Assertions

Introduction to Programming and Computational Problem Solving: Accelerated Pace

CSE 11

Lecture 17

Announcements

- Assignment 8 is due Jun 5, 11:59 PM
  - Upgrade beginning Jun 8, 12:01 AM

Exceptions

- Exceptions are runtime errors caused by your program and external circumstances
  - These errors can be caught and handled by your program

Exception handling

- Exception handling separates error-handling code from normal programming tasks
  - Makes programs easier to read and to modify
- The **`try`** block contains the code that is executed in **normal** circumstances
- The **`catch`** block contains the code that is executed in **exceptional** circumstances
- A method should **throw** an exception if the error needs to be handled by its caller
- **Warning: exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods**

Assertions

- Programming with Assertions
  https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
- An assertion is a Java statement that enables you to assert an assumption about your program
- An assertion contains a Boolean expression that should be true during program execution
- Assertions can be used to assure program correctness and avoid logic errors

Declaring assertions

- An assertion is declared using the Java keyword `assert`

  ```
  assert assertion;
  ```

  or

  ```
  assert assertion : detailMessage;
  ```

  where `assertion` is a Boolean expression and `detailMessage` is a primitive-type or an `Object` value

Executing assertions

- When an assertion statement is executed, Java evaluates the assertion
- If it is `false`, an `AssertionError` will be thrown
- The `AssertionError` class has a no-arg constructor and seven overloaded single-argument constructors of type `int`, `long`, `float`, `double`, `boolean`, `char`, and `Object`

Executing assertions

- For the first `assert` statement with no detail message, the no-arg constructor of `AssertionError` is used
- For the second assert statement with a detail message, an appropriate `AssertionError` constructor is used to match the data type of the message
- Since `AssertionError` is a subclass of `Error`, when an assertion becomes `false`, the program displays a message on the console and exits

Executing assertions example

```
public class AssertionDemo {
  public static void main(String[] args) {
    int i;
    int sum = 0;
    for (i = 0; i < 10; i++) {
      sum += i;
    }
    assert i == 10;
    assert sum > 10 && sum < 5 * 10 : "sum is " + sum;
  }
}
```

Executing assertions example

- **A best practice is to place assertions in a switch statement without a default case**
  - Example

```
switch (month) {
  case 1: ... ; break;
  case 2: ... ; break;
  ...
  case 12: ... ; break;
  default: assert false : "Invalid month: " + month;
}
```

Running programs with assertions

- By default, the assertions are disabled at runtime
- To enable them, use the switch `-enableassertions`, or `-ea` for short, as follows

```
java -ea AssertionDemo
```

- Assertions can be selectively enabled or disabled at class level or package level
- The disable switch is `-disableassertions` or `-da` for short
- For example, the following command enables assertions in package `package1` and disables assertions in class `Class1`

```
java -ea:package1 -da:Class1 AssertionDemo
```

Using exception handling or assertions

- **Assertions should not be used to replace exception handling**
- *Exception handling* deals with unusual circumstances during program execution
- *Assertions* are to assure the correctness of the program
- *Exception handling* addresses robustness
- *Assertions* address correctness
- Like exception handling, assertions are not used for normal tests, but for internal consistency and validity checks
- Assertions are checked at runtime and can be turned on or off at startup time

Using exception handling or assertions

- **Do not use assertions for argument checking in public methods**
- Valid arguments that may be passed to a public method are part of the method's contract
- The contract must always be obeyed whether assertions are enabled or disabled
  - For example, the following code in the Circle class should be rewritten using exception handling

```
public void setRadius(double newRadius) {
  assert newRadius >= 0;
  radius =  newRadius;
}
```

Programming with assertions

- **Use assertions to reaffirm assumptions**
- This gives you more confidence to assure correctness of the program
- A common use of assertions is to replace assumptions with assertions in the code
- **A best practice is to use assertions liberally**
- Assertions are checked at runtime and can be turned on or off at startup time, *unlike exception handling*

Next Lecture

- Binary file input/output