

Building Java Programs

Chapter 3

Methods and Parameters

Copyright (c) Pearson 2013.
All rights reserved.

Methods

Reading

- Building Java Programs, Ch. 3.1 - 3.4, Ch. 4.3

Watch

- <https://youtu.be/C2D8geYMOqM>
- <https://youtu.be/s9iGMR0dvwU>
- <https://youtu.be/00SCvovaXxc>

Learning Outcomes

- Methods in Java
- Program control flow
- Method parameters
- Returning values from a method
- Java objects and object methods
- Math methods
- String methods

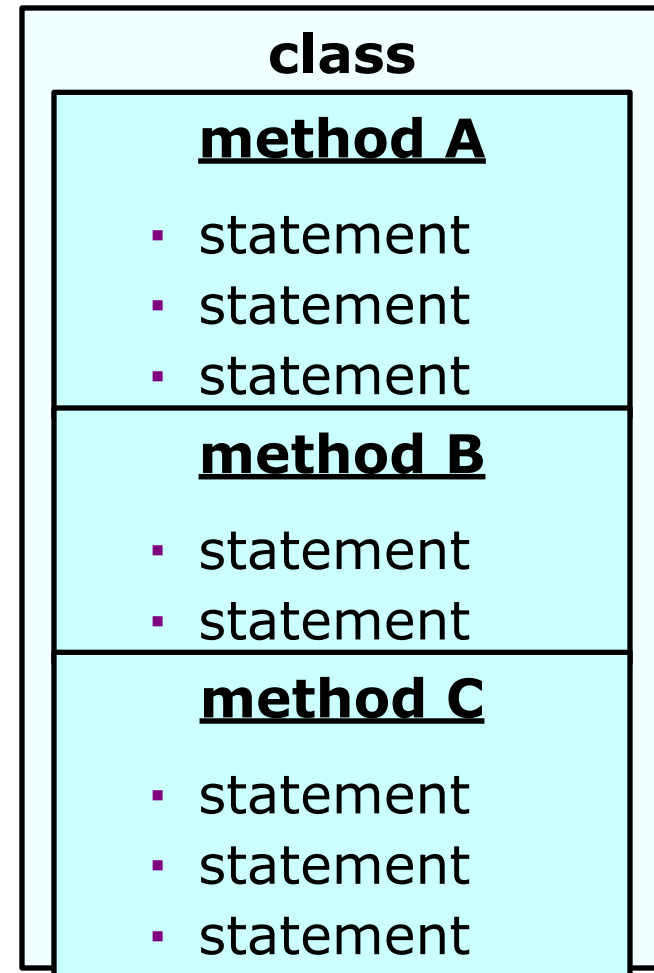
Static methods

- **static method:** A named group of statements.

- denotes the structure of a program
- eliminates redundancy by code reuse

– **procedural decomposition:**
dividing a problem into methods

- Writing a static method is like adding a new command to Java.



Using static methods

1. Design the algorithm.
 - Look at the structure, and which commands are repeated.
 - Decide what are the important overall tasks.
2. **Declare** (write down) the methods.
 - Arrange statements into groups and give each group a name.
3. **Call** (run) the methods.
 - The program's `main` method executes the other methods to perform the overall task.

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
```

Methods calling methods

```
public class MethodsExample {  
    public static void main(String[] args) {  
        message1();  
        message2();  
        System.out.println("Done with main.");  
    }  
    public static void message1() {  
        System.out.println("This is message1.");  
    }  
    public static void message2() {  
        System.out.println("This is message2.");  
        message1();  
        System.out.println("Done with message2.");  
    }  
}
```

- **Output:**

```
This is message1.  
This is message2.  
This is message1.  
Done with message2.  
Done with main.
```


Control flow

- When a method is called, the program's execution...
 - "jumps" into that method, executing its statements, then
 - "jumps" back to the point where the method was called.

```
public class MethodsExample {  
    public static void main(String[] args) {  
        message1() ;  
        message2() ;  
        System.out.println("Done with message2.");  
    }  
    ...  
}
```

```
public static void message1() {  
    System.out.println("This is message1.");  
}
```

```
public static void message2() {  
    System.out.println("This is message2.");  
    message1() ;  
    System.out.println("Done with message2.");  
}
```

```
public static void message1() {  
    System.out.println("This is message1.");  
}
```

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 recipes

- Recipe for baking **20** cookies:
 - Mix the following ingredients in a bowl:
 - **4** cups flour
 - **1** cup butter
 - **1** cup sugar
 - **2** eggs
 - **40** pounds chocolate chips ...
 - Place on sheet and Bake for about **10** minutes.
- Recipe for baking **40** cookies:
 - Mix the following ingredients in a bowl:
 - **8** cups flour
 - **2** cups butter
 - **2** cups sugar
 - **4** eggs
 - **80** pounds chocolate chips ...
 - Place on sheet and Bake for about **10** minutes.

Final cookie program

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies3 {
    public static void main(String[] args) {
        makeBatter();
        bake();           // 1st batch
        bake();           // 2nd batch
        decorate();
    }

    // Step 1: Make the cake batter.
    public static void makeBatter() {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
    }

    // Step 2: Bake a batch of cookies.
    public static void bake() {
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
    }

    // Step 3: Decorate the cookies.
    public static void decorate() {
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

Parameterized recipe

- Recipe for baking **20** cookies:
 - Mix the following ingredients in a bowl:
 - 4 cups flour
 - 1 cup sugar
 - 2 eggs
 - ...
- Recipe for baking **N** cookies:
 - Mix the following ingredients in a bowl:
 - $N/5$ cups flour
 - $N/20$ cups butter
 - $N/20$ cups sugar
 - $N/10$ eggs
 - $2N$ bags chocolate chips
 - Place on sheet and Bake for about 10 minutes.
- **parameter**: A value that distinguishes similar tasks.

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();
    }

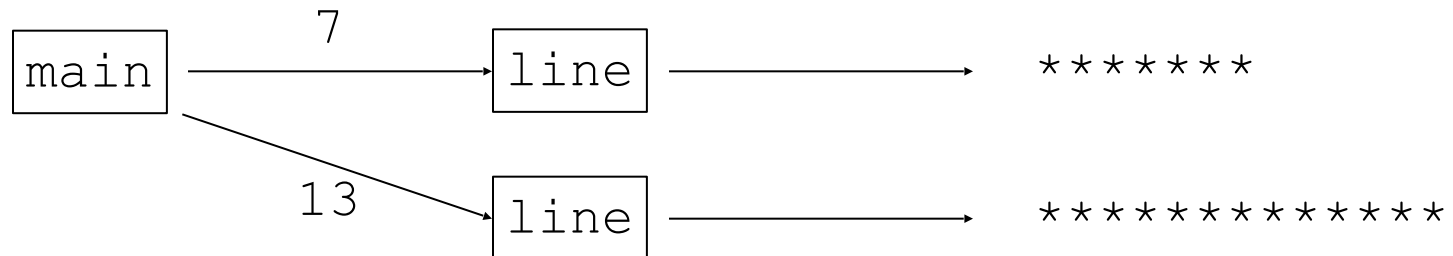
    public static void lineOf7() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

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

- This code is redundant.
- Would variables help?
Would constants 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 a parameter

Stating that a method requires a parameter in order to run

```
public static void name ( type name ) {  
    statement(s);  
}
```

- **Example:**

```
public static void sayPassword(int code) {  
    System.out.println("The password is: " + code);  
}
```

- When `sayPassword` is called, the caller must specify the integer code to print.

Passing a parameter

Calling a method and specifying values for its parameters

name (**expression**) ;

- Example:

```
public static void main(String[] args) {  
    sayPassword(42) ;  
    sayPassword(12345) ;  
}
```

Output:

```
The password is 42  
The password is 12345
```

Parameters and loops

- A parameter can guide the number of repetitions of a loop.

```
public static void main(String[] args) {  
    chant(3);  
}  
  
public static void chant(int times) {  
    for (int i = 1; i <= times; i++) {  
        System.out.println("Just a salad...");  
    }  
}
```

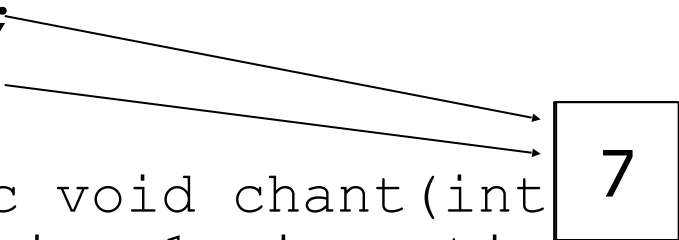
Output:

```
Just a salad...  
Just a salad...  
Just a salad...
```

How parameters are passed

- When the method is called:
 - The value is stored into the parameter variable.
 - The method's code executes using that value.

```
public static void main(String[] args) {  
    chant(3);  
    chant(7);  
}  
  
public static void chant(int times) {  
    for (int i = 1; i <= times; i++) {  
        System.out.println("Just a salad...");  
    }  
}
```



The diagram illustrates the passing of the value 7 from the `main` method to the `chant` method. Two arrows originate from the `chant(7);` call in the `main` method. One arrow points to the `times` parameter in the `chant` method signature, and the other points to a box containing the number 7, which represents the value being passed.

Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
chant();           // ERROR: parameter value  
required
```

- The value passed to a method must be of the correct type.

```
chant(3.7);        // ERROR: must be of type int
```

- Exercise: Change the `Stars` program to use a parameterized method for drawing lines of stars.

Stars solution

```
// Prints several lines of stars.  
// Uses a parameterized method to remove redundancy.  
public class Stars2 {  
    public static void main(String[] args) {  
        line(13);  
        line(7);  
        line(35);  
    }  
  
    // 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();  
    }  
}
```

Multiple parameters

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

- Declaration:

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

- Call:

```
methodName (value, value, ..., value) ;
```

Multiple params 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:

```
4444444444  
171717171717  
  
000000000
```

- Modify the `Stars` program to draw boxes with parameters.

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);  
}  
}
```

Value semantics

- **value semantics:** When primitive variables (`int`, `double`) are passed as parameters, their values are copied.
 - Modifying the parameter will not affect the variable passed in.

```
public static void strange(int x) {  
    x = x + 1;  
    System.out.println("1. x = " + x);  
}
```

```
public static void main(String[] args) {  
    int x = 23;  
    strange(x);  
    System.out.println("2. x = " + x);  
    ...  
}
```

Output:

```
1. x = 24  
2. x = 23
```

"Parameter Mystery" problem

```
public class ParameterMystery {  
    public static void main(String[] args) {  
        int x = 9;  
        int y = 2;  
        int z = 5;  
  
        mystery(z, y, x);  
  
        mystery(y, x, z);  
    }  
}
```

```
public static void mystery(int x, int z, int y) {  
    System.out.println(z + " and " + (y - x));  
}  
}
```

Strings

- **string**: A sequence of text characters.

```
String name = "text";
```

```
String name = expression;
```

- Examples:

```
String name = "Marla Singer";
```

```
int x = 3;
```

```
int y = 5;
```

```
String point = "(" + x + ", " + y + ")";
```

Strings as parameters

```
public class StringParameters {  
    public static void main(String[] args) {  
        sayHello("Marty") ;  
  
        String teacher = "Bictolia";  
        sayHello(teacher) ;  
    }  
  
    public static void sayHello(String name) {  
        System.out.println("Welcome, " + name);  
    }  
}
```

Output:

```
Welcome, Marty  
Welcome, Bictolia
```

- Modify the `Stars` program to use string parameters. Use a method named `repeat` that prints a string many times.

Stars solution

```
// Prints several lines and boxes made of stars.  
// Fourth version with String parameters.
```

```
public class Stars4 {  
    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) {  
        repeat("*", count);  
        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("*");  
        repeat(" ", width - 2);  
        System.out.println("*");  
    }  
    line(width);  
}
```

// Prints the given String the given number of times.

```
public static void repeat(String s, int times) {  
    for (int i = 1; i <= times; i++) {  
        System.out.print(s);  
    }  
}  
}
```


Return values

Java's Math class

Method name	Description
<code>Math.abs (value)</code>	absolute value
<code>Math.ceil (value)</code>	rounds up
<code>Math.floor (value)</code>	rounds down
<code>Math.log10 (value)</code>	logarithm, base 10
<code>Math.max (value1, value2)</code>	larger of two values
<code>Math.min (value1, value2)</code>	smaller of two values
<code>Math.pow (base, exp)</code>	base to the exp power
<code>Math.random ()</code>	random double between 0 and 1
<code>Math.round (value)</code>	nearest whole number
<code>Math.sqrt (value)</code>	square root
<code>Math.sin (value)</code> <code>Math.cos (value)</code> <code>Math.tan (value)</code>	sine/cosine/tangent of an angle in radians
<code>Math.toDegrees (value)</code> <code>Math.toRadians (value)</code>	convert degrees to radians and back

Constant	Description
<code>Math.E</code>	2.7182818...
<code>Math.PI</code>	3.1415926...

Calling Math methods

`Math.methodName (parameters)`

- Examples:

```
double squareRoot = Math.sqrt(121.0);  
System.out.println(squareRoot);           // 11.0
```

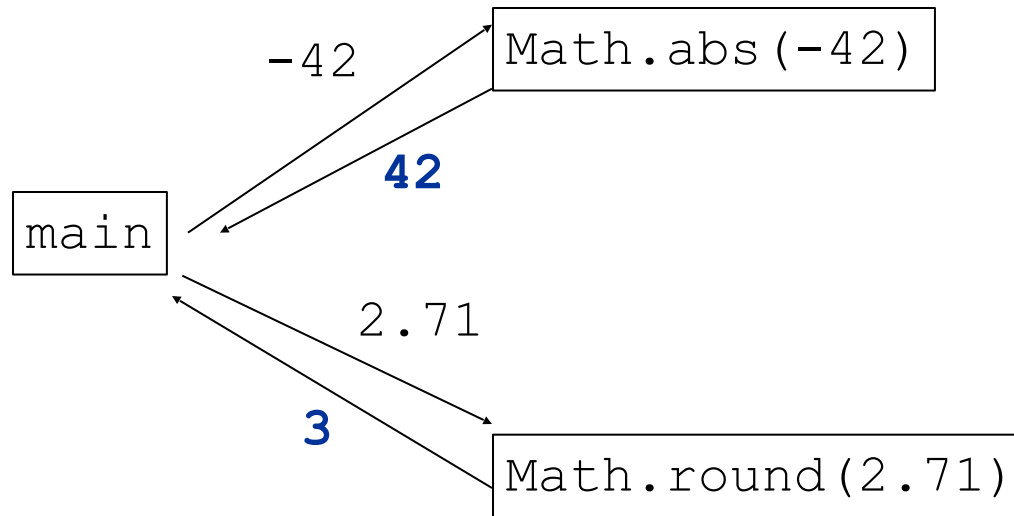
```
int absoluteValue = Math.abs(-50);  
System.out.println(absoluteValue);        // 50
```

```
System.out.println(Math.min(3, 7) + 2);    // 5
```

- The `Math` methods do not print to the console.
 - Each method produces ("returns") a numeric result.
 - The results are used as expressions (printed, stored, etc.).

Return

- **return:** To send out a value as the result of a method.
 - The opposite of a parameter:
 - Parameters send information in from the caller to the method.
 - Return values send information out from a method to its caller.
 - A call to the method can be used as part of an expression.



Math questions

- Evaluate the following expressions:
 - `Math.abs(-1.23)`
 - `Math.pow(3, 2)`
 - `Math.pow(10, -2)`
 - `Math.sqrt(121.0) - Math.sqrt(256.0)`
 - `Math.round(Math.PI) + Math.round(Math.E)`
 - `Math.ceil(6.022) + Math.floor(15.9994)`
 - `Math.abs(Math.min(-3, -5))`
- `Math.max` and `Math.min` can be used to bound numbers.
Consider an `int` variable named `age`.
 - What statement would replace negative ages with 0?
 - What statement would cap the maximum age to 40?

Quirks of real numbers

- Some Math methods return double or other non-int types.

```
int x = Math.pow(10, 3);    // ERROR: incompat. types
```

- Some double values print poorly (too many digits).

```
double result = 1.0 / 3.0;  
System.out.println(result);    // 0.3333333333333333
```

- The computer represents doubles in an imprecise way.

```
System.out.println(0.1 + 0.2);
```

– Instead of 0.3, the output is 0.30000000000000004

Type casting

- **type cast:** A conversion from one type to another.
 - To promote an `int` into a `double` to get exact division from `/`
 - To truncate a `double` from a real number to an integer

- Syntax:

(type) expression

Examples:

```
double result = (double) 19 / 5;           // 3.8
int result2 = (int) result;                 // 3
int x = (int) Math.pow(10, 3);              // 1000
```

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.

```
- double x = (double) 1 + 1 / 2;           // 1
- double y = 1 + (double) 1 / 2;           // 1.5
```

- You can use parentheses to force evaluation order.
 - double average = (double) (a + b + c) / 3;
- A conversion to double can be achieved in other ways.
 - double average = 1.0 * (a + b + c) / 3;

Returning a value

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

- Example:

```
// Returns the slope of the line between the given points.  
public static double slope(int x1, int y1, int x2, int y2) {  
    double dy = y2 - y1;  
    double dx = x2 - x1;  
    return dy / dx;  
}
```

- slope(1, 3, 5, 11) returns 2.0

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, 0, 6, 3);  
    System.out.println("The slope is " + result); // ERROR:  
                                                // result not defined  
}  
  
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

- Instead, returning sends the variable's value back.
 - The returned value must be stored into a variable or used in an expression to be useful to the caller.

```
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;  
}
```

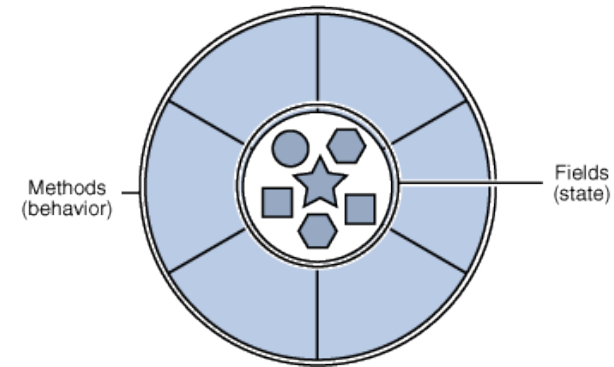
Objects and Classes; Strings

Classes and objects

- **class:** A program entity that represents either:
 1. A program / module, or
 2. A type of objects.
 - A class is a blueprint or template for constructing objects.
 - Example: The `DrawingPanel` class (type) is a template for creating many `DrawingPanel` objects (windows).
 - Java has 1000s of classes. Later (Ch.8) we will write our own.
- **object:** An entity that combines data and behavior.
 - **object-oriented programming (OOP):** Programs that perform their behavior as interactions between objects.

Objects

- **object:** An entity that contains data and behavior.
 - data: variables inside the object
 - behavior: methods inside the object
 - You interact with the methods; the data is hidden in the object.



- Constructing (creating) an object:
Type **objectName** = new **Type** (**parameters**) ;
- Calling an object's method:
objectName . **methodName** (**parameters**) ;

Blueprint analogy

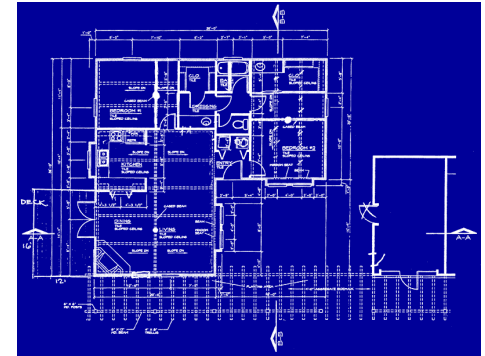
iPod blueprint/factory

state:

current song
volume
battery life

behavior:

power on/off
change station/song
change volume
choose random song



creates

iPod #1

state:

song = "1,000,000 Miles"
volume = 17
battery life = 2.5 hrs

behavior:

power on/off
change station/song
change volume
choose random song



iPod #2

state:

song = "Letting You"
volume = 9
battery life = 3.41 hrs

behavior:

power on/off
change station/song
change volume
choose random song



iPod #3

state:

song = "Discipline"
volume = 24
battery life = 1.8 hrs

behavior:

power on/off
change station/song
change volume
choose random song



Strings

- **string**: An object storing a sequence of text characters.
 - Unlike most other objects, a `String` is not created with `new`.

```
String name = "text";  
String name = expression;
```

- Examples:

```
String name = "Marla Singer";
```

```
int x = 3;  
int y = 5;  
String point = "(" + x + ", " + y + " )";
```

Indexes

- Characters of a string are numbered with 0-based indexes:

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

- First character's index : 0
- Last character's index : 1 less than the string's length
- The individual characters are values of type `char` (seen later)

String methods

Method name	Description
<code>indexOf(str)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>length()</code>	number of characters in this string
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	the characters in this string from index1 (inclusive) to index2 (<u>exclusive</u>); if index2 is omitted, grabs till end of string
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";  
System.out.println(gangsta.length());    // 7
```

String method examples

```
// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());           // 12
System.out.println(s1.indexOf("e"));        // 8
System.out.println(s1.substring(7, 10));    // "Reg"
String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());      // "arty s"
```

- Given the following string:

```
// index      0123456789012345678901
String book = "Building Java Programs";
```

- How would you extract the word "Java" ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "lil bow wow";  
s.toUpperCase();  
System.out.println(s);    // lil bow wow
```

- To modify a variable's value, you must reassign it:

```
String s = "lil bow wow";  
s = s.toUpperCase();  
System.out.println(s);    // LIL BOW WOW
```