

Lecture 1: Basic Java Syntax

Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp

Copyright (c) Pearson 2013.
All rights reserved.

Java Terminology

- **class:**
 - (a) A module or program that can contain executable code.
 - (b) A description of a type of objects. (Animal class, Human class, Employee class, Car class)
- **statement:** An executable piece of code that represents a complete command to the computer.
 - every basic Java statement ends with a semicolon ;
- **method:** A named sequence of statements that can be executed together to perform a particular action or computation.

A Java program

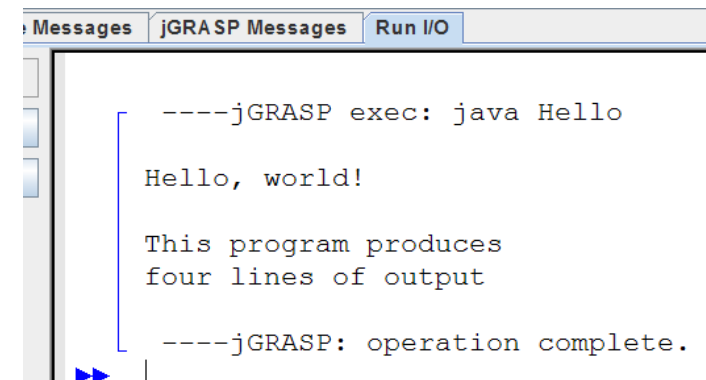
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
        System.out.println();  
        System.out.println("This program produces");  
        System.out.println("four lines of output");  
    }  
}
```

- **Its output:**

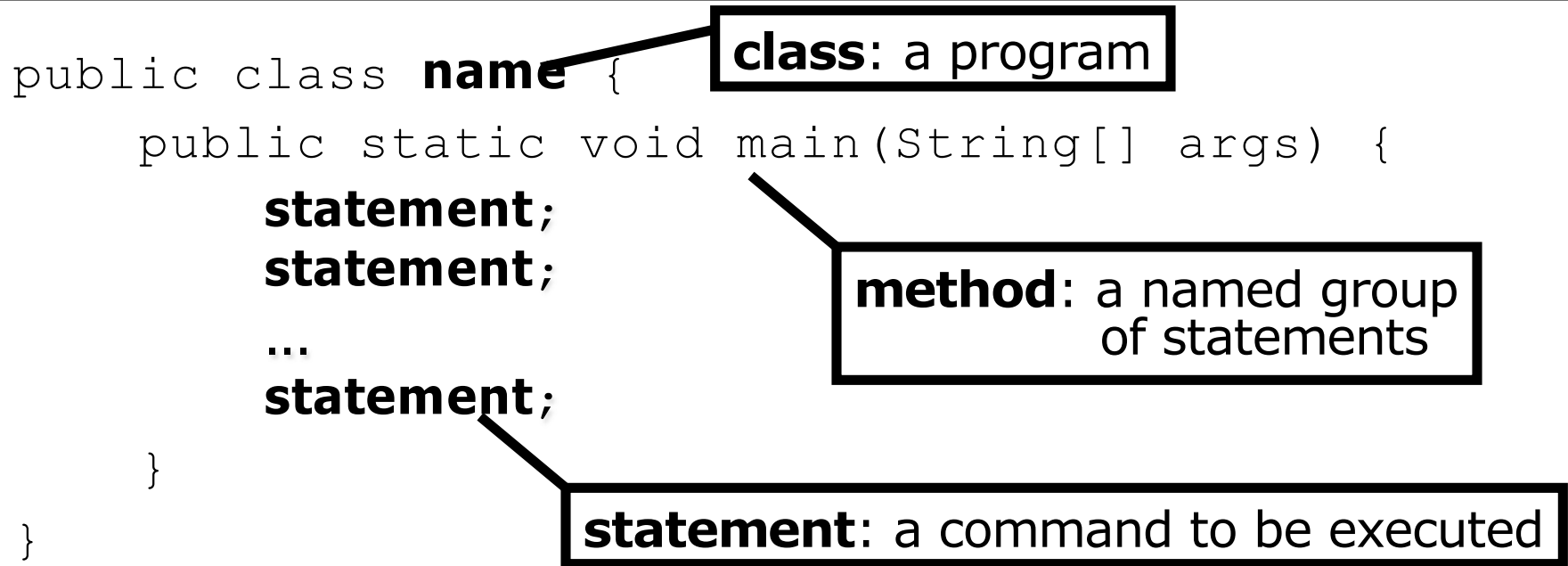
Hello, world!

This program produces
four lines of output

- **console:** Text box into which the program's output is printed.



Structure of a Java program



- Every executable Java program consists of a **class**,
 - that contains a **method** named `main`,
 - that contains the **statements** (commands) to be executed.

System.out.println

- A statement that prints a line of output on the console.
 - pronounced "print-linn"
 - sometimes called a "println statement" for short
- Two ways to use `System.out.println` :
 - `System.out.println("text");`
Prints the given message as output.
 - `System.out.println();`
Prints a blank line of output.

Names and identifiers

- You must give your program a name.

```
public class GangstaRap {
```

- Naming convention: capitalize each word (e.g. `MyClassName`)
- Your program's file must match exactly (`GangstaRap.java`)
 - includes capitalization (Java is "case-sensitive")
- **identifier**: A name given to an item or variable in your program.
 - must start with a letter or `_` or `$`
 - subsequent characters can be any of those or a number
 - **legal**: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
 - **illegal**: `me+u` `49ers` `side-swipe` `Ph.D's`

Keywords

- **keyword:** An identifier that you cannot use because it already has a reserved meaning in Java.

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Rules for naming identifiers

- Identifiers must start with a letter and can then be followed by any number of letters and digits. The following are legal identifiers.

first hiThere numStudents Twoby4
first23Name87

- Java does allow the set of letters to include the dollar sign and the underscore symbol. Thus, the following are also legal.

two_plus_two _count \$2donuts MAX_COUNT

Rules for naming identifiers

- Java reserved words cannot be used as identifier. For example, it is illegal in Java to do:

int **class**=7; OR double **public**=1;

- The following are ILLEGAL identifiers.

two+two hi there hi-There 2by4

Other Conventions

- All class names should begin with a capitalized letter.
- The names of methods should begin with lowercase letters, as in method main. (e.g. public static void print())
- When putting several words together, we capitalize the first letter of each word.

```
int numberOfStudents      public static void printClassList()
```

- Constants should have all letters in uppercase and words separated by underscores.

```
int DAYS_IN_WEEK=7;
```

Rules for naming identifiers

Suppose that you were going to put together the words "all my children" into an identifier. Depending upon what the identifier is used for, you'd turn this into:

- AllMyChildren for a class name (starts with a capital, capitalizes remaining words)
- allMyChildren for a method name (starts with a lowercase letter, capitalizes remaining words)
- ALL_MY_CHILDREN for a constant name (all uppercase separated by underscores; described in Chapter 2)

Syntax

- **syntax:** The set of legal structures and commands that can be used in a particular language.
 - Every basic Java statement ends with a semicolon ;
 - The contents of a class or method occur between { and }
- **syntax error (compiler error):** A problem in the structure of a program that causes the compiler to fail.
 - Missing semicolon
 - Too many or too few { } braces
 - Illegal identifier for class name
 - Class and file names do not match
 - ...

Syntax error example

```
1 public class Hello {  
2     pooblic static void main(String[] args) {  
3         System.owt.println("Hello, world!")_  
4     }  
5 }
```

- Compiler output:

```
Hello.java:2: <identifier> expected  
    pooblic static void main(String[] args) {  
        ^
```

```
Hello.java:3: ';' expected  
    }  
    ^
```

```
2 errors
```

- The compiler shows the line number where it found the error.
- The error messages can be tough to understand!

Strings

- **string**: A sequence of characters to be printed.
 - Starts and ends with a " quote " character.
 - The quotes do not appear in the output.

- Examples:

```
"hello"
```

```
"This is a string.  It's very long!"
```

- Restrictions:

- May not span multiple lines.

```
"This is not  
a legal String."
```

- May not contain a " character.

```
"This is not a "legal" String either."
```

Comments

- **comment:** A note written in source code by the programmer to describe or clarify the code.
 - Comments are not executed when your program runs.
- Syntax:
 - `// comment text, on one line`
 - or,
 - `/* comment text; may span multiple lines */`
- Examples:
 - `// This is a one-line comment.`
 - `/* This is a very long
multi-line
comment. */`

Using comments

- Where to place comments:
 - at the top of each file (a "comment header")
 - at the start of every method (seen later)
 - to explain complex pieces of code
- Comments are useful for:
 - Understanding larger, more complex programs.
 - Multiple programmers working together, who must understand each other's code.

Comments example

```
/* Suzy Student, CS 101, Fall 2019
   This program prints lyrics about ... something. */

public class BaWitDaBa {
    public static void main(String[] args) {
        // first verse
        System.out.println("Bawitdaba");
        System.out.println("da bang a dang diggy diggy");
        System.out.println();

        // second verse
        System.out.println("diggy said the boogy");
        System.out.println("said up jump the boogy");
    }
}
```

Lab 1

Write your first program. Print out some messages on the console from the main method. Remember to follow the naming conventions as well as proper indentations.



Static methods

Algorithms

- **algorithm:** A list of steps for solving a problem.
- Example algorithm: "Bake sugar cookies"
 - Mix the dry ingredients.
 - Cream the butter and sugar.
 - Beat in the eggs.
 - Stir in the dry ingredients.
 - Set the oven temperature.
 - Set the timer.
 - Place the cookies into the oven.
 - Allow the cookies to bake.
 - Spread frosting and sprinkles onto the cookies.
 - ...



Problems with algorithms

- *lack of structure*: Many tiny steps; tough to remember.
- *redundancy*: Consider making a double batch...
 - Mix the dry ingredients.
 - Cream the butter and sugar.
 - Beat in the eggs.
 - Stir in the dry ingredients.
 - Set the oven temperature.
 - Set the timer.
 - Place the first batch of cookies into the oven.
 - Allow the cookies to bake.
 - Set the timer.
 - Place the second batch of cookies into the oven.
 - Allow the cookies to bake.
 - Mix ingredients for frosting.
 - ...

Structured algorithms

- **structured algorithm:** Split into coherent tasks.

1 Make the cookie batter.

- Mix the dry ingredients.
- Cream the butter and sugar.
- Beat in the eggs.
- Stir in the dry ingredients.

2 Bake the cookies.

- Set the oven temperature.
- Set the timer.
- Place the cookies into the oven.
- Allow the cookies to bake.

3 Add frosting and sprinkles.

- Mix the ingredients for the frosting.
- Spread frosting and sprinkles onto the cookies.

...

Removing redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy.

1 Make the cookie batter.

- Mix the dry ingredients.
- ...

2a Bake the cookies (first batch).

- Set the oven temperature.
- Set the timer.
- ...

2b Bake the cookies (second batch).

3 Decorate the cookies.

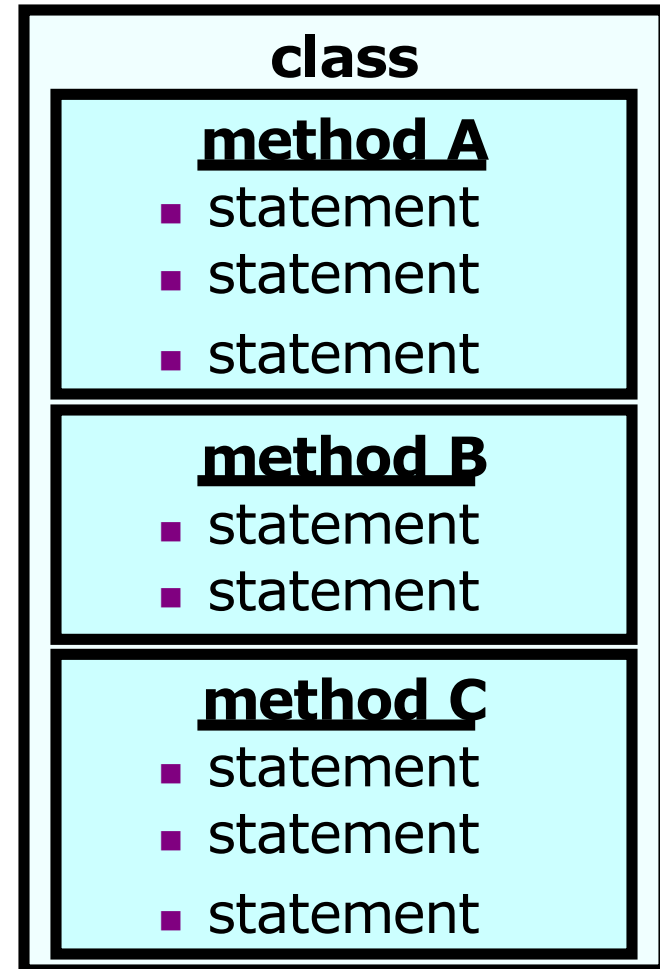
- ...

A program with redundancy

```
public class BakeCookies {  
    public static void main(String[] args) {  
        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.");  
        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.");  
        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.");  
        System.out.println("Mix ingredients for frosting.");  
        System.out.println("Spread frosting and sprinkles.");  
    }  
}
```


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.

Design of an algorithm

// This program displays a delicious recipe for baking cookies.

```
public class BakeCookies2 {  
    public static void main(String[] args) {  
        // Step 1: Make the cake batter.  
        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 2a: Bake cookies (first batch).  
        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 2b: Bake cookies (second batch).  
        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.  
        System.out.println("Mix ingredients for frosting.");  
        System.out.println("Spread frosting and sprinkles.");  
    }  
}
```

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
```

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.");  
    }  
}
```

Methods calling methods

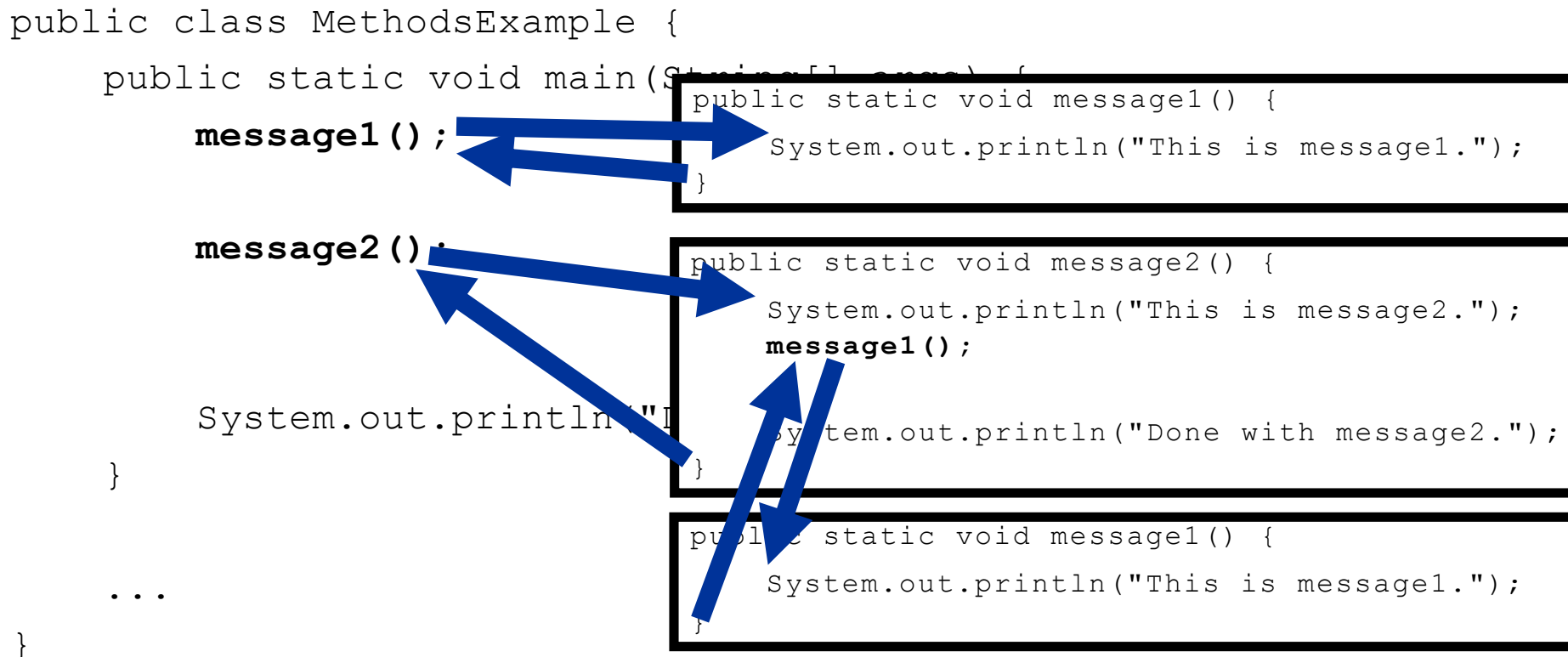
```
public class MethodsExample {  
    public static void main(String[] args) {  
        message1();  
        message2();  
        System.out.println("Done with main.");  
    }  
    public static void message1() {  
        System.out.println("This is message1.");  
    }  
    public static void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



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.)

Lab 2

Modify your first program to include a static method that prints out some message. Call the method from the main method.