CS2136: Paradigms of Computation

Class 11: Java: Javadoc

Classes: Data, Methods, Encapsulation

Copyright 2001, 2002, 2003 Michael J. Ciaraldi and David Finkel
Javadoc
Javadoc

- Purpose: automatically generate documentation.
- Where it applies (section 3.7 of language spec)
- What’s included (section 18.1)
- Summary sentence (section 18.3)
- Tagged paragraphs (section 18.4)
- Invoking Javadoc
“Java defines three kinds of comments:

- /* text */ A traditional comment: all the text from the ASCII characters /* to the ASCII characters */ is ignored (as in C and C++).
- // text A single-line comment: all the text from the ASCII characters // to the end of the line is ignored (as in C++).
- /** documentation */ A documentation comment: the text enclosed by the ASCII characters /** and */ can be processed by a separate tool to prepare automatically generated documentation of the following class, interface, constructor, or member (method or field) declaration.”

Remember, you need the two stars for a doc comment.
What’s Included
(section 18.1)

"The text of a documentation comment consists of the characters between the /** that begins the comment and the */ that ends it. The text is divided into one or more lines. On each of these lines, leading * characters are ignored; for lines other than the first, blanks and tabs preceding the initial * characters are also discarded. So, for example, in the comment:

/**XYZ
 ** Initialize to pre-trial defaults.
  123*/

the text of the comment has three lines. The first line consists of the text "XYZ"; the second line consists of the text " Initialize to pre-trial defaults." and the third line consists of the text "123"."
Summary Sentence
(Section 18.3)

“The first sentence of each documentation comment should be a summary sentence, containing a concise but complete description of the declared entity. This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tagline (as defined below).”
Tagged Paragraphs
(section 18.4)

“A line of a documentation comment that begins with the character @ followed by one of a few special keywords starts a tagged paragraph. The tagged paragraph also includes any following lines up to, but not including, either the first line of the next tagged paragraph or the end of the documentation comment.”

You cannot define your own keywords.

In general, you can write anything after the keyword (not “anything after the ‘@’ ”).
Tagged Paragraph Keywords

- For classes and interfaces:
  - @author
  - @version

- For methods and constructors:
  - @param
  - @return
  - @exception

- Anywhere:
  - @see
Invoking Javadoc

```
javadoc -private filename
```

E.g.

```
javadoc -private Dot.java
```

E.g.

```
javadoc -private *.java
```
Classes
Classes

z Remember, in Java almost everything is either:
  y a class (defines data and methods)
  y an object (instance of a class)
Defining a Class

- Define the data fields (a.k.a. attributes).
- Define the methods.
Data

Can be:
- Primitives
- References to objects

You could have a class with only data (no methods).
- Like a data structure in C or Pascal.
- You would have to access the data directly.
Methods

- Invoke them on the object.
- Specify zero or more arguments (parameters).
- Return value can be any valid type including void.
- Disambiguate methods by their signatures i.e., calling sequences.
- Not by return type.
Allocating an Object: The Constructor

z (Almost) every class has a constructor method.

  y Constructor has same name as class.
  y Triggered by keyword `new`.

z A constructor might take a variable number of parameters.

  y Including none at all!
A Sample Class: Dot
See Dot.java and DotTest.java

- This class holds one point on a two-dimensional plane.
- Need to access it with rectangular or polar coordinates.
Coordinates

z Rectangular

y x, y

y Measures distance from origin along x-axis and y-axis.

z Polar

y r, • (theta)

y Measures radius from origin, angle counterclockwise from x-axis.
Geometric interpretation

- $r =$ distance of $(x, y)$ to $(0, 0)$, $\theta =$ angle
Coordinates Conversion

Polar coordinates of a point

• Let \((x, y)\) be a point of the plane. Its polar coordinates \((r, \theta)\) are defined by:

\[
    r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1}\left(\frac{y}{x}\right)
\]

• or equivalently

\[
    x = r \cos\theta, \quad y = r \sin\theta
\]
import java.util.*;
import java.lang.*;
public class Dot {
    private double x, y;
    Dot() { // No parameters: use defaults
        x = 0.0d; y = 0.0d; }
    Dot(double xarg, double yarg) {
        // 2 parameters: use both
        x = xarg; y = yarg; }
}
**Access Methods: Get**

```java
public double getX() { return x; }
public double getY() { return y; }
public double getR() {
    return Math.sqrt((x * x) + (y * y));
}
public double getTheta() {
    return Math.atan2(y, x);
}
```
Access Methods: Set

```java
public void setX(double xarg) { x = xarg; }
public void setY(double yarg) { y = yarg; }
```

z Setting with polar coordinates is left as an exercise for the interested student. J
public String toString() {
    // converts a dot's location to a string
    String ts;
    ts = "x = " + Double.toString(x) + " y = " + Double.toString(y);
    return ts;
}

Can be called any time, or automatically by print() and println().
Next Time

- More Java: More on Classes and Inheritance