CSCI 136
Data Structures & Advanced Programming

Conditions & Assertions

Fall 2020

// Pre: instructor.name().equals("Bill")

Program Correctness Aids

Error Detection

Two types: Compile-time and Run-time

- Compiler error checking
  - Syntax errors
  - Type errors (but not all of them)
    - Java is *statically typed*: Variables must have type declared
      - Allows much more extensive compile-time type checking
- Run-time error checking
  - Type errors
    - E.g.: Passing incorrect type to `equals()`
  - Logic errors
    - E.g.: Division by 0

Code Documentation

Code should be well-documented

- Clear description of correct usage

For methods, this should include

- Description of expectations
  - What information should be passed to parameters
  - Constraints on that information
- Description of effects of executing method
  - Effects on object on which method was invoked
  - Effects on parameters
  - Other effects

Pre and Post Conditions

A widely used documentation convention

Example:

```
/* Compute the square root of a number
 * Pre: x is non-negative
 * Post: return value is non-negative
 * square root of x
*/
public static double sqrt(double x) {
     …
}
```

Pre and Post Conditions

Example

- Recall `charAt(int index)` in Java String class
- What are the pre-conditions for charAt?
  - 0 <= index < length()
- What are the post-conditions?
  - Method returns char at position index in string
- Expectation: Use in your methods as appropriate

```
/* pre: 0 ≤ index < length
 * post: returns char at position index
 */
public char charAt(int index) { … }
```

Other Examples

Pre-conditions are often used to avoid error-checking code in frequently executed methods

Ex. Equality-testing

- instead of using `instanceof` check

```
    // Pre: other is of type Card
    // Post: Returns true if suits and ranks match
    public boolean equals(Object other) {
       Card oc = (Card) other;
       return this.getRank() == oc.getRank() &&
              this.getSuit() == oc.getSuit();
}
```

Pre and Post Conditions

- Pre and post conditions "form a contract"
  - *If* pre-condition is true when method is *invoked*
  - *Then* Post-condition holds when method *returns*
- These conditions document requirements that user of method should satisfy
- But, as comments, *they are not enforced*

Assert Class

- Pre- and post-condition comments are important for *documenting* code.
- It would be *better* in some cases to check that a pre-condition was violated.
- Program could then
  - Catch error and gracefully halt
  - Provide helpful error message
- The Assert class (in structure5 package) supports this goal

Assert Class

The Assert class contains the methods

```
public static void pre(boolean test, String message);
public static void post(boolean test, String message);
public static void condition(boolean test, String message);
public static void fail(String message);
```

If the boolean test is NOT satisfied, an exception is raised, the message is printed and the program halts. That is:

- The *test* is a condition we desire to be *true*
- The *message* is printed if the condition is *false*

Assert Examples

The Vector class uses Assert in many places

```
// Pre: initialCapacity >= 0
public Vector(int initialCapacity) {
      Assert.pre(initialCapacity >= 0,"Capacity
            must not be negative");

// Pre: 0 <= index && index < size()
public E elementAt(int index) {
      Assert.pre(0 <= index && index < size(),"index
            is within bounds");
```

Note: Asserts and pre/post conditions serve different purposes

- Pre/Post conditions document usage constraints to user
- Asserts perform run-time checks ensuring that conditions are met

General Rules about Assert

- State pre/post conditions in comments
- Check conditions in code using `Assert` class
  - or Java's `assert` keyword
- Use `Assert.fail()` in unexpected cases
  - E.g.: In the default block of certain switch statements
- Note: They are always active
  - Java's `assert` check can be disabled at runtime

The Java assert keyword

- An alternative to Duane's Assert class
- Added in Java 1.4
- Two variants
  - `assert boolean_expression`
    - Throws an `AssertionError` if the expression is false
  - `assert boolean_expression : other_expression`
    - In addition, prints value of `other_expression`
- By default, assert statements are ignored
  - Turn them on with –ea flag
  - Usage: java –ea MyJavaClass

Summary

- *Pre-conditions* specify conditions required for successful method execution
- *Post-conditions* specify effects of method assuming pre-conditions are satisfied
- They are *documentation* elements
- Assertions are run-time checks
  - Checks that condition is satisfied
  - Two options: Assert class or Java's assert keyword
- Your code should use these tools!