Code Clarity - Comments, Preconditions and Postconditions

CMPUT 115 - Lecture 3
Department of Computing Science
University of Alberta
© D. Szafron,

Slide # 2

About This Lecture

- In this lecture we will learn how to write Java code that is easy to understand, both as it is being written and later when it is being read.

Slide # 3

Outline

- Comments
- Preconditions and Postconditions
- Assertions

Slide # 4

Comments

A comment is an annotation that is added to program code that is ignored by the compiler and run-time system. Example:

```java
public class Ratio {
    /* an object for storing a fraction */
}
```

A comment should describe
- what a program segment should do
- when it should do it (i.e. under what circumstances)

There should be an opening comment for each
- Class
- Method

Slide # 5

Why bother with comments?

- Comments are useful:
  1. *as the code is being written:*
     - allows programmer to check for desired functionality
  2. *later on:*
     - helps programmers understand what has been written & why it was written that way
  - This aids productivity; reduces chance of error
  - Saves time, effort and … money!

Slide # 6

Good & Bad Comments

- A good comment is one that is cherished by a programmer for the clarity it brings
  ```java
  public class Ratio {
      /* an object for storing a fraction like 2/3 */
  }
  ```

- A neutral comment is one that neither helps nor hinders the programmer
  ```java
  protected int numerator; /* numerator of ratio */
  ```

- A bad comment is one that confuses or inhibits the programmer
  ```java
  public class Ratio {
      /* this class is tricky: talk to Bob */
  }
  ```
When do you write comments?

- Comments should be created as soon as a designer or programmer knows the functionality of the segment being designed or implemented.
- Comments should not be added as an afterthought!
- Note: if you use meaningful identifiers for variables and methods, you will need fewer explanatory comments throughout your code.

Pre and Post Conditions

- Natural language is not very precise at describing program functionality; too verbose.
- Mathematics is more precise, but not always as easy to understand.
- In this course, we will use an informal form of a notation called preconditions and postconditions for our method comments.

Example

- To illustrate the use of pre-conditions and post-conditions, we will use the Ratio constructor.
- You can look back at the “Interfaces” lecture for the full code listing of Ratio.java if you have forgotten about this class.

```java
public Ratio(int top, int bottom) {
    /* Initialize the receiver to be the fraction whose numerator is the top and whose denominator is the bottom. The bottom cannot be zero. */
}
```

Preconditions

- A precondition tells when (under what circumstances) a method can be called:

```java
public Ratio(int top, int bottom) {
    /* Initialize the receiver to be the fraction whose numerator is the top and whose denominator is the bottom.
    The bottom cannot be zero. */
    pre: bottom != 0
    post: constructs a ratio equivalent to top/bottom
}
```

Postconditions

- A postcondition tells what a method must do:

```java
public Ratio(int top, int bottom) {
    /* Initialize the receiver to be the fraction whose numerator is the top and whose denominator is the bottom.
    pre: bottom != 0
    post: constructs a ratio equivalent to top/bottom */
}
```

Method Comment Format

- The preconditions and postconditions form the comment for the method. So now our constructor looks like…

```java
public Ratio(int top, int bottom) {
    /* Initialize the receiver to be the fraction whose numerator is the top and whose denominator is the bottom.
    The bottom cannot be zero. */
    pre: bottom != 0
    post: constructs a ratio equivalent to top/bottom */
}
Assertions

An assertion is a statement about the state of your program that must be true.
Most assertions can be represented as Boolean expressions that can actually be evaluated when the program is run.
In this course we will never check postconditions.
We will also sometimes remove precondition checks to improve efficiency after the program is tested (code in the lecture notes will not check preconditions to save space on the slides).

Summary

- Comments and annotations are important – they help us understand code
- The use of pre-conditions and post-conditions is a more formal, structured approach that we will use in this course
- Comments shall be precise
  - If the method cannot be described by one or two sentences, rethink about your design