Documentation and Programming Style

Appendix A

Naming Variables and Classes

- Name should suggest use of variable
- Follow practice/convention of other programmers
  - Variable names start with lower case
  - Class names start with upper case
  - If variable has multiple words, each word starts with capital letter
  - Constant names in all caps

Indenting

- Use indenting to indicate structure
  - Makes program easier to read
- Each class begins at left margin, uses braces to enclose definition

```
public class CircleCalculation
{
   . . .
} // end CircleCalculation
```

Indenting

- Data fields and methods appear indented within these braces

```
public class CircleCalculation
{
   public static final double PI = Math.PI;

   public static void main(String[] args)
   {
      double radius; // In inches
      double area;   // In square inches
      . . .
   } // end main
} // end CircleCalculation
```

Indenting

- Each level of nesting indented from previous level to show nesting more clearly.
- When in doubt, use ctl-tab-F in eclipse.
- Outermost structure not indented at all.

```
public class CircleCalculation
{
   public static final double PI = Math.PI;

   public static void main(String[] args)
   {
      double radius; // In inches
      double area;   // In square inches
      . . .
   } // end main
} // end CircleCalculation
```

Indenting

- If statement does not fit on one line
  - Write it on two or more lines.
- When you write single statement on more than one line
  - Indent the successive lines more than the first line

```
System.out.println("The volume of a sphere whose radius is " +
                   radius + " inches is " + volume +
                   " cubic inches.");
```

Self Documenting

- Documentation for a program describes what the program does and how it does it
- Clean style, well-chosen names make program purpose and logic clear
- Programs also need explanation to make completely clear.
  - Given in the form of comments.

Comments

- Single line comments
  - Double slash

```
String sentence; // Spanish version
```

- Comment blocks
  - Enclosed by /* *comment* */
- When to use comments
  - Explanation at beginning of program
  - What program does, name of author, date, etc.

Java Documentation Comments

- Utility program named *javadoc*
  - Generates HTML documents that describe your classes
- Extracts
  - Header for your class
  - Headers for all public methods
  - Comments written in a certain form

Java Documentation Comments

- Conditions for *javadoc* to extract a comment
  - Must occur immediately before public class definition or header of public method.
  - Comment must begin with /** and end with */
- Comments written for *javadoc* usually contain special tags
  - Tag *@author* identifies programmer's name, required of all classes and interfaces.

Java Documentation Comments

- Other tags of interest to us are used with methods
  - Must appear in following order within comment that precedes method header:

```
@param
@return
@throws
```

Java Documentation Comments

- The @param tag.
  - Written for every parameter in a method
  - List these tags in order in which parameters appear in method's header
- Example tag

```
@param code      The character code of the ticket category.
@param customer  The string that names the customer.
```

- Resulting documentation

```
code - The character code of the ticket category.
customer - The string that names the customer.
```

Java Documentation Comments

- The @return tag
  - Must write a @return tag for every method that returns a value
  - Tag must come after any @param tags in comment
  - Do *not* use this tag for void methods and constructors

Java Documentation Comments

- The @throws tag
  - If a method can throw a checked exception, name by using @throws tag

Sample *javadoc* Comments

```
/** Adds a new entry to a roster.
    @param newEntry     The object to be added to the roster.
    @param newPosition  The position of newEntry within the roster.
    @return  True if the addition is successful.
    @throws  RosterException if newPosition < 1 or newPosition > 1 + the length
             of the roster. */
```

Resulting javadoc Output

add

```
public boolean add(java.lang.Object newEntry,
                   int newPosition)
            throws RosterException
```

Adds a new entry to a roster.

Parameters:

`newEntry` - The object to be added to the roster.
`newPosition` - The position of newEntry within the roster.

Returns:

True if the addition is successful.

Throws:

`RosterException` - if newPosition < 1 or newPosition > 1 + the length of the roster.

Documentation and Programming Style

End