Lect #4

- More details about the Java Program
  - Semantics (meaning) and Syntax (grammar rules)
    - Methods
    - Statements
    - Variables, Types, Assignment,
    - Values and Expressions

Reading:

- L-D-C: Chapter 2

Announcements

- Labs: how many need to sign up yet?
- Class Rep: “My name is Josh Pene and I'm a second year student studying BCom majoring in management and data science. I have been a class rep for QUAN 102 & MGMT 205, so I have some experience in the role. I have no problem meeting new people and I'm also a fantastic problem solver. Let's pass COMP together!”

Temperature Converter (again)

```
import ecs100.*;

/** Program for converting between temperature scales */
public class TemperatureCalculator{

    /** Print conversion formula */
    public void printFormula ( ) {
       UI.println("Celsius = (Fahrenheit - 32) *5/9");
    }

    /** Ask for Fahrenheit and convert to Celsius */
    public void doFahrenheitToCelsius(){
        double fahrenheit = UI.askDouble("Farenheit:");
        double celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
        UI.println(fahrenheit +  " F -> " + celsius + " C");
    }
}
```

Writing your own programs

How?

- Use other programs as models, and then modify
  - Very useful strategy
  - Lectures have examples that you can use as models for your assignment programs

A new program

- Calculator to
  - convert kilograms to pounds, and to ounces
  - convert pounds and ounces to kilograms

```
import ecs100.*;
/** Program to convert weights */
```

```
public class TemperatureCalculator{

    public void doFahrenheitToCelsius(){
        double fahrenheit = UI.askDouble("Farenheit:");
        double celsius = (fahrenheit – 32.0) * 5.0 / 9.0;
        UI.println(fahrenheit +  " F -> " + celsius + " C");
    }
```

Writing your own programs

How?

- Use other programs as models, and then modify
  - Very useful strategy

BUT

  - It can be hard to work out how to modify

Need to understand the language

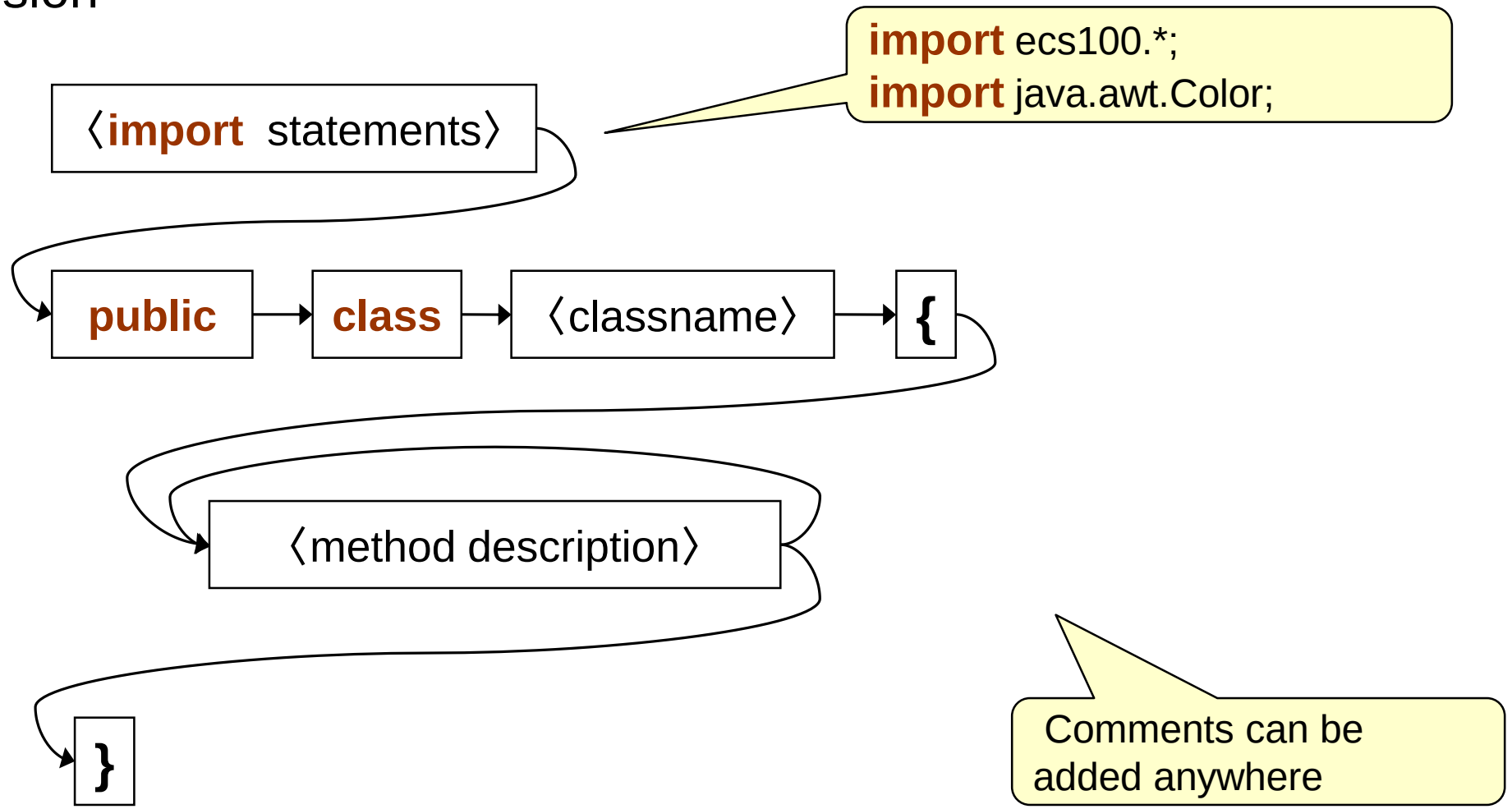
⇒ vocabulary

⇒ syntax rules

⇒ meaning (“semantics”)

Syntax rules: Program structure

- First version



Comments

Three kinds of comments:

- Documentation comments

Top of class,
Before each method

eg /** Program for converting between temperature scales */

- end-of-line comments

at end of any line

eg double celsius = (fahren – 32.0) * 5.0 / 9.0; // compute answer

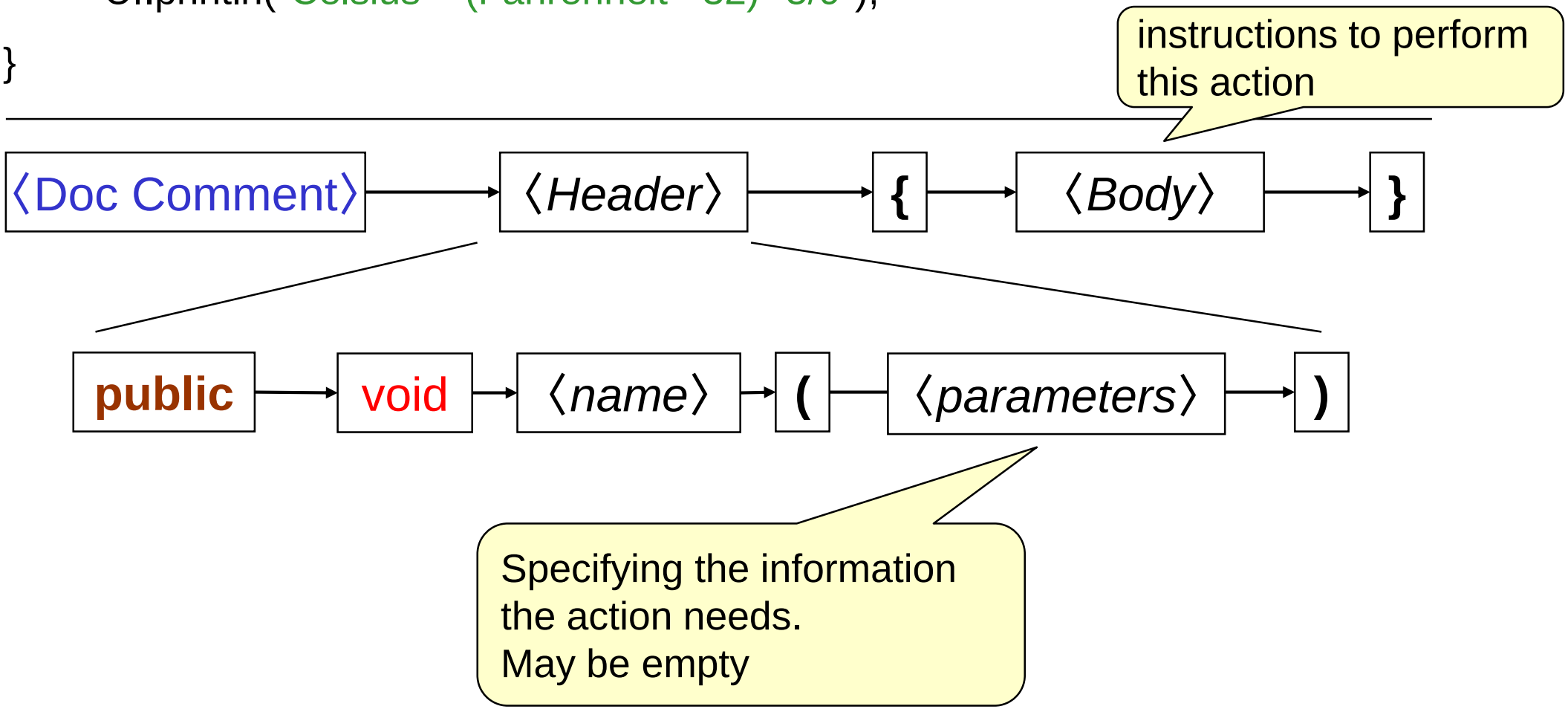
- anywhere comments

multi-line, or
middle of line, or …

eg /* double fahren = celsius * 9 / 5 + 32;
UI.println(celsius + “C is " + fahren + " F"); */

Method Definitions

```
/** Print out the conversion formulas */
   public  void  printFormula ( ) {
         UI.println("Celsius = (Fahrenheit - 32) *5/9");
   }
```



"Statements" (instructions)

(programmer jargon: single instructions are called "statements" for silly historical reasons!)

Two important kinds of statements:

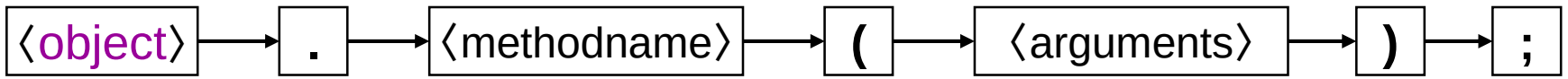
- method call statement:
  - tell some object to perform one of its methods.
    eg: tell the UI object to ask the user for a number
    eg: tell this object to print the celsius value of a temperature
    eg: tell the UI object to print out a string
    eg: tell the UI object to add a button
- assignment statement
  - compute some value and put it in a place in memory.

Method Calls

```
/** Print out the conversion formulas */
public void printFormula(){
    UI.println( "Celsius = (Fahrenheit - 32) *5/9" );
}
```

- Method call Statement:

who **.** *what* **(** *data to use* **)** **;**



UI **.** println **(** “Celsius = (Fahren…” **)** **;**

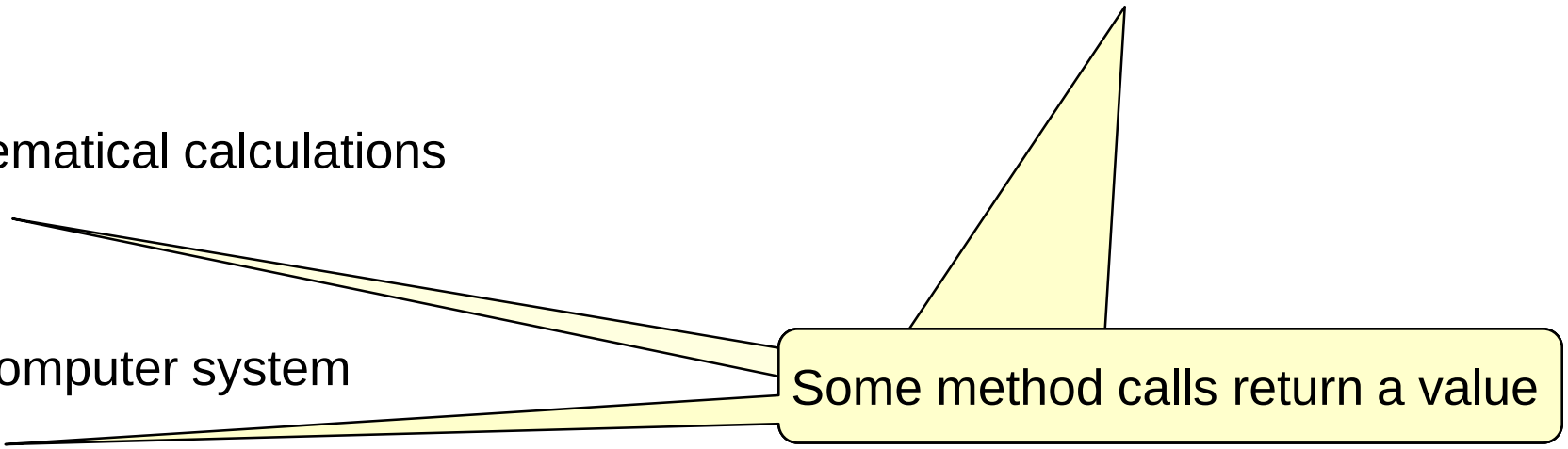
- Meaning of Statement:
  - Tell the object
    to perform the method
    using the argument values provided

Objects and their methods in Java

- What objects are there?

Predefined eg:

- UI a "User Interface" window with several panes
  - ➔ quit() addButton(…) println(….) drawRect(…) clearGraphics(), askDouble(…) askString(…)
- Math methods for mathematical calculations
  - ➔ random( ), sin(…)
- System representing the computer system
  - ➔ currentTimeMillis( )



Others

- this The object(s) defined by this class in your program
- New objects that your program creates

Lect #5

- More details on Java
- Programes with graphical output.

Variables

```
/** Convert from fahrenheit to Celsius */
public  void  doFahrenheitToCelsius(){
    double fahrenheit = UI.askDouble("Fahrenheit:" );
    double celsius = (fahrenheit – 32.0) * 5.0 / 9.0;
    UI.println(fahrenheit  + “F is " + celsius + “ C");
}
```

Use a variable whenever you need the computer to remember something temporarily.

- A variable is a place in memory that can hold a value.
  - Must specify the **type** of value that can be put in the variable
    ⇒ “Declare” the variable first time it is mentioned.

    ⟨type⟩ → ⟨variable⟩

    Asking for a place
  - Must put a value into a variable before you can use it
    ⇒ “assign” to the variable
  - Can *use* the value by specifying the variable’s name
  - Can change the value in a variable (unlike mathematical variable)

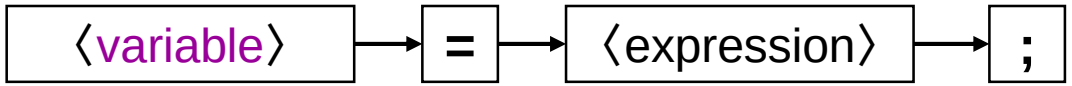
Assignment Statements

```
/** Convert from fahrenheit to Celsius */
public  void  doFahrenheitToCelsius(){
    double fahrenheit = UI.askDouble("Fahrenheit:" );
    double celsius = (fahrenheit – 32.0) * 5.0 / 9.0;
    UI.println(fahrenheit  + “F is " + celsius + “ C");
}
```

Putting a value into a variable

- Assignment Statement:

where = *what* ;



name-of-place = *specification-of-value* ;

formula = " Celsius = (Fahrenheit - 32) *5/9"

Meaning: Compute the value and put it in the place

Expressions

```
/** Convert from fahrenheit to Celsius */
public  void  doFahrenheitToCelsius(){
    double fahrenheit = UI.askDouble("Fahrenheit:" );
    double celsius = (fahrenheit – 32.0) * 5.0 / 9.0;
    UI.println(fahrenheit  + “F is " + celsius + “ C");
}
```

combining declaration and assignment into a single line

+ for Strings: "concatenates" the Strings

- Expressions describe how to compute a value.
- Expressions are constructed from
  - values
  - variables
  - operators (+, -, *, /, etc)
  - method calls that return a value
  - sub-expressions, using (…)
  - …

Values / Data

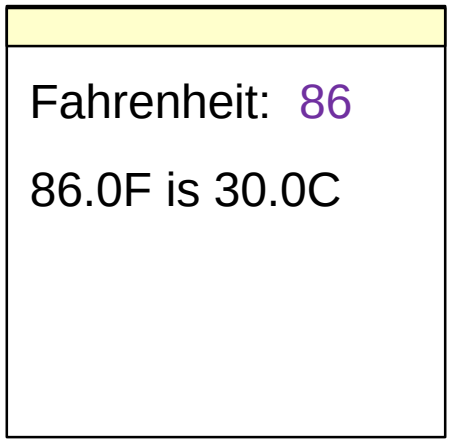
There are lots of different kinds ("Types") of values:

- Numbers
  - Integers **int** (or long) 42 -194573203
  - real numbers **double** (or float) 42.0 16.43 6.626e-34
  - …
- Characters char 'X' '4'
- Text **String** " F -> "
- Colours Color Color.red Color.green
- Methods this::doFahrenheitToCelsius
- Other Objects
- …

Method Calls and variables: a metaphor

Method Definition: Like a pad of worksheets

```
public void doFahrenheitToCelsius(){
    double fahrenheit = UI.askDouble("Fahrenheit:");
    double celsius  = (fahrenheit – 32.0) * 5.0 / 9.0;
    UI.println(fahrenheit + " is " + celsius + " C");
}
```



Calling a Method:

86 0

30 0

tempCalc1.fahrenheitToCelsius();

⇒ get a "copy" of the method worksheet

⇒ perform each action in the body.

⇒ throw the worksheet away (losing all the information on it)

Summary of Java program structure

- A Class specifies a type of object
  - TemperatureCalculator.class describes TemperatureCalculator objects
- A Class contains a collection of methods
  - Each method is an action the objects can perform.
  - TemperatureCalculator objects can do celsiusToFahrenheit, fahrenheitToCelsius, printFormula
  - If you have an object, you can call its methods on it.
- A method definition contains statements
  - Each statement specifies one step of performing the action
  - Method call statements
  - Declaration and Assignment statements

Assignment 1

- Calculator programs
- Programs to draw flags and other shapes

- core/completion/challenge
  - core is based fairly directly on the lectures
  - completion requires more problem solving
  - challenge may require finding additional stuff out for yourself.

Programs with graphics output

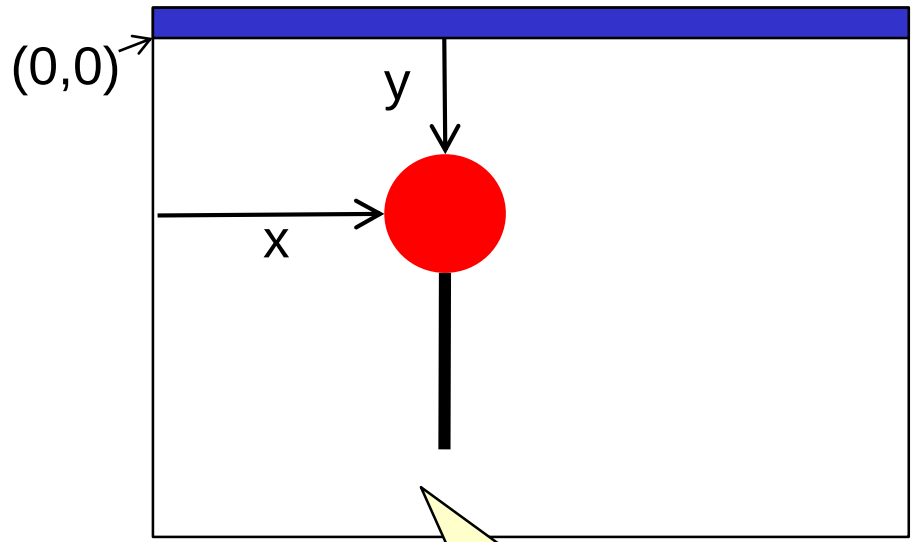
- Write a program to draw a lollipop:

Design

- What shapes can we draw?
  - UI has methods to draw rectangles, ovals, lines, arcs,…

⇒ Draw
one thick black line
one red oval,

Shapes are drawn on top of previous shapes



Coordinates measured from left and top

- How do we draw them?
  Need to set the color first (initially black)
  then call the draw/fill methods:
  - must specify the positions and size
    - rectangles/ovals: left, top, width, height
    - lines: x and y of each end.

Some UI methods

Text:

UI.clearText()

UI.print(*anything*) UI.println(*anything*) UI.printf(*format-string*, *values*…)

UI.askString(*prompt-string*) UI.askToken(*prompt-string*)

UI.askDouble(*prompt-string*) UI.askInt(*prompt-string*)

UI.askBoolean(*prompt-string*)

Graphics:

Eg: Color.red

UI.clearGraphics() UI.setColor(*color*) UI.setLineWidth(*width*)

UI.drawRect(*left, top, wd, ht*) UI.fillRect(*left, top, wd, ht*)

UI.drawOval(*left, top, wd, ht*) UI.fillOval(*left, top, wd, ht*)

UI.drawLine(x_1, y_1, x_2, y_2)

UI.drawImage(*file, left, top*)

……

Lect #6

- Programming with Graphical Output - drawing.

Writing the program

- Need import statements
- Need a class (with a descriptive comment)
- Need a method (with a descriptive comment)

```
import  ecs100.*;
import  java.awt.Color;

/** Draws little shapes on the graphics pane */
public class  Drawer {

    /** Draw an red lollipop with a stick */
    public void drawLollipop() {
        // actions
    }
}
```

Write the method body as comments first

Writing the program: using comments

```
import  ecs100.*;
import java.awt.Color ;

/** Draws little pictures on the graphics pane */
public class  Drawer {

    /** Draw an red lollipop on a stick */
    public void drawLollipop() {
        // set line width to 10
        // draw line          (300,200)  to  (300, 400)
        // set line width to 1
        //  set color to red
        // fill oval          @(260,160)   80x80
    }
}
```

Do it in BlueJ!

Now express each comment in Java (look up documentation as necessary)

Read the Documentation!

- Full documentation for all the standard Java library code
  (the "API" : Application Programming Interface)

- Version of Java API documentation on course web site:
  - "Java Documentation" in side bar
  - http://ecs.victoria.ac.nz/foswiki/pub/Main/JavaResources/javaAPI-102.html

- Tailored for Comp 102
  - Includes documentation of the ecs100 library: (UI, Trace, etc,)
  - puts most useful classes at the top of the list.

- Use the documentation while you are programming!
  - Control-space in Bluej brings up the options plus documentation.

Writing the program

```
import ecs100.*;
import java.awt.Color ;

/** Draws little pictures on the graphics pane */
public class  Drawer {
    /** Draw a lollipop */
    public void drawLollipop() {
        UI.setLineWidth(10);                // set line width to 10
        UI.drawLine(300, 200, 300, 400);    // draw line
        UI.setLineWidth(1);                 // set line width back to 1
        UI.setColor(Color.red);             // set color to red
        UI.fillOval(260, 160, 80, 80);      // draw blob
    }
}
```

Now compile and run the program

Improving the design

- This program is very inflexible:
- What if
  - We want the lollipop to be in a different position?
  - We want the lollipop to be bigger or smaller?
  - We want the stick to be longer?
  - ….
  - We want to draw two of them?
- Current design is filled with literal values

  ⇒ difficult to understand

  ⇒ difficult to change
  (have to find all the places and redo all the arithmetic

Move or resize the Lollipop.

```
import ecs100.*;
import java.awt.Color ;

/** Draws little pictures on the graphics pane */
public class  Drawer {

    /** Draw a lollipop */
    public void doDrawLollipop() {
        UI.setColor(Color.black);          // set color to black
        UI.setLineWidth(10);               // set line width to 10
        UI.drawLine(300, 200, 300, 400);   // draw line
        UI.setLineWidth(1);                // set line width back to 1
        UI.setColor(Color.red);            // set color to red
        UI.fillOval(260, 160, 80, 80);     // draw blob
    }
}
```

Move it left

Move it down

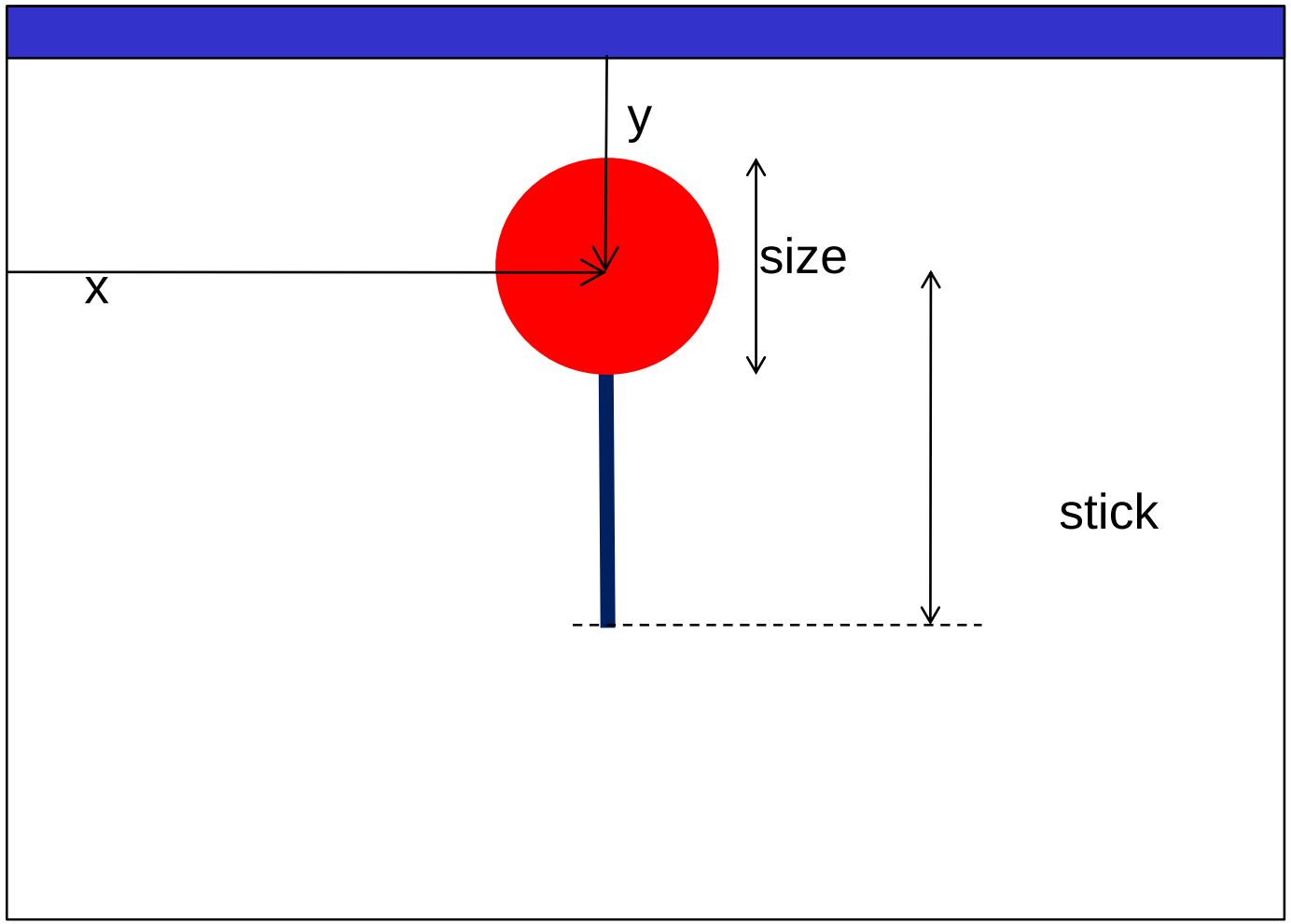
Change blob size

Improving the design

- Better design: Use named constants and variables
  - ⇒ easier to write and easier to change
  - ⇒ get the computer to do the arithmetic

- Use named constants for values that won't change while the program is running.

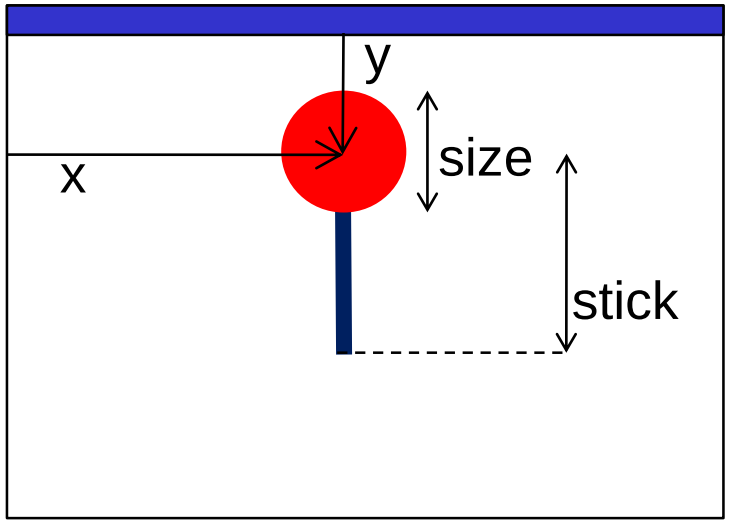
Values to specify lollipop & stick



Improving the program: constants

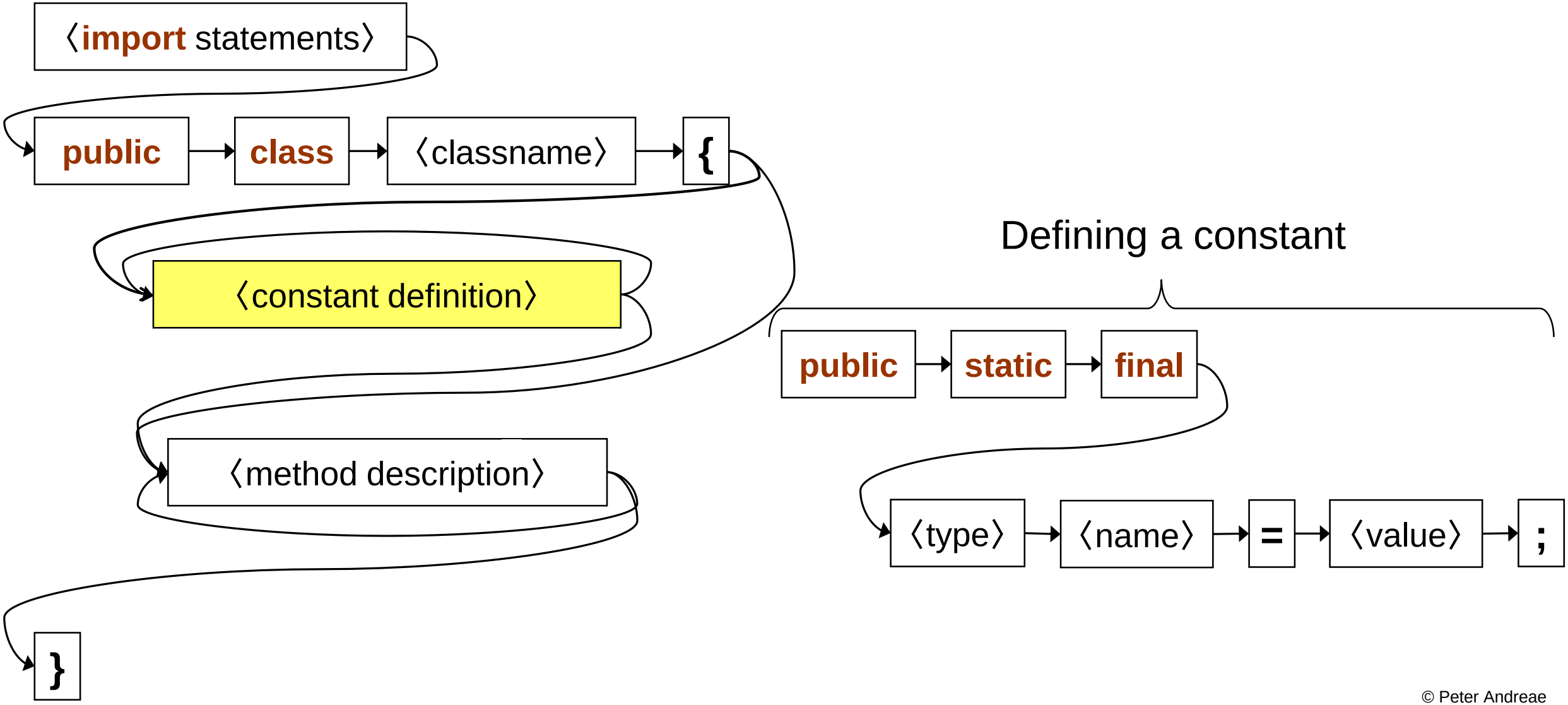
```
import  ecs100.*;    import  java.awt.Color;
/** Draw a lollipop with a stick */
public class  Drawer {
    public static final double  X = 300.0;        // horizontal center of lollipop
    public static final double  Y = 180.0;        // vertical center of lollipop
    public static final double  SIZE = 80.0;      // diameter of lollipop
    public static final double  STICK =  200.0;         // length of lollipop stick

    /** Draw a lollipop */
    public void doDrawLollipop() {
        UI.setLineWidth(SIZE/8.0);
        UI.drawLine(X, Y, X, Y+STICK);
        UI.setLineWidth(1);
        UI.setColor(Color.red);
        UI.fillOval(X-SIZE/2.0,  Y-SIZE/2.0,  SIZE, SIZE);
    }
}
```

Easy to change:
one place!



Syntax rules: Program structure

- 2nd version

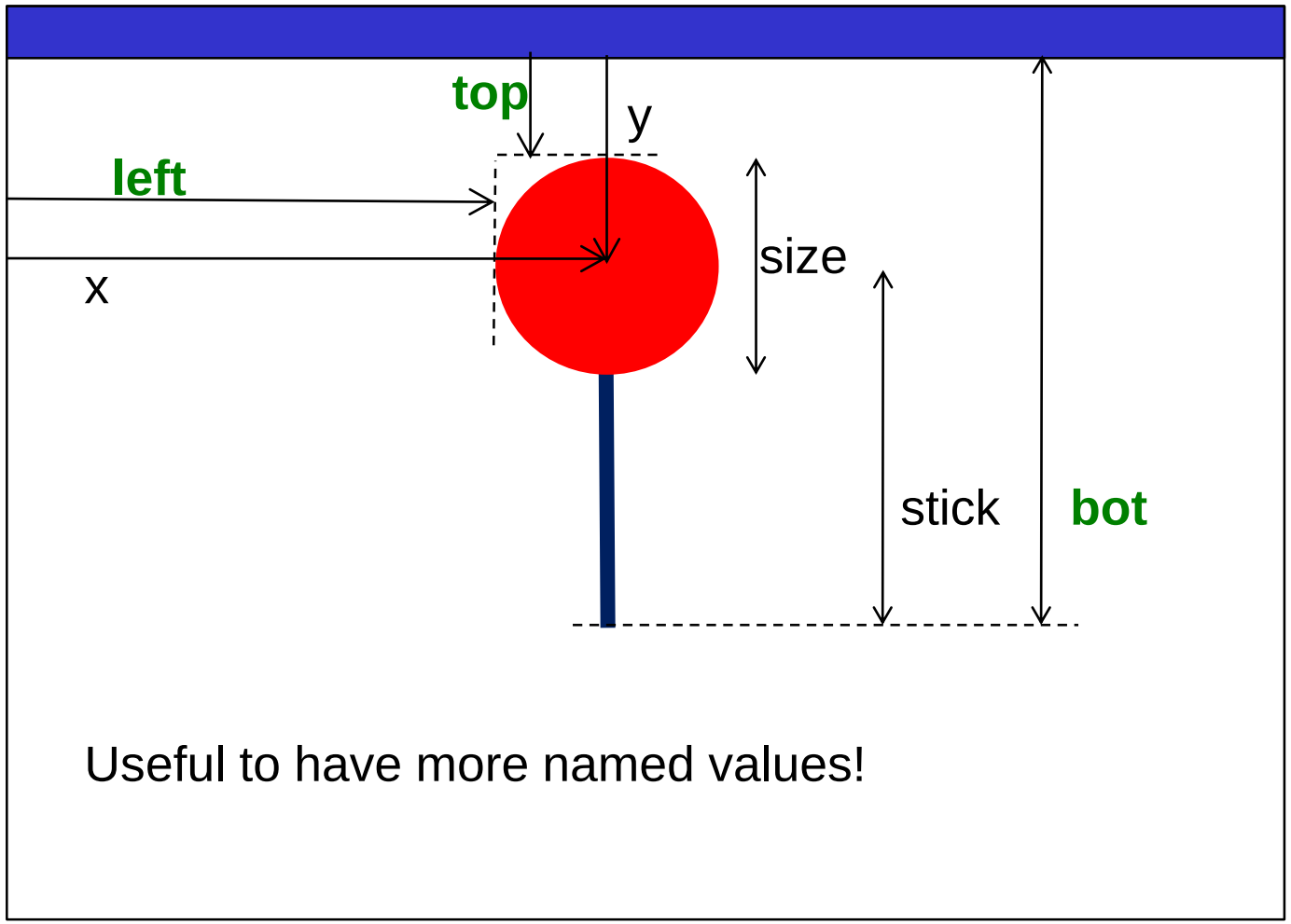


Improving the program: more names

```
public static final double X = 300.0;          // horizontal center of lollipop
public static final double Y = 180.0;          // vertical center of lollipop
public static final double SIZE = 80.0;        // diameter of lollipop
public static final double STICK =  200.0;// length of lollipop stick


/** Draw a lollipop */
public void doDrawLollipop() {
    UI.setLineWidth(10);
    UI.drawLine(X, Y, X, Y+STICK);
    UI.setLineWidth(1);
    UI.setColor(Color.red);
    UI.fillOval(X-SIZE/2.0, Y-SIZE/2.0, SIZE, SIZE);
}
```

Still have a problem:
What do these expressions mean?

Values to specify lollipop & stick



Improving the program: variables

```
public static final double X = 300.0;          // horizontal center of lollipop
public static final double Y = 180.0;          // vertical center of lollipop
public static final double SIZE = 80.0;        // diameter of lollipop
public static final double STICK =  200.0;    // length of lollipop stick


/** Draw a lollipop */
public void doDrawLollipop() {
    double left = X – SIZE/2.0;          // left of lollipop
    double top = Y – SIZE/2.0;           // top of lollipop
    double bot =  Y + STICK;             // bottom of stick
    UI.setLineWidth(10);
    UI.drawLine(X, Y, X, bot);
    UI.setLineWidth(1);
    UI.setColor(Color.red);
    UI.fillOval(left, top, SIZE, SIZE);
}
```

Principle of good design

- Use well named constants or variables wherever possible, rather than literal values

  ⇒ easier to understand

  ⇒ easier to get right

  ⇒ much easier to modify

- Choosing the *right* constants or variables is an art!!
  - why did I choose “x" instead of “left" ?
  - why did I choose “y" instead of stick bottom?
- We have effectively *parameterised* the drawing
  - Four values (X, Y, SIZE, STICK) control the whole thing.

Even better design: parameters

- Every time we want a lollipop of a different size or in a different position, we have to modify the code.
- How come we don't have to do that with drawRect?
- drawRect has four parameters:

Definition of drawRect:

```
public void drawRect(double left, double top, double wd, double ht) {……}
```

Parameters

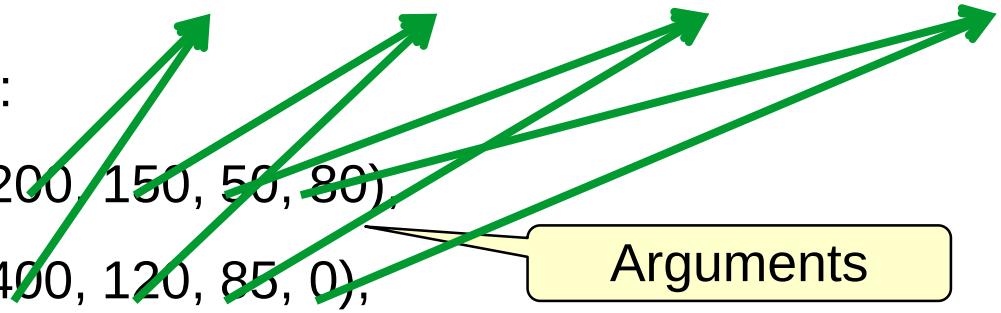
In the library files

Calling drawRect:

```
UI.drawRect(200, 150, 50, 80);
UI.drawRect(400, 120, 85, 0);
```

Arguments

In our program

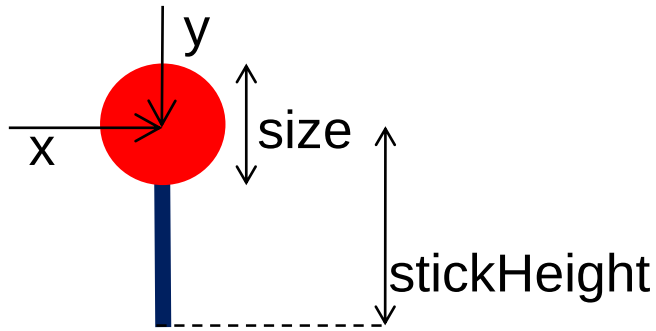


⇒ drawRect can make many different rectangles.

Why can't we do that with lollipop?

Improving the program: using parameters

```
/** Draw a lollipop at  (300, 180), asking the user for its size */
public void doDrawLollipop() {
    double size = UI.askDouble("Diameter:");
    double stickHeight =  UI.askDouble("Stick height");
    this.drawLollipop(300, 180, size, stickHeight);
}

public void drawLollipop(double x, double y, double size, double stick) {
    double left = x – size/2.0;          // left of lollipop
    double top = y – size/2.0;           // top of lollipop
    double bot =  y + stick;             // bottom of stick
    UI.setLineWidth(10);
    UI.drawLine(x, y, x, bot);
    UI.setLineWidth(1);
    UI.setColor(Color.red);
    UI.fillOval(left, top, size, size);
}
```



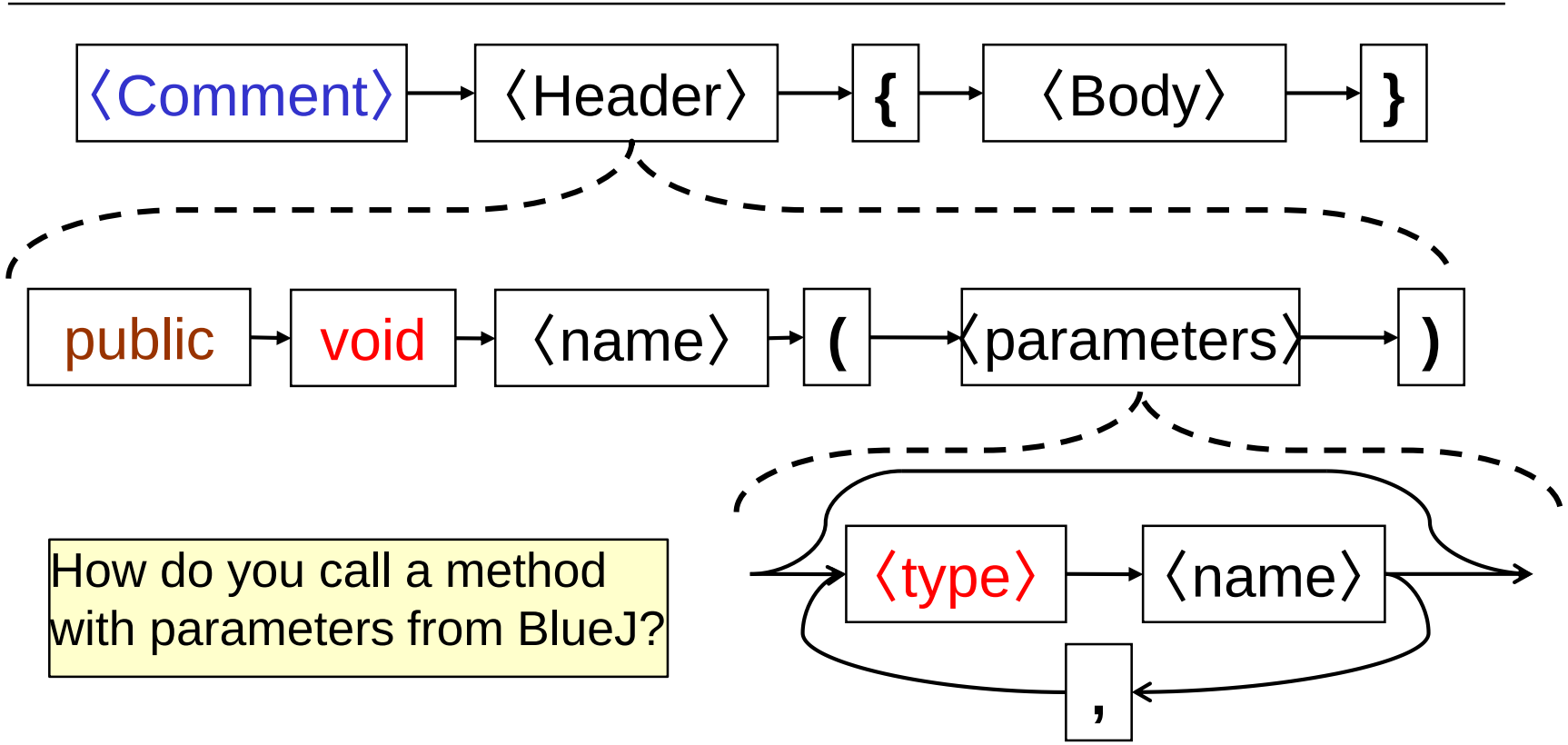
Parameters

Special variables which are given values each time the method is called.

Body of method can use the values in the parameters

Syntax: Method Definitions (v2)

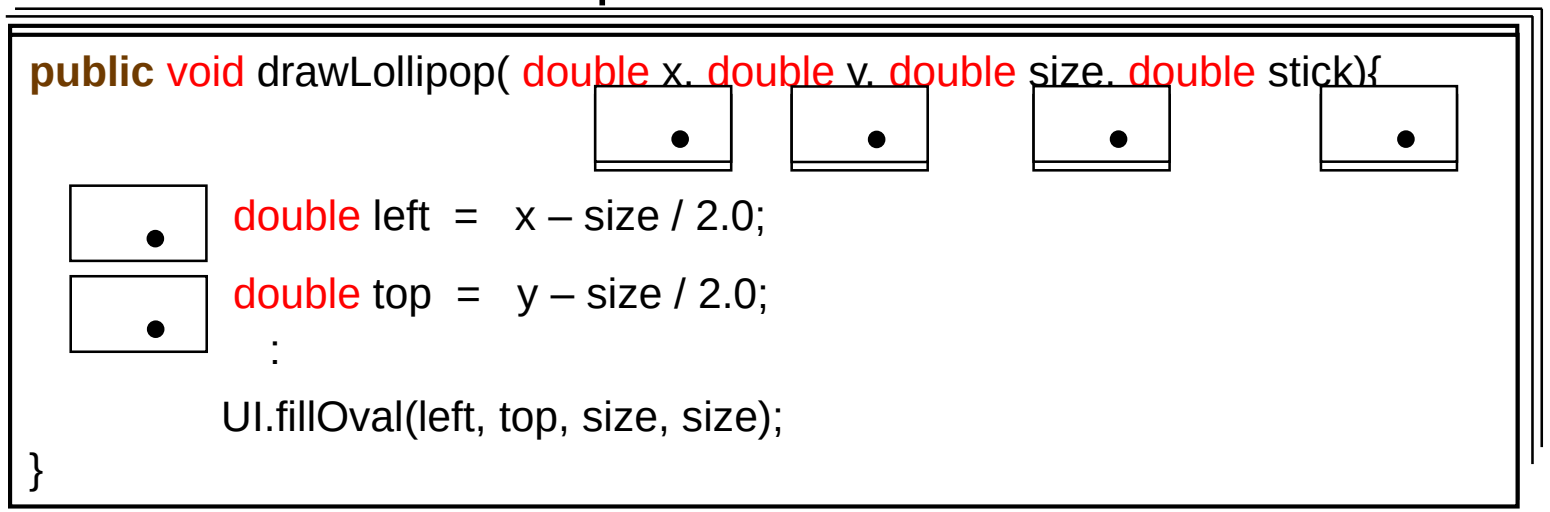
```
/** Draw a lollipop on a stick */
public void drawLollipop(double x, double y, double size, double stick ){
    double left = x – size/ 2.0;
            :
```



How do you call a method with parameters from BlueJ?

Method Calls with parameters

Method Definition: Like a pad of worksheets



Calling a Method:

this.drawLollipop(300, 100, 75, 95);

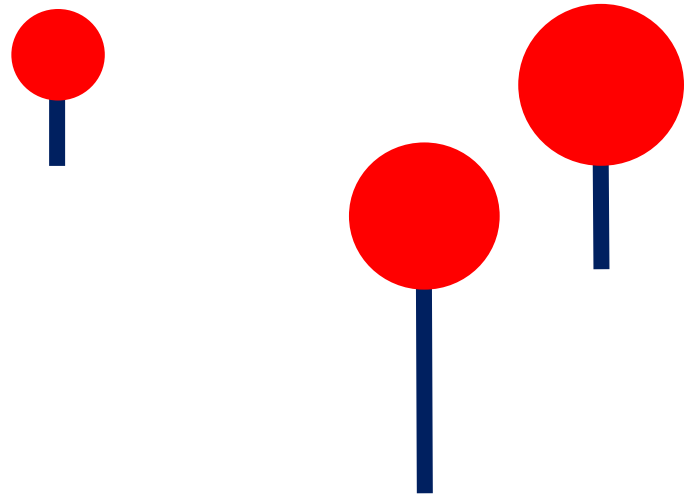
⇒ get a “copy” of the method worksheet

⇒ copy the arguments to the parameter places

⇒ perform each action in the body

⇒ throw the worksheet away (losing all the information on it)

Calling drawLollipop



```
public class  Drawer {
    public void doDrawLollipops() {
        double diam = UI.askDouble("diameter:");
        this.drawLollipop(300, 180, diam, 200);
        this.drawLollipop(50, 60, diam/2.0, 90);
        this.drawLollipop(400, 100, diam, 70);
    }
    /** Draw a lollipop */
    public void drawLollipop(double x, double y, double size, double stick) {
        double left = x – size/2.0;                 // left of lollipop
        double top = y – size/2.0;                  // top of lollipop
        double bot =  y + stick;                    // bottom of stick
        UI.setLineWidth(10);
        UI.drawLine(x, y, x, bot);
        UI.setLineWidth(1);
        UI.setColor(Color.red);
        UI.fillOval(left, top, size, size);
    }
```

Principle of good design

- Parameterising a method makes it more flexible and general
  - Allows us to call the same method with different arguments to do the same thing in different ways
  - Allows us to reuse the same bit of code