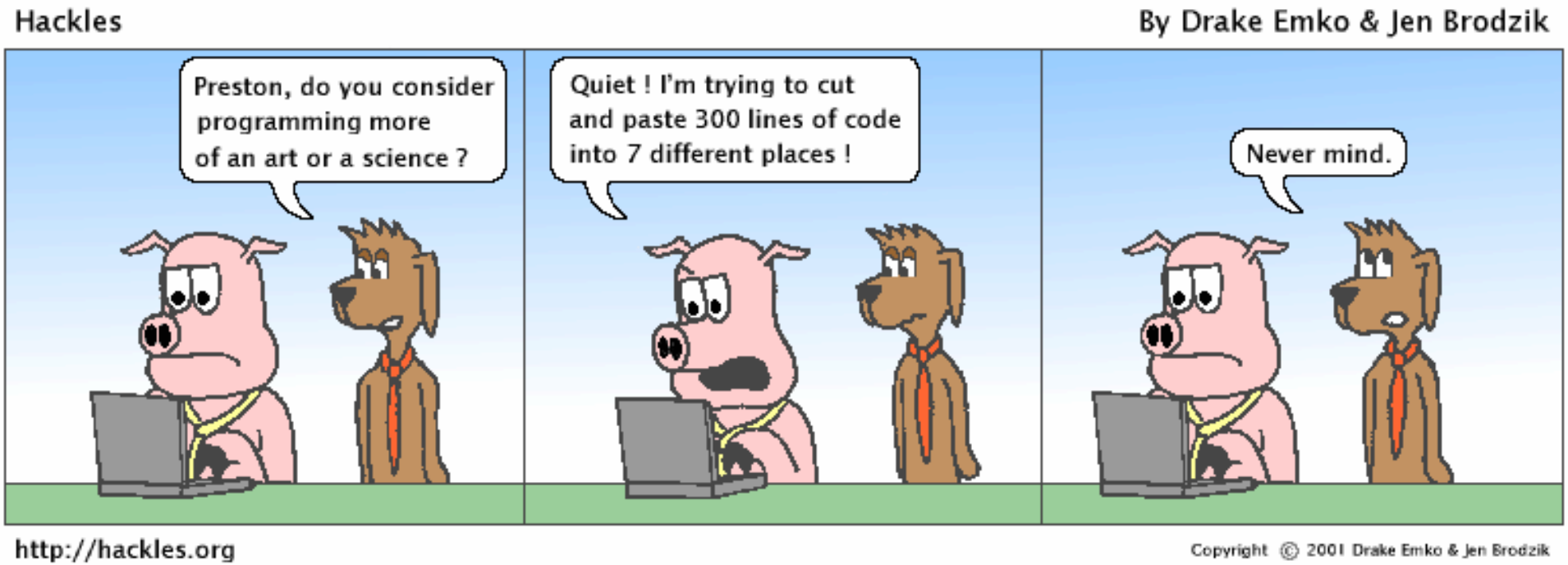
CS& 141, Winter 2021

Lecture 2: Graphics; Expressions and Variables



Data and expressions

Data types

- Internally, computers store everything as 1s and 0s

```
104    → 01101000
"hi"   → 0110100001101001
h      → 01101000
```

- How are `h` and `104` differentiated?

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types
  - Examples: integer, real number, string

Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

Name	Description		Examples
`int`	integers	(up to $2^{31} - 1$)	`42, -3, 0, 926394`
`double`	real numbers	(up to 10^{308})	`3.1, -0.25, 9.4e3`
`char`	single text characters		`'a', 'X', '?', '\n'`
`boolean`	logical values		`true, false`

  - Why does Java distinguish integers vs. real numbers?

Expressions

- **expression**: A value or operation that computes a value.
  - Examples:

```
1 + 4 * 5
(7 + 2) * 6 / 3
42
```

  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.

Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - `+` addition
  - `-` subtraction (or negation)
  - `*` multiplication
  - `/` division
  - `%` modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.
  - `1 + 1` evaluates to `2`
  - `System.out.println(3 * 4);` prints `12`
    - How would we print the text `3 * 4` ?

Integer division with /

- When we divide integers, the quotient is also an integer.
  - `14 / 4` is `3`, not `3.5`

```
     3              4              52
4 ) 14         10 ) 45        27 ) 1425
    12             40             135
     2              5               75
                                     54
                                     21
```

- More examples:
  - `32 / 5` is `6`
  - `84 / 10` is `8`
  - `156 / 100` is `1`

  - Dividing by 0 causes an error when your program runs.

Integer remainder with %

- The % operator computes the remainder from integer division.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
      3
4 ) 14
    12
     2
```

```
      43
5 ) 218
    20
     18
     15
      3
```

What is the result?

```
45 % 6
2 % 2
8 % 20
11 % 0
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

    `1 + 3 * 4` is `13`

    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12
    ```
    is `18`

  - Parentheses can force a certain order of evaluation:

    `(1 + 3) * 4` is `16`

  - Spacing does not affect order of evaluation

    `1+3 * 4-2` is `11`

Precedence examples

```
1 * 2 + 3 * 5 % 4
 \_/
  |
  2   + 3 * 5 % 4
          \_/
           |
  2   +  15   % 4
           \___/
             |
  2   +     3
   \_________/
        |
        5
```

```
1 + 8 % 3 * 2 - 9
      \_/
       |
1 +   2   * 2 - 9
        \___/
          |
1 +       4     - 9
 \_______/
     |
     5           - 9
      \_________/
          |
         -4
```

Real numbers (type `double`)

- Examples: `6.022` , `-42.0` , `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.
- The operators `+ - * / % ()` all still work with `double`.
  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`
  - Precedence is the same: `()` before `* / %` before `+ -`

Real number example

```
2.0 * 2.4 + 2.25 * 4.0 / 2.0
  \____/
    |
  4.8     + 2.25 * 4.0 / 2.0
                \____/
                  |
  4.8     +     9.0    / 2.0
                  \______/
                     |
  4.8     +          4.5
     \______________/
            |
          9.3
```

Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`
- The conversion is per-operator, affecting only its operands.

```
7 / 3 * 1.2 + 3 / 2
 \_/
  |
  2   * 1.2 + 3 / 2
   \_____/
     |
    2.4      + 3 / 2
                \_/
                 |
    2.4      +   1
      \_____________/
            |
           3.4
```

```
2.0 + 10 / 3 * 2.5 - 6 / 4
          \_____/
             |
2.0 +     3    * 2.5 - 6 / 4
           \_______/
              |
2.0 +       7.5      - 6 / 4
                         \_/
                          |
2.0 +       7.5      -   1
  \_____________/
        |
      9.5            -   1
        \_____________________/
                 |
                8.5
```

- `3 / 2` is `1` above, not `1.5`.

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42      is "hello42"
1 + "abc" + 2     is "1abc2"
"abc" + 1 + 2     is "abc12"
1 + 2 + "abc"     is "3abc"
"abc" + 9 * 3     is "abc27"
"1" + 1           is "11"
4 - 1 + "abc"     is "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
  - Output: `Grade: 83.5`

Variables

Receipt example

What's bad about the following code?

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  type name;

  - The name is an *identifier*.

  - `int zipcode;`

  - `double myGPA;`

Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**;

  - `int zipcode;`
    `zipcode = 90210;`

zipcode	90210

  - `double myGPA;`
    `myGPA = 1.0 + 2.25;`

myGPA	3.25

Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
System.out.println("x is " + x);      // x is 3

System.out.println(5 * x - 1);        // 5 * 3 - 1
```

- You can assign a value more than once:

x	11

```
int x;
x = 3;
System.out.println(x + " here");      // 3 here

x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```

Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

  type name = **value**;

  - `double myGPA = 3.95;`

myGPA	3.95

  - `int x = (11 % 3) + 12;`

x	14

Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.
  - = means, *"store the value at right in variable at left"*
    - The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

```
int x = 3;
x = x + 2;    // ???
```

x	5

Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;  // ERROR: incompatible types`

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`

myGPA	4.0

  - `double avg = 11 / 2;`
    - Why does `avg` store `5.0` and not `5.5` ?

avg	**5.0**

Compiler errors

- A variable can't be used until it is assigned a value.
  - ```
    int x;
    System.out.println(x);    // ERROR: x has no value
    ```
- You may not declare the same variable twice.
  - ```
    int x;
    int x;                    // ERROR: x already exists
    ```
  - ```
    int x = 3;
    int x = 5;                // ERROR: x already exists
    ```
    - How can this code be fixed?
    ```

Printing a variable's value

- Use + to print a string and a variable's value on one line.
  - ```
    double grade = (95.1 + 71.9 + 82.6) / 3.0;
    System.out.println("Your grade was " + grade);

    int students = 11 + 17 + 4 + 19 + 14;
    System.out.println("There are " + students +
                       " students in the course.");
    ```
  - Output:

    ```
    Your grade was 83.2
    There are 65 students in the course.
    ```
    ```

Receipt question

Improve the receipt program using variables.

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);

        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);

        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);

        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .15 +
                           (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

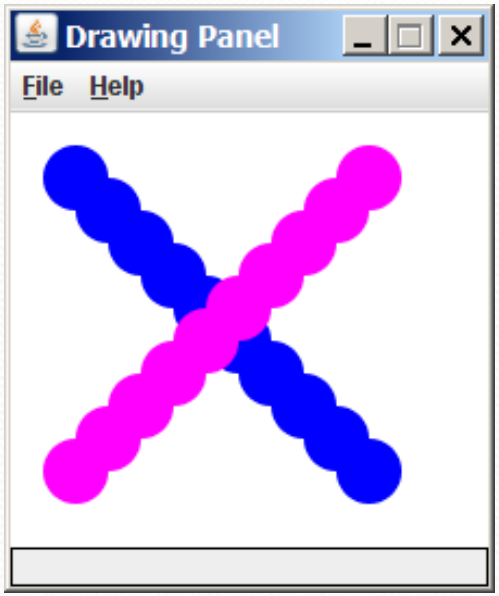
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```

Graphics

Graphical objects

We will draw graphics in Java using 3 kinds of objects:

- `DrawingPanel`: A window on the screen.
  - Not part of Java; provided by the instructor. See class web site.
- `Graphics`: A "pen" to draw shapes and lines on a window.
- `Color`: Colors in which to draw shapes.



DrawingPanel

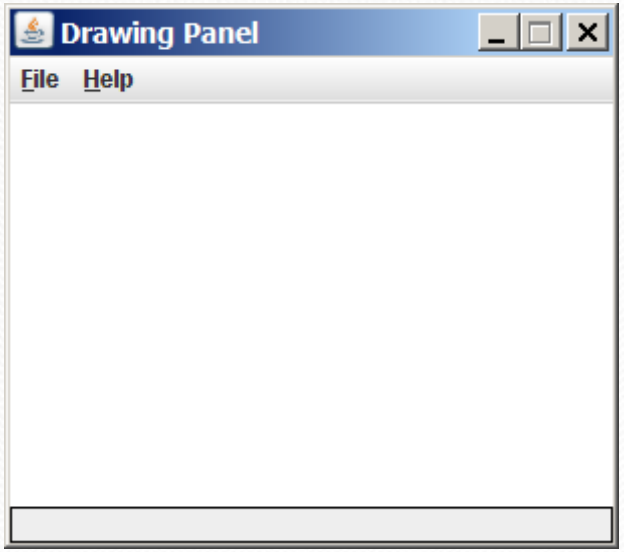
- To create a window:

  `DrawingPanel` ***<name>*** `= new DrawingPanel(`***<width>***`,` ***<height>***`);`

  Example:

  ```
  DrawingPanel panel = new DrawingPanel(300, 200);
  ```

- The window has nothing on it.
  - We can draw shapes and lines on it using another object of type `Graphics`.



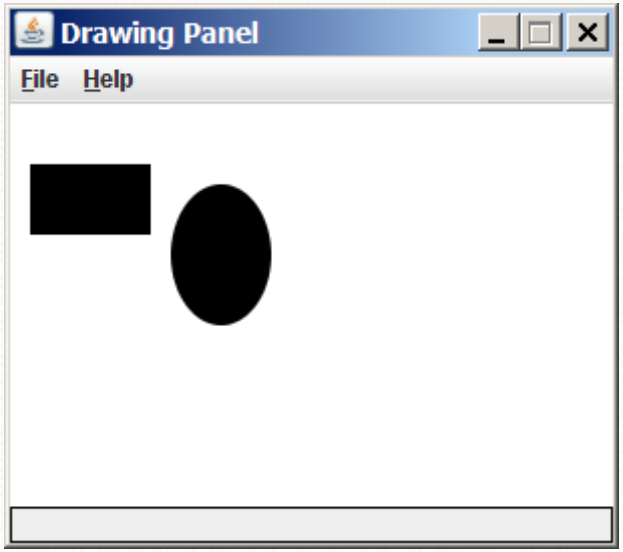
Graphics

- Shapes are drawn using an object of class `Graphics`.
  - You must place an import declaration in your program:
    `import java.awt.*;`
  - Access it by calling `getGraphics` on your `DrawingPanel`.

```
Graphics g = panel.getGraphics();
```

- Draw shapes by calling methods on the `Graphics` object.

```
g.fillRect(10, 30, 60, 35);
g.fillOval(80, 40, 50, 70);
```

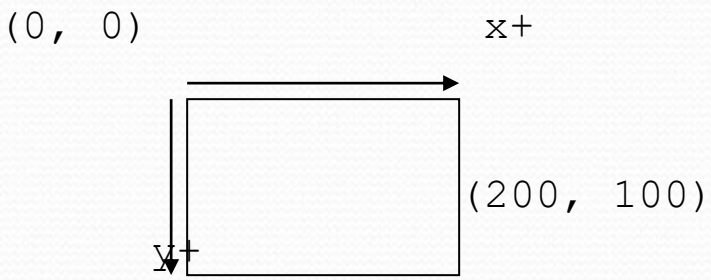


Graphics methods

Method name	Description
`g.drawLine(`*x1*, *y1*, *x2*, *y2*`);`	line between points (*x1*, *y1*), (*x2*, *y2*)
`g.drawOval(`*x*, *y*, *width*, *height*`);`	outline largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.drawRect(`*x*, *y*, *width*, *height*`);`	outline of rectangle of size *width* * *height* with top-left at (*x*, *y*)
`g.drawString(`*text*, *x*, *y*`);`	text with bottom-left at (x, y)
`g.fillOval(`*x*, *y*, *width*, *height*`);`	fill largest oval that fits in a box of size *width* * *height* with top-left at (*x*,*y*)
`g.fillRect(`*x*, *y*, *width*, *height*`);`	fill rectangle of size *width* * *height* with top-left at (*x*, *y*)
`g.setColor(`*Color*`);`	set `Graphics` to paint any following shapes in the given color

Coordinate system

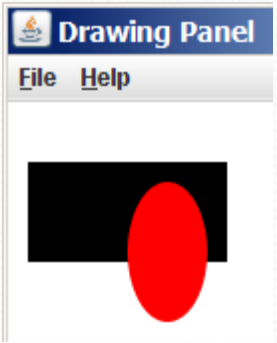
- Each (x, y) position is a *pixel* ("picture element").
- (0, 0) is at the window's top-left corner.
  - x increases rightward and the y increases <u>downward</u>.
- The rectangle from (0, 0) to (200, 100) looks like this:



Colors

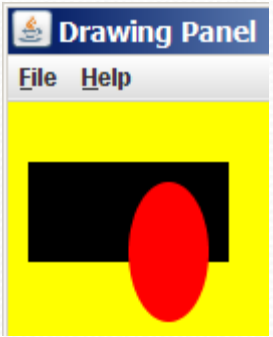
- Colors are specified by `Color` class constants named:
  `BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, `YELLOW`
  - Pass to `Graphics` object's `setColor` method:

```
g.setColor(Color.BLACK);
g.fillRect(10, 30, 100, 50);
g.setColor(Color.RED);
g.fillOval(60, 40, 40, 70);
```



- The background color can be set by calling `setBackground` on the `DrawingPanel`:

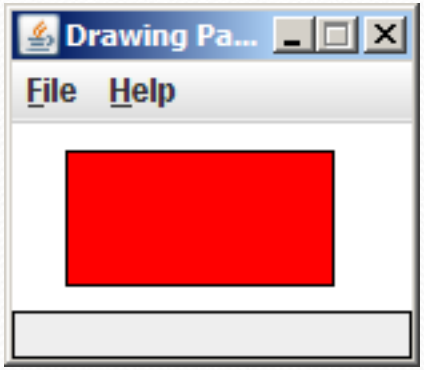
```
panel.setBackground(Color.YELLOW);
```



Outlined shapes

- To draw a shape with a fill and outline, first *fill* it in the fill color and then *draw* the same shape in the outline color.

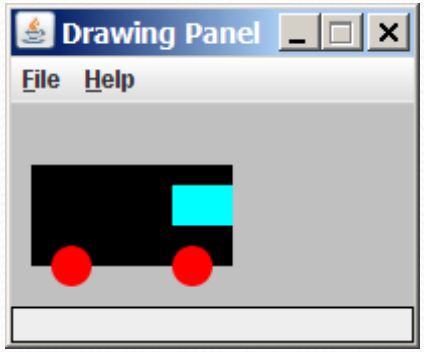
```
import java.awt.*;  // so I can use Graphics

public class DrawOutline {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(150, 70);
        Graphics g = panel.getGraphics();

        // inner red fill
        g.setColor(Color.RED);
        g.fillRect(20, 10, 100, 50);

        // black outline
        g.setColor(Color.BLACK);
        g.drawRect(20, 10, 100, 50);
    }
}
```



Superimposing shapes

- When two shapes occupy the same pixels, the last one drawn is seen.

```
import java.awt.*;

public class DrawCar {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();

        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);

        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);

        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```



Repetition with `for` loops

- So far, repeating an action results in redundant code:

```
makeBatter();
bakeCookies();
bakeCookies();
bakeCookies();
bakeCookies();
bakeCookies();
frostCookies();
```

- Java's **`for` loop** statement performs a task many times.

```
mixBatter();

for (int i = 1; i <= 5; i++) {    // repeat 5 times
    bakeCookies();
}

frostCookies();
```

`for` loop syntax

```
for (initialization; test; update) {
    statement;
    statement;
    ...
    statement;
}
```

header

body

- Perform **initialization** once.
- Repeat the following:
  - Check if the **test** is true. If not, stop.
  - Execute the **statement**s.
  - Perform the **update**.

Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution
- Controlled code may include one or more statements
- The for loop is an example of a looping control structure

Initialization

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Tells Java what variable to use in the loop
  - The variable is called a *loop counter*
    - can use any name, not just `i`
    - can start at any value, not just `1`
    - only valid in the loop
  - Performed once as the loop begins

Test

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Tests the loop counter variable against a limit
  - Uses comparison operators:

    `<` less than

    `<=` less than or equal to

    `>` greater than

    `>=` greater than or equal to

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable++;	**variable** = **variable** + 1;
variable--;	**variable** = **variable** – 1;

```
int x = 2;
x++;                    // x = x + 1;
                        // x now stores 3

double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                        // gpa now stores 1.5
```

Modify-and-assign operators

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += **value**;	**variable** = **variable** + **value**;
variable -= **value**;	**variable** = **variable** - **value**;
variable *= **value**;	**variable** = **variable** * **value**;
variable /= **value**;	**variable** = **variable** / **value**;
variable %= **value**;	**variable** = **variable** % **value**;

```
x += 3;                    // x = x + 3;

gpa -= 0.5;                // gpa = gpa - 0.5;

number *= 2;               // number = number * 2;
```

Repetition over a range

```
System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

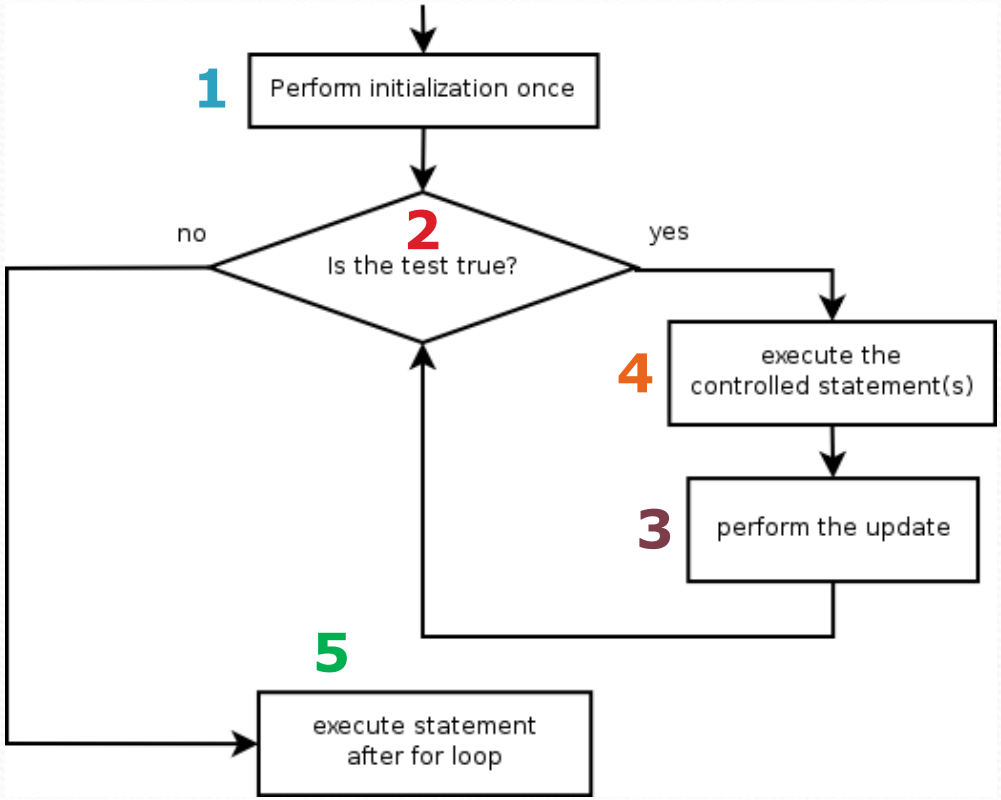
- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
          1          2        3
for (int i = 1; i <= 4; i++) {
   4 System.out.println(i + " squared = " + (i * i));
}
5 System.out.println("Whoo!");
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```



System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line

```
int highestTemp = 5;
for (int i = -3; i <= highestTemp / 2; i++) {
    System.out.print((i * 1.8 + 32) + "  ");
}
```

  - Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

    - Concatenate `"  "` to separate the numbers

Rocket Exercise

- Write a method that produces the following output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```

Counting down

- The **update** can use -- to make the loop count down.
  - The **test** must say > instead of <

```
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
     System.out.print(i + ", ");
}
System.out.println("blastoff!");
System.out.println("The end.");
```

  - Output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```