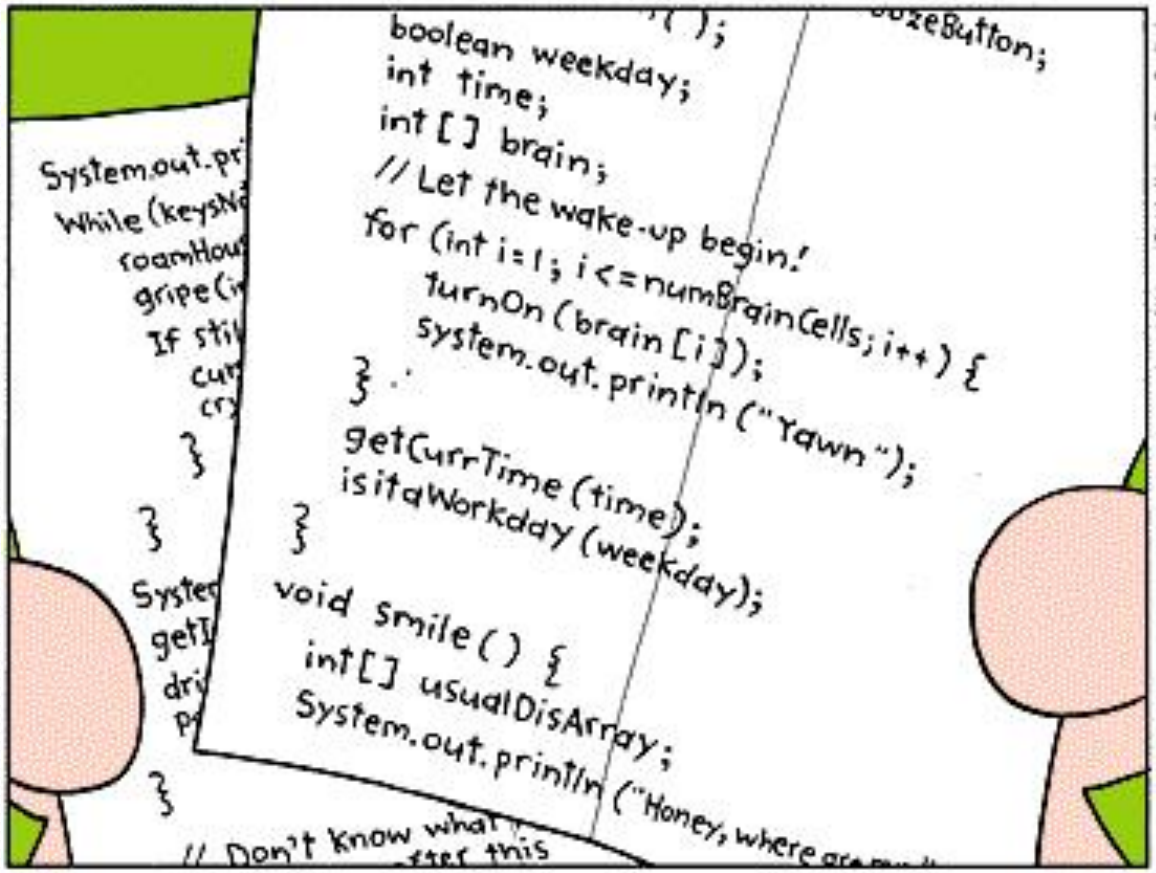


Building Java Programs

static methods, constants

reading: 1.4 - 1.5, 2.4 – 2.5, 3.1 - 3.2

Declaring a method

Gives your method a name so it can be executed

- Syntax:

```
public static void name() {
    statement;
    statement;
    ...
    statement;
}
```

- Example:

```
public static void printWarning() {
    System.out.println("This product causes cancer");
    System.out.println("in lab rats and humans.");
}
```

Calling a method

Executes the method's code

- Syntax:

```
name();
```

  - You can call the same method many times if you like.

- Example:

```
printWarning();
```

  - Output:

```
This product causes cancer
in lab rats and humans.
```

Program with static method

```
public class FreshPrince {
    public static void main(String[] args) {
        rap();                      // Calling (running) the rap method
        System.out.println();
        rap();                      // Calling the rap method again
    }

    // This method prints the lyrics to my favorite song.
    public static void rap() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

Output:

```
Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
```

When to use methods

- Place statements into a static method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create static methods for:
  - An individual `println` statement.
  - Only blank lines. (Put blank `println`s in `main`.)
  - Unrelated or weakly related statements. (Consider splitting them into two smaller methods.)

Redundant figures

- Consider the task of printing the following lines/boxes:

```
**********

*******

**************************************

**********
*        *
**********

*****
*   *
*   *
*****
```

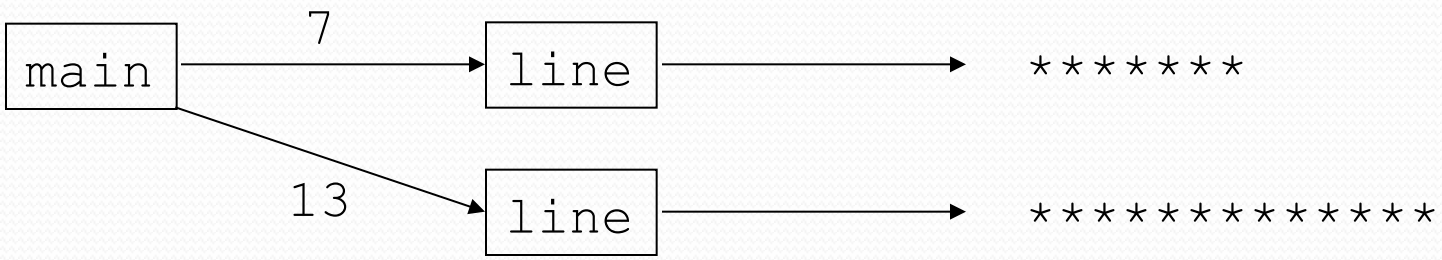
A redundant solution

```
public class Stars1 {
    public static void main(String[] args) {
        lineOf13();
        lineOf7();
        lineOf35();
        box10x3();
        box5x4();
    }

    public static void lineOf13() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf7() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf35() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

- This code is redundant.
- Would variables help?
- What is a better solution?
  - `line` - A method to draw a line of any number of stars.
  - `box` - A method to draw a box of any size.

Parameterization

- **parameter**: A value passed to a method by its caller.
  - Instead of `lineOf7`, `lineOf13`, write `line` to draw any length.
    - When *declaring* the method, we will state that it requires a parameter for the number of stars.
    - When *calling* the method, we will specify how many stars to draw.



Declaring parameters

- A method can accept multiple parameters. (separate by ,)
  - When calling it, you must pass values for each parameter.

- Java:

```
public static void <name>(<type> <name>, ..., <type> <name>) {
    <statement>(s);
}
```

- Python:

```
def <name>(<name>, ..., <name>) {
    <statement>(s)
```

Passing parameters

Calling a method and specifying values for its parameters

- Java:

```
<name>(<expression>, … , <expression>);
```

- Python:

```
<name>(<expression>, … , <expression>)
```

The same in both languages!

Parameters example

```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717

00000000
```

Stars solution

```
// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

Stars solution, cont'd.

```
...

    // Prints a box of stars of the given size.
    public static void box(int width, int height) {
        line(width);

        for (int line = 1; line <= height - 2; line++) {
            System.out.print("*");
            for (int space = 1; space <= width - 2; space++) {
                System.out.print(" ");
            }
            System.out.println("*");
        }

        line(width);
    }
}
```

Returning a value

```
public static type name(parameters) {
    statements;
    ...
    return expression;
}
```

- When Java reaches a return statement:
  - it evaluates the expression
  - it substitutes the return value in place of the call
  - it goes back to the caller and continues after the method call

Return examples

```
// Converts degrees Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}
```

- You can shorten the examples by returning an expression:

```
public static double fToC(double degreesF) {
    return 5.0 / 9.0 * (degreesF - 32);
}
```

Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method.

```
public static void main(String[] args) {
    slope(0, 1, 6, 3);
    System.out.println("The slope is " + result);  // ERROR:
}                                                  // cannot find symbol: result

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

Fixing the common error

- Returning sends the variable's *value* back. Store the returned value into a variable or use it in an expression.

```
public static void main(String[] args) {
    double s = slope(0, 0, 6, 3);
    System.out.println("The slope is " + s);
}

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

Exercise

Alter the BMI program from last time to add methods. Methods should contain:

- The statements are related structurally, and/or
- The statements are repeated.

```
Enter next person's information:
height (in inches)? 70.0
weight (in pounds)? 194.25

BMI = 27.868928571428572
overweight

Is there more information to enter? 42
```

Exercise solution

```
// This program computes people's body mass index (BMI) and
// compares them.  The code uses Scanner for input, and parameters/returns.
import java.util.*;  // so that I can use Scanner
public class Week3 {
    public static void main(String[] args) {
      intro();
      Scanner console = new Scanner(System.in);
      int countNormal = 0;
      int total = 0;
      int again = 42;
      while (again != 0) {
         total++;
         System.out.println("Enter next person's information:");
         System.out.print("height (in inches)? ");
         double height = console.nextDouble();
         System.out.print("weight (in pounds)? ");
         double weight = console.nextDouble();
         double bmi = calcBMI (weight, height);
         System.out.println("BMI = " + bmi);
         countNormal += printType(bmi);
         System.out.print("Is there more information to enter? ");
         again = console.nextInt();
      }
      stats(countNormal, total);
   }
   …
```

Exercise solution Contd.

```
public static void intro() {
    System.out.println("This program reads height and weight data and");
    System.out.println("Computes body mass index (BMI).");
 }
 public static double calcBMI (double weight, double height) {
    double bmi = weight / (height * height) * 703;
    return bmi;
 }
 public static void stats(int countNormal, int total) {
    System.out.println("Total count of people              = " +
             + total);
    System.out.println("Percentage of people normal weight = "
             + 1.0 * countNormal / total);
 }
 public static int printType(double bmi) {
    if (bmi < 18.5) {
       System.out.println("underweight");
    } else if (bmi < 25) {
       System.out.println("normal");
       return 1;
    } else if (bmi < 30) {
       System.out.println("overweight");
    } else {
       System.out.println("obese");
    }
    return 0;
 }
}
```

Class constants

- **class constant**: A fixed value visible to the whole program.
  - value can be set only at declaration; cannot be reassigned

- Syntax:

```
public static final type name = value;
```

  - name is usually in ALL_UPPER_CASE

  - Examples:

```
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 3.5;
public static final int SSN = 658234569;
```

Using a constant

```
public class Sign {
    public static final int HEIGHT = 5;

    public static void main(String[] args) {
        drawLine();
        drawBody();
        drawLine();
    }

    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= HEIGHT * 2; i++) {
            System.out.print("/\\");
        }
        System.out.println("+");
    }

    public static void drawBody() {
        for (int line = 1; line <= HEIGHT; line++) {
            System.out.print("|");
            for (int spaces = 1; spaces <= HEIGHT * 4; spaces++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```