long-running programs
Interactive Debugger

**Breakpoint**
- Specify location(s) in code
- Execute program until breakpoint encountered
- Can skip past uninteresting code