02 – Java Fundamentals

ICSI 201
Introduction to Computer Science

Qi Wang
Department of Computer Science
University at Albany
State University of New York

Course Materials

- All course materials are **copyrighted** and can be used by the students who are enrolled in the class only.
- Posting/sharing the course materials for other uses without permission is prohibited.

Outline

- Character Strings
- Expressions
- Primitive Data Types
- Variables and Assignment
- Interactive Programs
- Data Conversion
- Packages
- Creating Objects
- The String Class, the Random class
- Format output

Character Strings

Character Strings

- A string of characters can be represented as a *string literal* by putting double quotes around the text:

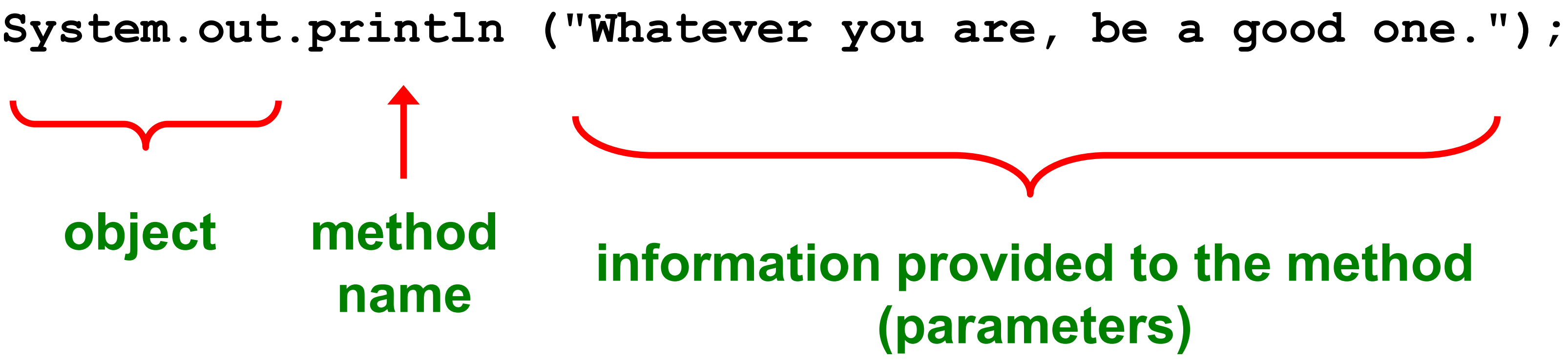
  Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

- Every character string is an object in Java, defined by the `String` class.
- Every string literal represents a `String` object.

The println Method

- The `System.out` object represents a destination (the monitor screen) to which we can send output.



```
System.out.println ("Whatever you are, be a good one.");
```

object

method name

information provided to the method (parameters)

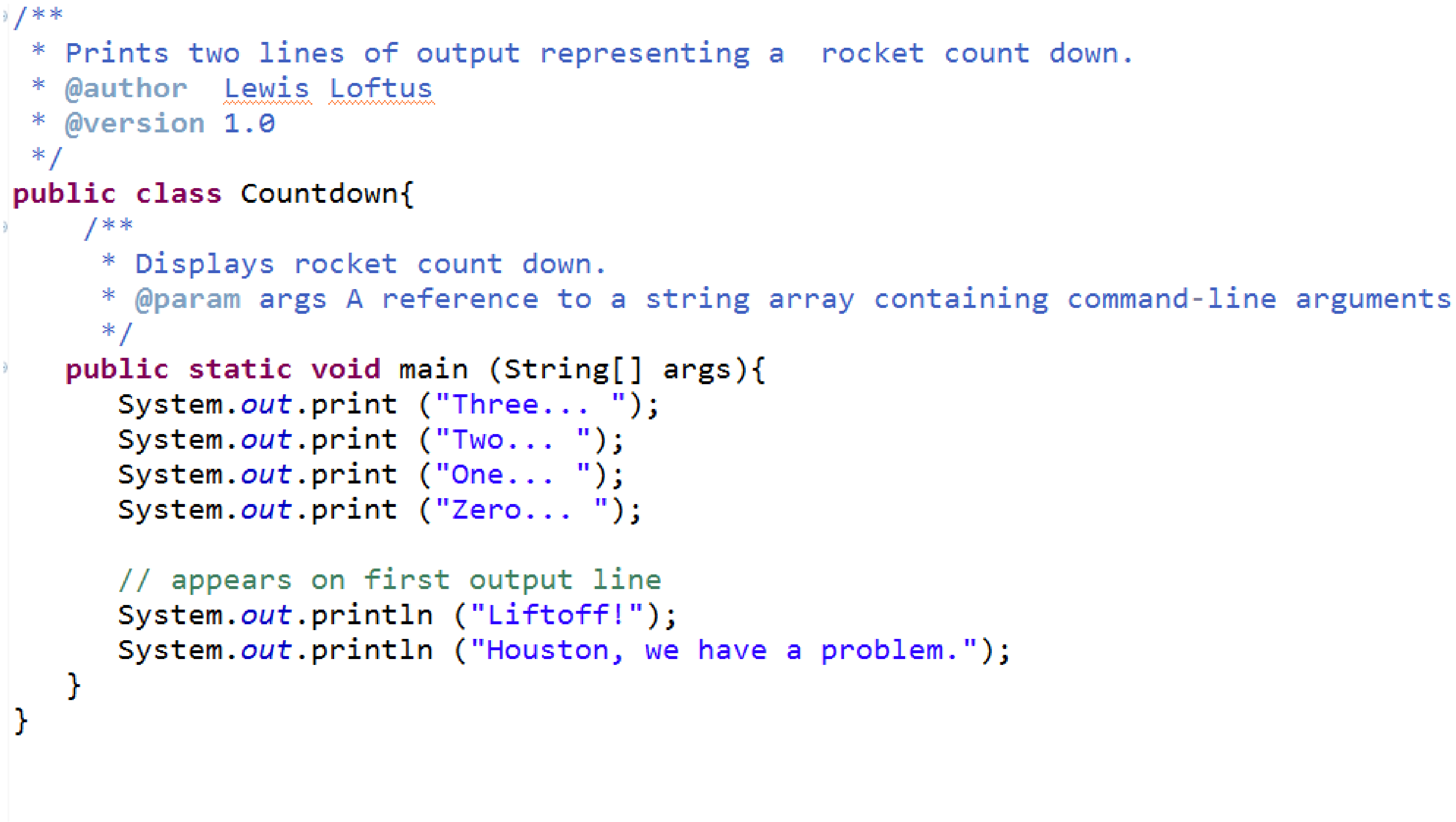
The print Method

- The `System.out` object provides another service as well - `print`.

- The `print` method is similar to the `println` method, except that it does not advance to the next line.

- Therefore, anything printed after a `print` statement will appear on the same line.

The print Method



```
/**
 * Prints two lines of output representing a  rocket count down.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class Countdown{
    /**
     * Displays rocket count down.
     * @param args A reference to a string array containing command-line arguments
     */
   public static void main (String[] args){
      System.out.print ("Three... ");
      System.out.print ("Two... ");
      System.out.print ("One... ");
      System.out.print ("Zero... ");

      // appears on first output line
      System.out.println ("Liftoff!");
      System.out.println ("Houston, we have a problem.");
   }
}
```

String Concatenation

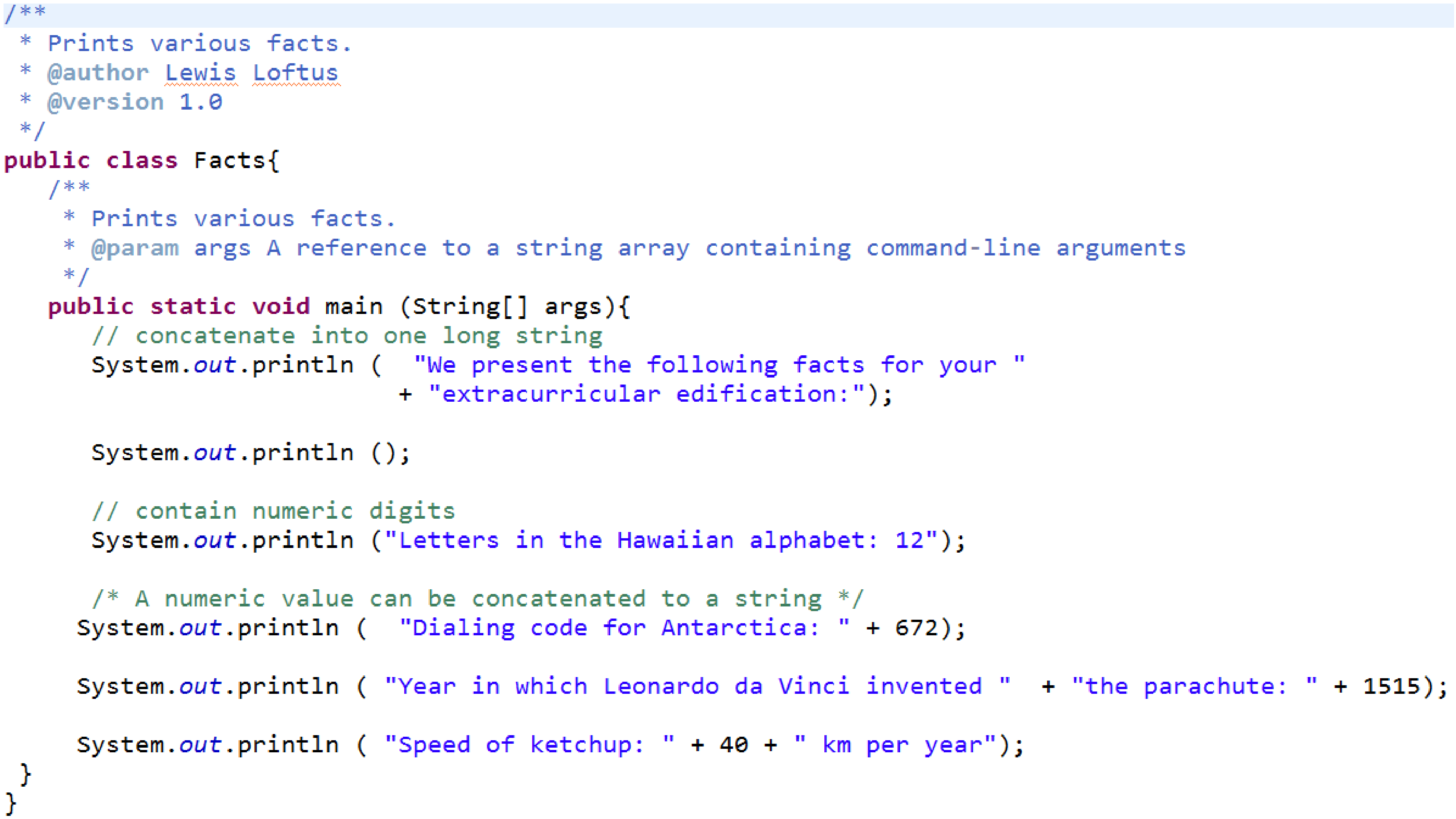
- The *string concatenation operator* (+) is used to append one string to the end of another.

```
    "Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string.

```
    “COSC " + 1315
```

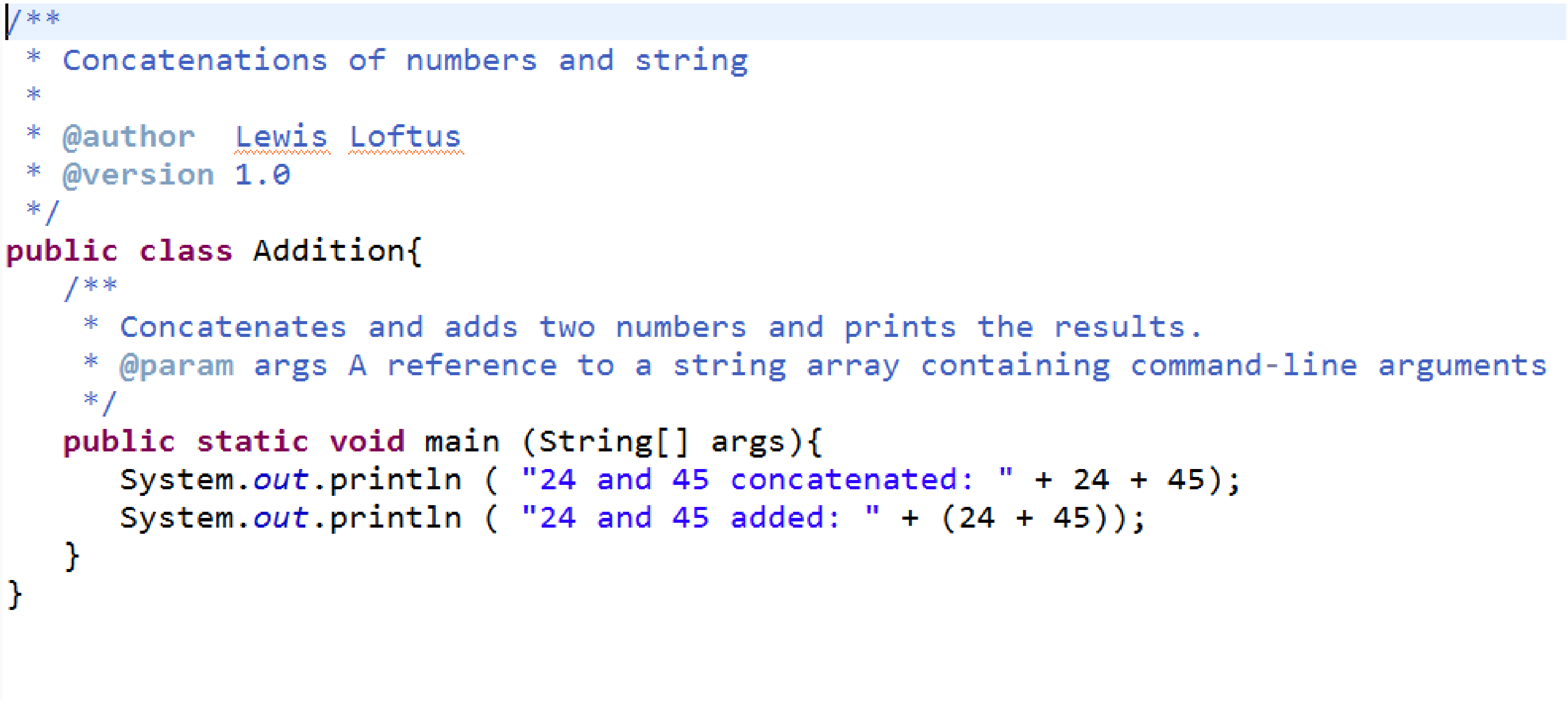
- A string literal cannot be broken across two lines in a program.



```java
/**
 * Prints various facts.
 * @author Lewis Loftus
 * @version 1.0
 */
public class Facts{
   /**
    * Prints various facts.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      // concatenate into one long string
      System.out.println (  "We present the following facts for your "
                           + "extracurricular edification:");

      System.out.println ();

      // contain numeric digits
      System.out.println ("Letters in the Hawaiian alphabet: 12");

      /* A numeric value can be concatenated to a string */
     System.out.println (  "Dialing code for Antarctica: " + 672);

     System.out.println ( "Year in which Leonardo da Vinci invented "  + "the parachute: " + 1515);

     System.out.println ( "Speed of ketchup: " + 40 + " km per year");
 }
}
```

String Concatenation

- The + operator is also used for arithmetic addition.
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation.
- If both operands are numeric, it adds them.
- The + operator is evaluated left to right, but parentheses can be used to force the order.



```
/**
 * Concatenations of numbers and string
 *
 * @author  Lewis Loftus
 * @version 1.0
 */
public class Addition{
   /**
    * Concatenates and adds two numbers and prints the results.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      System.out.println ( "24 and 45 concatenated: " + 24 + 45);
      System.out.println ( "24 and 45 added: " + (24 + 45));
   }
}
```

Escape Sequences

- What if we wanted to print a quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string.

```
System.out.println ("I said "Hello" to
                      you.");
```

Escape Sequences

- An *escape sequence* is a series of characters that represents a special character.
- An escape sequence begins with a backslash character (\\).
- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Escape Sequences

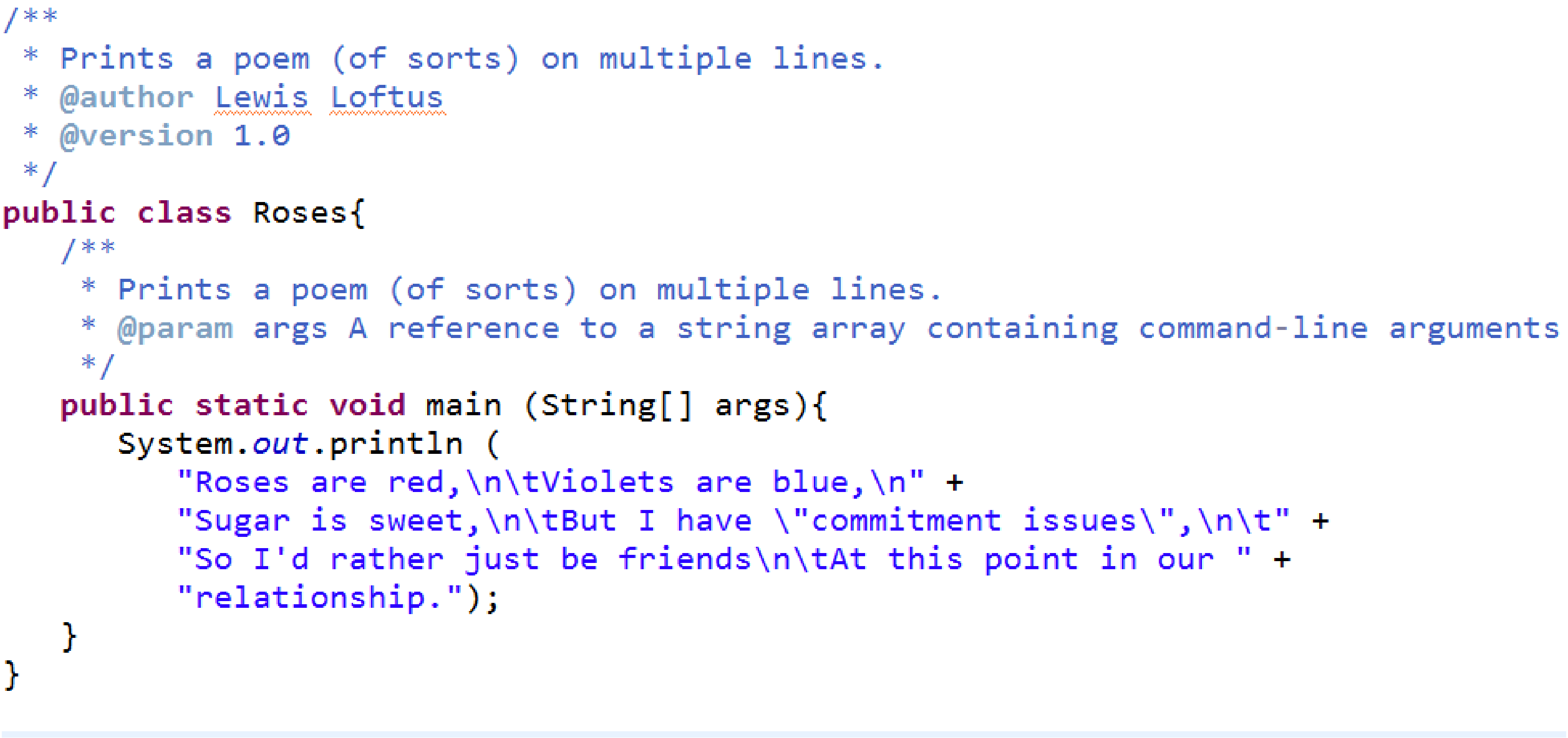
- Now we can print a quote character using its escape sequence.

```
System.out.println(
          "I said \"Hello\" to you.");
```

Print a double quote

Print a double quote

```
I said "Hello" to you.
```



```java
/**
 * Prints a poem (of sorts) on multiple lines.
 * @author Lewis Loftus
 * @version 1.0
 */
public class Roses{
   /**
    * Prints a poem (of sorts) on multiple lines.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      System.out.println (
         "Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```

Expressions

Expressions

- An *expression* is a combination of one or more operators and operands.
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

Division and Remainder

- If both operands to the division operator (`/`) are integers, the result is an integer (the fractional part is discarded).

`14 / 3`	equals	`4`
`8 / 12`	equals	`0`

- The remainder operator returns the remainder of the left operand when divided by the right operand.

`14 % 3`	equals	`2`
`8 % 12`	equals	`8`

Arithmetic Operations

- If one operand or both operands are floating point number, the result is a floating number(the fractional part isn’t discarded).

`14 / 2.0`	**equals**	`7.0`
`8 * 0.2`	**equals**	`1.6`
`0.8 + 0.2`	**equals**	`1.0`
`0.8 - 0.5`	**equals**	`0.3`
`8 % 3.0`	**equals**	`2.0`

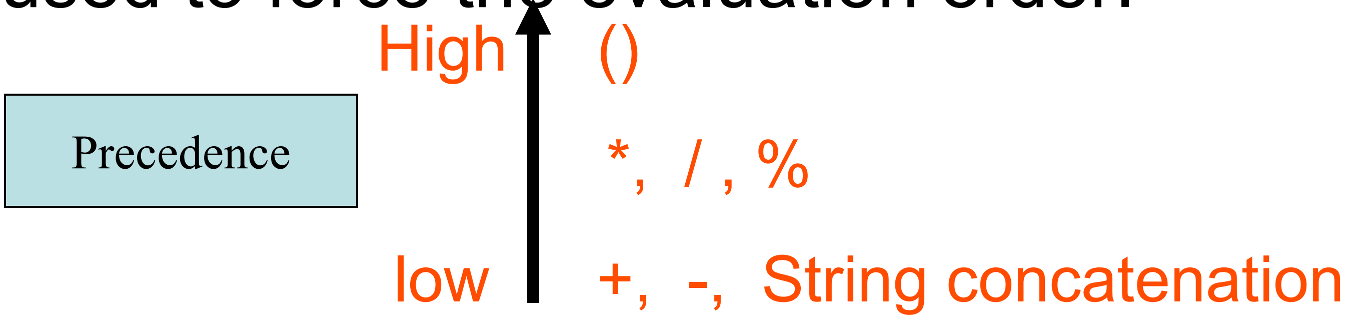
- If both operands are integers, the result is an integer.

Operator Precedence

- Operators can be combined into complex expressions.

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated.
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order.



Precedence

High

low

()

*, / , %

+, -, String concatenation

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

```
a / (b + c) - d % e
  2    1    4   3
```

```
a / (b * (c + (d - e)))
  2    3    2    1
```

Primitive Data Types

Primitive Data

- There are eight primitive data types in Java.
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`

Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store.

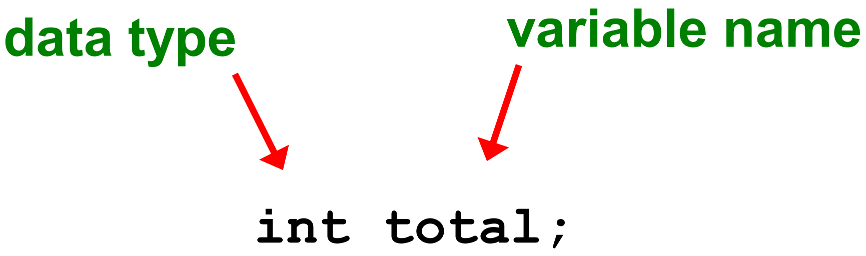
Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	< -9 x 10^{18}	> 9 x 10^{18}
`float`	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

Variables and Assignment

Variables

- A *variable* is a name for a location in memory.

- A variable must be *declared* by specifying the variable's name and the type of information that it will hold.



data type

variable name

```
int total;
```

```
int count, temp, result, totalResult;
```

Multiple variables can be created in one declaration

- By convention, a variable name should be in camel case. For example, `totalResult`.

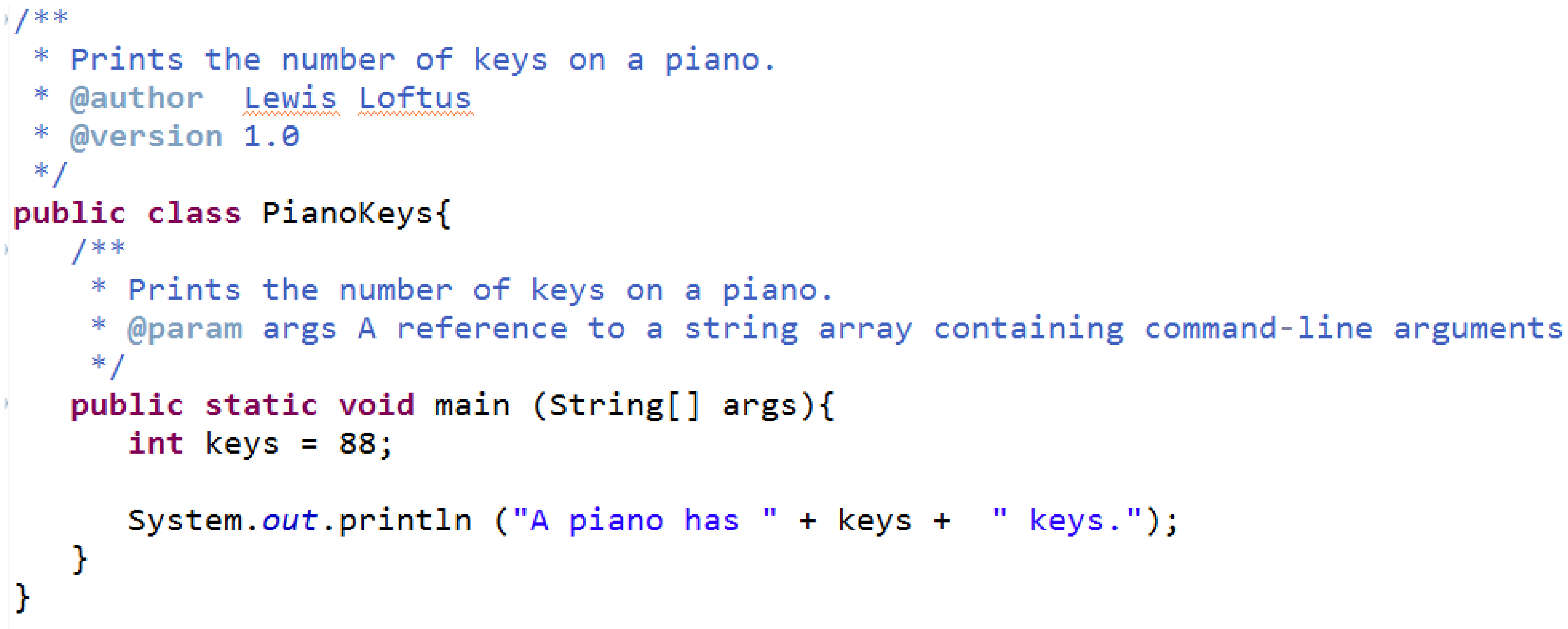
Variable Initialization

- A variable can be given an initial value in the declaration.

```
int sum = 0;
int base = 32, max = 149;
```

- = is called assignment operator.
- When a variable is referenced in a program, its current value is used.

Variable Initialization

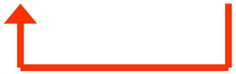


```
/**
 * Prints the number of keys on a piano.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class PianoKeys{
   /**
    * Prints the number of keys on a piano.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      int keys = 88;

      System.out.println ("A piano has " + keys +  " keys.");
   }
}
```

Assignment

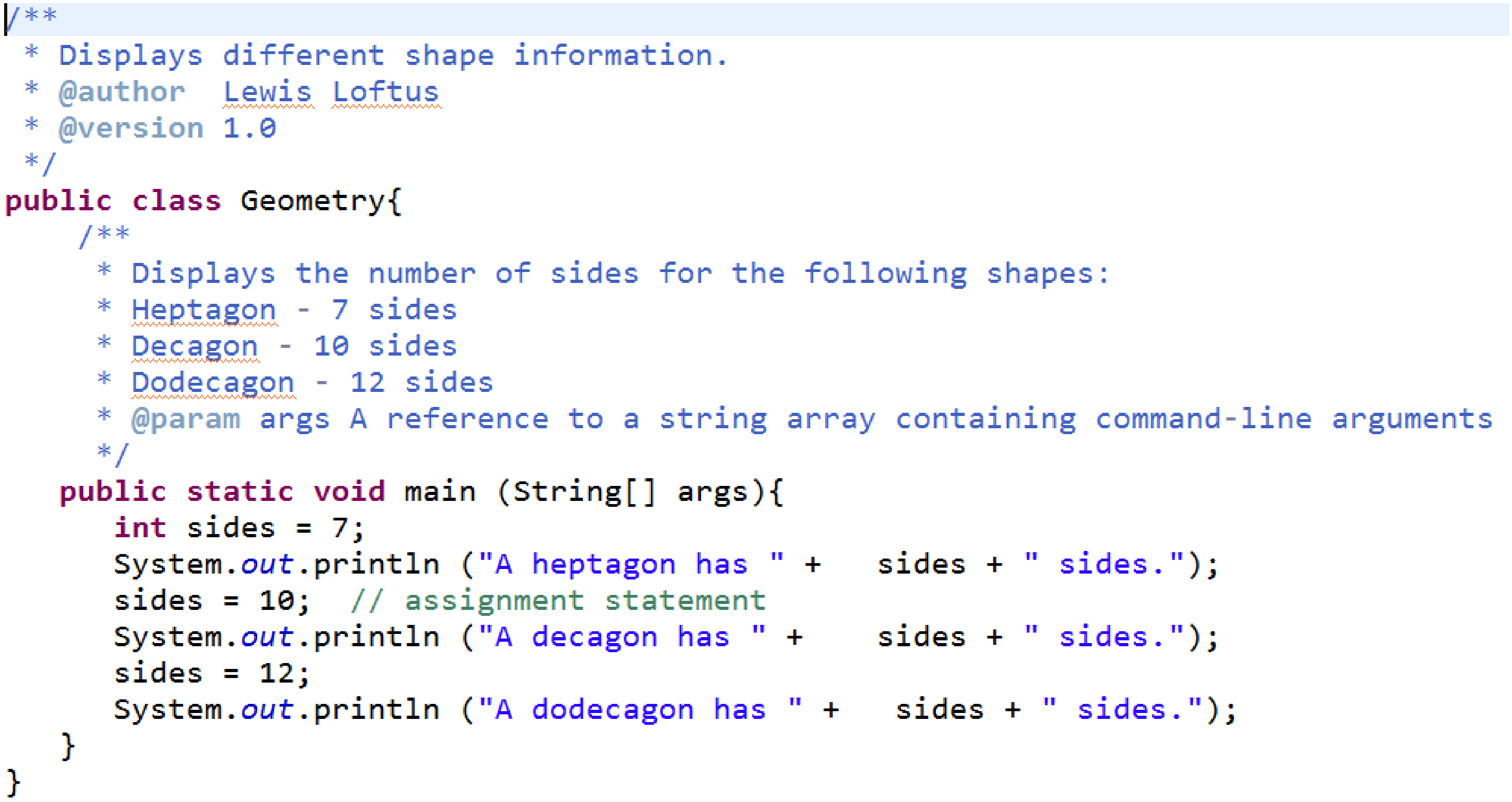
- An *assignment statement* changes the value of a variable.
- The assignment operator is the = sign.

```
total = 55;
```



- The expression on the right (RHS) is evaluated and the result is stored in the variable on the left.
- The value that was in `total` is overwritten.
- You can only assign a value to a variable that is consistent with the variable's declared type.

Assignment



```java
/**
 * Displays different shape information.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class Geometry{
    /**
     * Displays the number of sides for the following shapes:
     * Heptagon - 7 sides
     * Decagon - 10 sides
     * Dodecagon - 12 sides
     * @param args A reference to a string array containing command-line arguments
     */
   public static void main (String[] args){
      int sides = 7;
      System.out.println ("A heptagon has " +   sides + " sides.");
      sides = 10;  // assignment statement
      System.out.println ("A decagon has " +    sides + " sides.");
      sides = 12;
      System.out.println ("A dodecagon has " +   sides + " sides.");
   }
}
```

Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence.
- The compiler will issue an error if you try to change the value of a constant.
- In Java, we use the `final` modifier to declare a constant.

```
final int MIN_HEIGHT = 69;
```

Constants

- Constants are useful for three important reasons:
  - First, they give meaning to otherwise unclear literal values. For example, MAX_LOAD means more than the literal 250.
  - Second, they facilitate program maintenance.
    - If a constant is used in multiple places, its value need only be updated in one place.
  - Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers.

Characters

- A `char` variable stores a single character.
- Character literals are delimited by single quotes:

```
  'a'   'X'    '7'    '$'    ','    '\n'
```

- Example declarations:

```
char topGrade = 'A';

char terminator = ';', separator = ' ‘;
```

- Note the distinction between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters.

Character Sets

- A *character set* is an ordered list of characters, with each character corresponding to a unique number.
- A `char` variable in Java can store any character from the *Unicode character set.*
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters.
- It is an international character set, containing symbols and characters from many world languages.

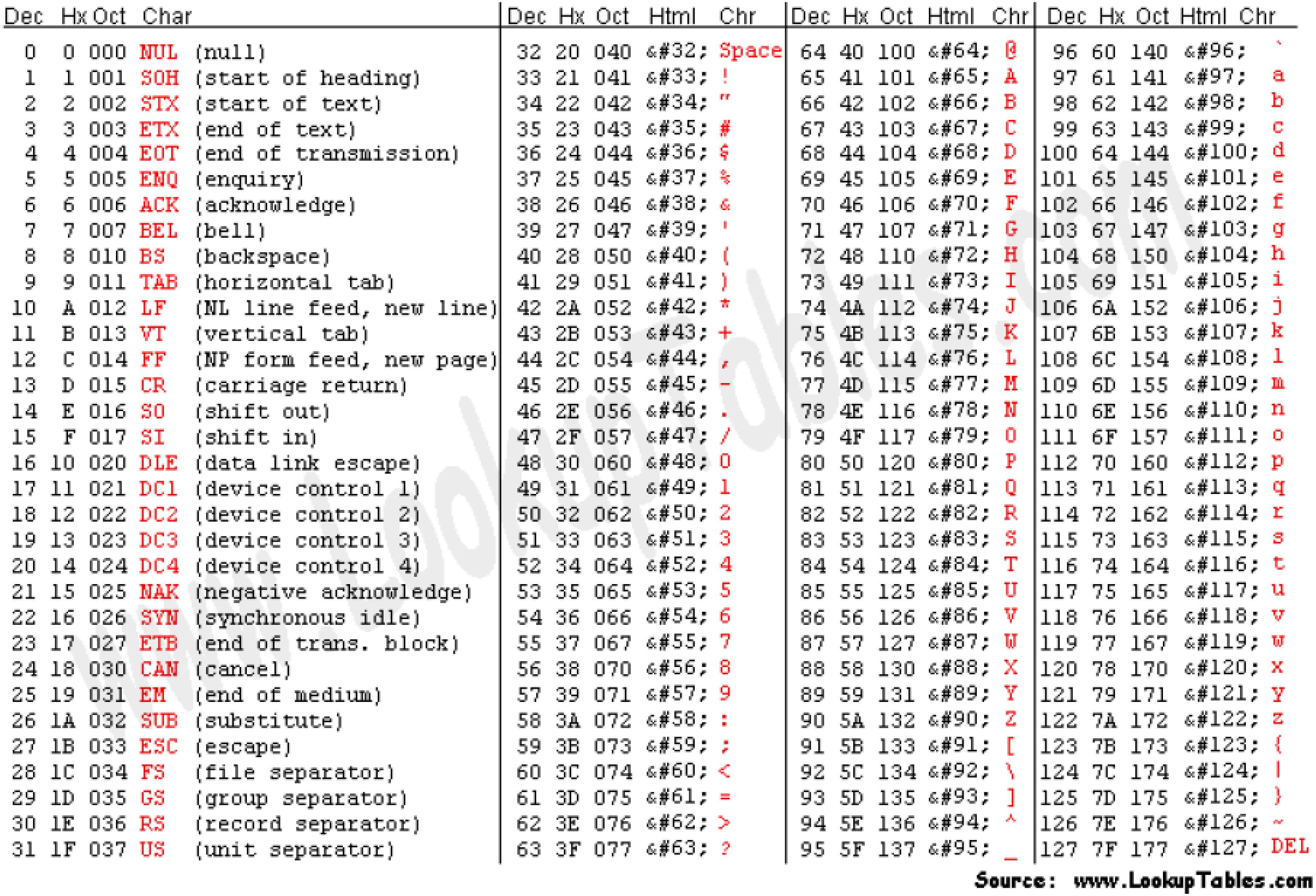
Characters

- The *ASCII character set* is older and smaller than Unicode but is still quite popular.
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	**A, B, C, …**
lowercase letters	**a, b, c, …**
punctuation	**period, semi-colon, …**
digits	**0, 1, 2, …**
special symbols	**&, |, \\, …**
control characters	**carriage return, tab, ...**

integer

character



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Boolean

- A `boolean` value represents a true or false condition.

- The reserved words `true` and `false` are the only valid values for a boolean type.

```
boolean done = false;
```

- A `boolean` variable can also be used to represent any two states, such as a light bulb being on or off.

Interactive Programs

Interactive Programs

- Programs generally need input on which to operate.
- The `Scanner` class provides convenient methods for reading input values of various types.
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard.
- Keyboard input is represented by the `System.in` object.

Reading Input

- The following line creates a Scanner object that reads from the keyboard:

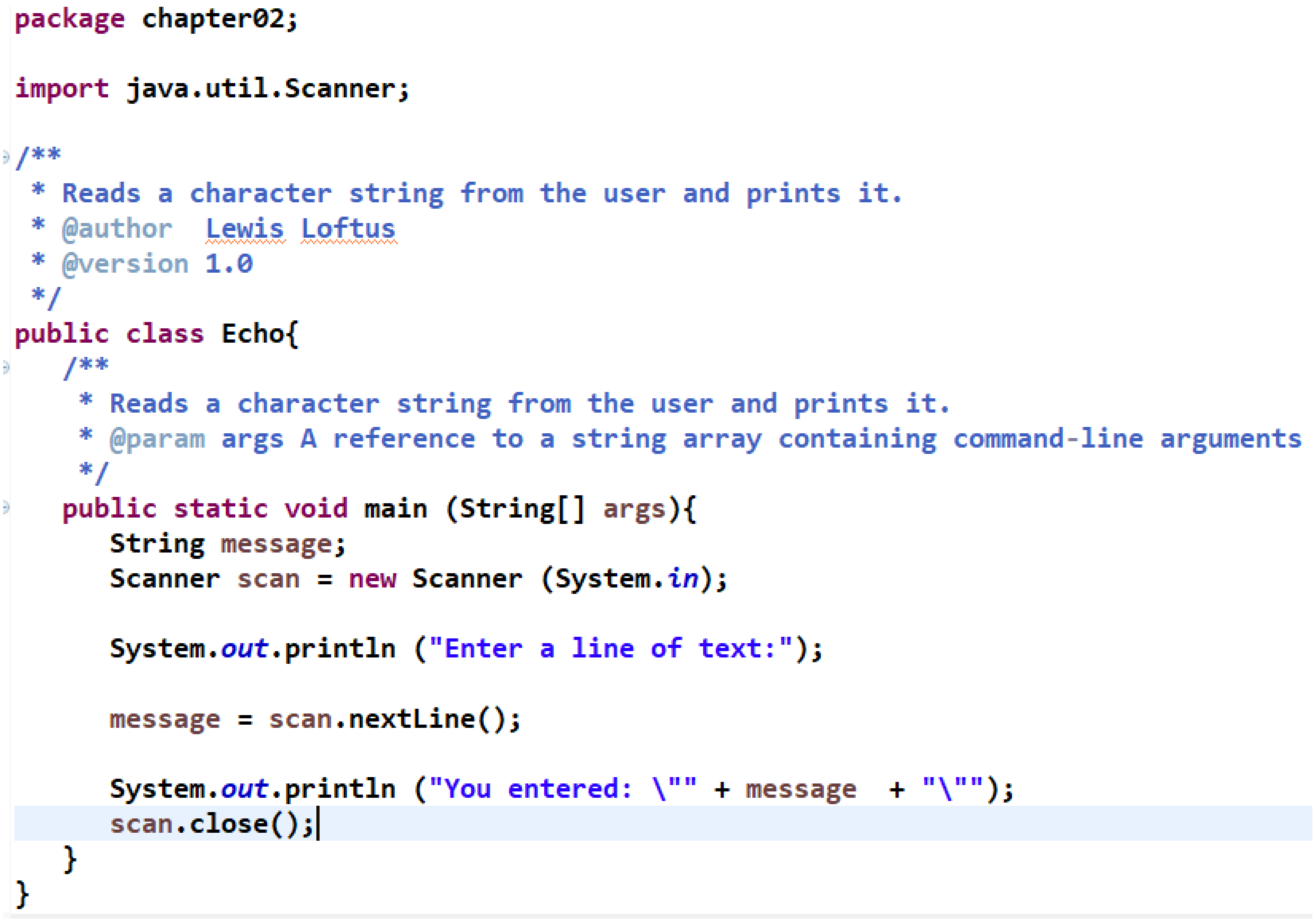
```
Scanner scan = new Scanner (System.in);
```

- The `new` operator creates the `Scanner` object.
- Once created, the `Scanner` object can be used to invoke various input methods, such as `nextLine()`.

```
String answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` class library and must be imported into a program to be used.
- The `nextLine` method reads all the input until the end of the line is found.



```java
package chapter02;

import java.util.Scanner;

/**
 * Reads a character string from the user and prints it.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class Echo{
   /**
    * Reads a character string from the user and prints it.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message  + "\"");
      scan.close();
   }
}
```

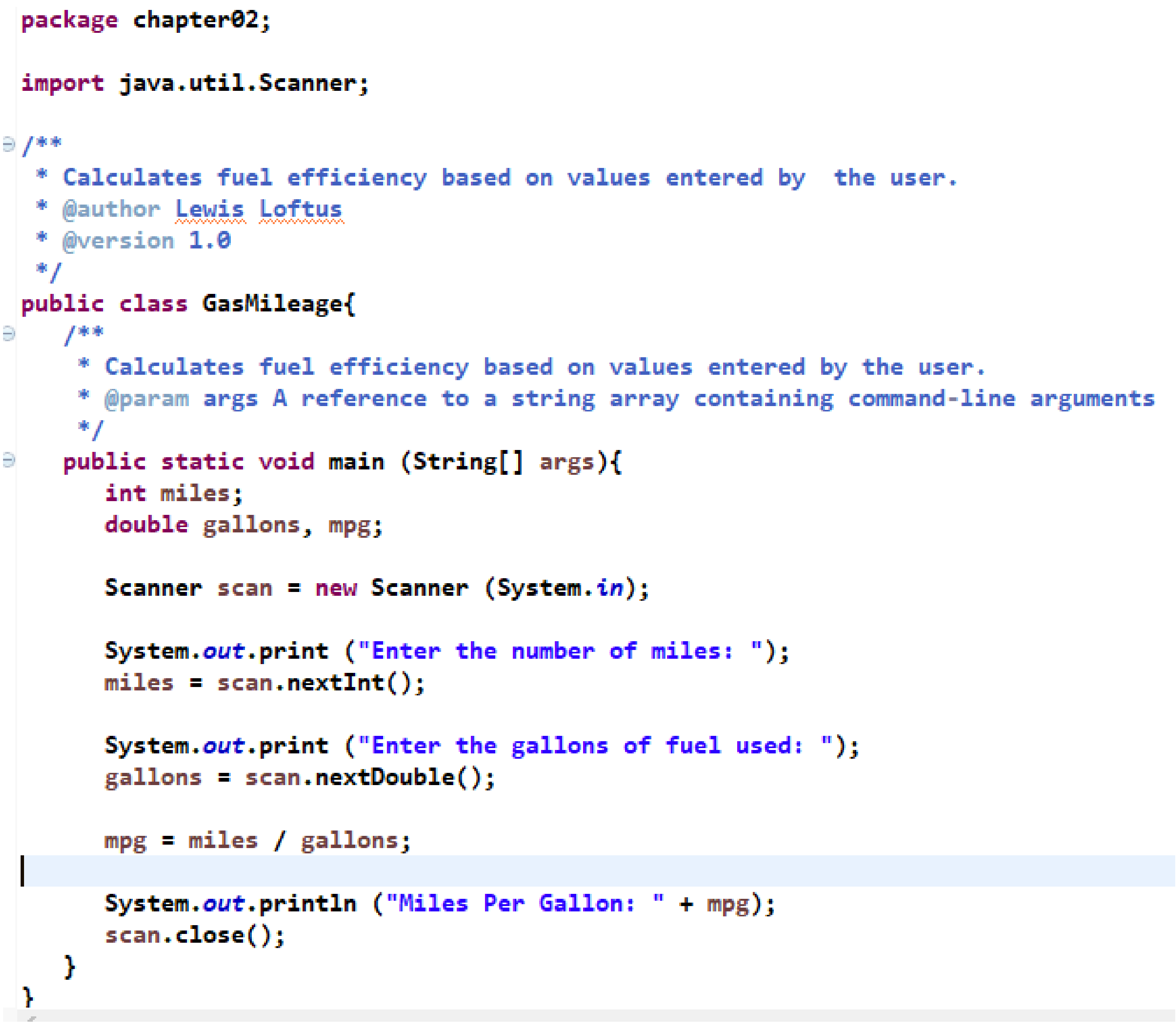
Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input.

- White space includes space characters, tabs, new line characters.

- The `next` method of the `Scanner` class reads the next input token and returns it as a string.

- Methods such as `nextInt` and `nextDouble` read data of particular types.



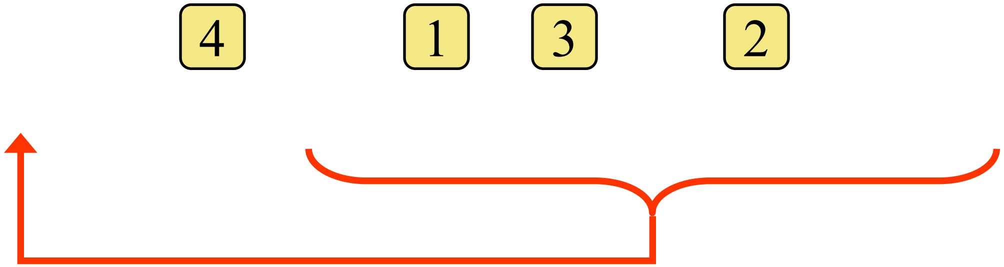
```java
package chapter02;

import java.util.Scanner;

/**
 * Calculates fuel efficiency based on values entered by  the user.
 * @author Lewis Loftus
 * @version 1.0
 */
public class GasMileage{
   /**
    * Calculates fuel efficiency based on values entered by the user.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      int miles;
      double gallons, mpg;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the number of miles: ");
      miles = scan.nextInt();

      System.out.print ("Enter the gallons of fuel used: ");
      gallons = scan.nextDouble();

      mpg = miles / gallons;

      System.out.println ("Miles Per Gallon: " + mpg);
      scan.close();
   }
}
```

Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators.

First the expression on the right hand side of the = operator is evaluated.

```
answer  =  sum / 4 + MAX * lowest;
```



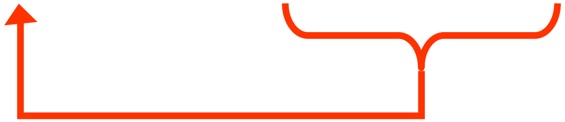
Then, the result is stored in the variable on the left hand side.

Assignment Revisited

- Both sides of an assignment statement can contain the same variable.

First, 1 is added to the original value of *count*.

```
count  =  count + 1;
```



Then, the result is stored back into *count* (overwriting the original value).

Increment and Decrement

- The increment and decrement operators use only one operand.
- The *increment operator* (++) adds one to its operand.
- The *decrement operator* (--) subtracts one from its operand.
- The statement

```
count++;
```

is functionally equivalent to

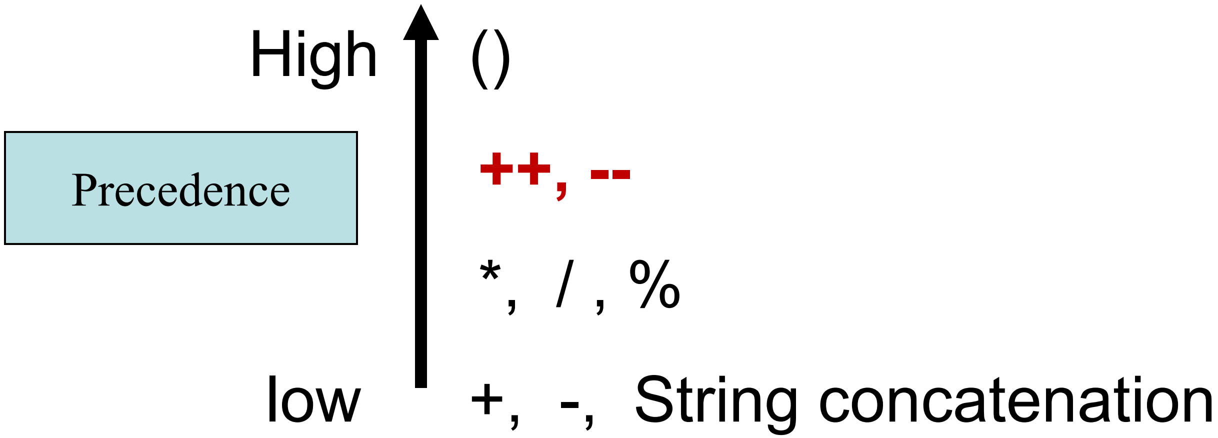
```
count = count + 1;
```

Increment and Decrement

- The increment and decrement operators can be applied in two forms:
  - *postfix* form: `count++`
  - *prefix* form: `++count`
- When used as part of a larger expression, the two forms can have different effects.
- Because of their subtleties, the increment and decrement operators should be used with care.

Increment and Decrement

- The precedence of increment and decrement operators are shown below.



Precedence

High ()

++, --

*, / , %

low +, -, String concatenation

Assignment Operators

- Often, we perform an operation on a variable, and then store the result back into that variable.
- Java provides *assignment operators* to simplify that process.
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators (compound assignment operators) in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right hand side is evaluated first, then the result is combined with the original variable.
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands.
- If the operands to the += operator are strings, the assignment operator performs string concatenation.
- The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+).

Data Conversion

Data Conversion

- Sometimes it is convenient to convert data from one type to another.
- For example, in a particular situation we may want to treat an integer as a floating point value.
- These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation.
- Conversions must be handled carefully to avoid losing information.

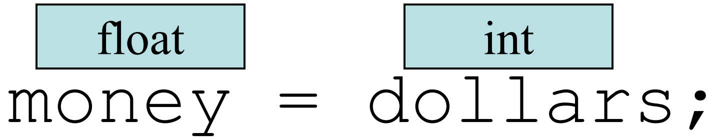
Data Conversion

- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`).
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`).
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - promotion
  - casting

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another.

- If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`.



```
money = dollars;
```

- Only widening conversions can happen via assignment.

- Note that the value or type of `dollars` did not change.

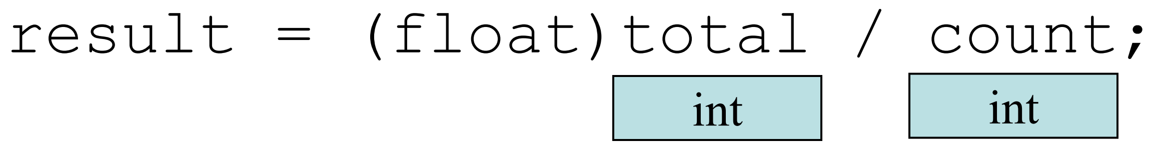
Data Conversion

- *Promotion* happens automatically when operators in expressions convert their operands.
- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a floating point value to perform the following calculation:

```
result = sum / count;
```

Casting

- *Casting* is the most powerful, and dangerous, technique for conversion.
- Both widening and narrowing conversions can be accomplished by explicitly casting a value.
- To cast, the type is put in parentheses in front of the value being converted.
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:



```
result = (float)total / count;
```

Packages

Class Libraries

- A *class library* is a collection of classes that we can use when developing programs.
- The *Java standard class library* is part of any Java development environment.
- Various classes we've already used (`System` , `Scanner`, `String`) are part of the Java standard class library.
- Other class libraries can be obtained through third party vendors, or you can create them yourself.

Packages

- The classes of the Java standard class library are organized into *packages.*

- Some of the packages in the standard class library are:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name.*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name.
  - This is required by code convention.

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character.
  - This is against the code convention.

```
import java.util.*;
```

The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs.
- It's as if all programs contain the following line:

```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs.
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported.

Creating Objects

Creating Objects

- A variable holds either a primitive type or a *reference* to an object.
- A class name can be used as a type to declare an *object reference variable.*

```
String title;
```

- No object is created with this declaration.
- An object reference variable holds the address of an object.
- The object itself must be created separately.

Creating Objects

- Generally, we use the `new` operator to create an object.

```
title = new String ("Java Software Solutions");
```

This calls the String *constructor*, which is a special method that sets up the object

- Creating an object is called *instantiation.*
- An object is an *instance* of a particular class.

Invoking Methods

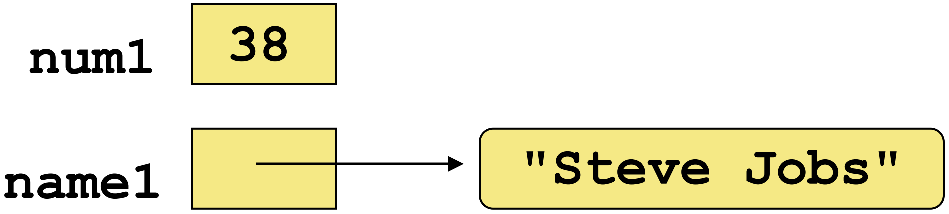
- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods.

```
count = title.length();
```

- A method may *return a value*, which can be used in an assignment or expression.
  - A value return method
- A method invocation can be thought of as asking an object to perform a service.
  - A void method

References

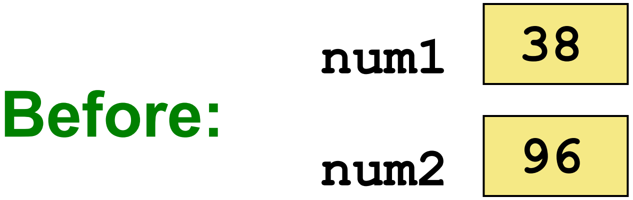
- Note that a primitive variable contains the value itself, but an object variable contains the address of the object.
- An object reference can be thought of as a pointer to the location of the object.
- Rather than dealing with arbitrary addresses, we often depict a reference graphically.



Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable.

- For primitive types:

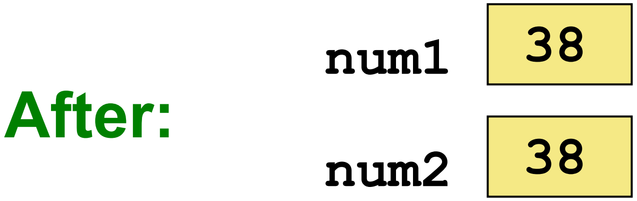


Before:

num1 38

num2 96

```
num2 = num1;
```



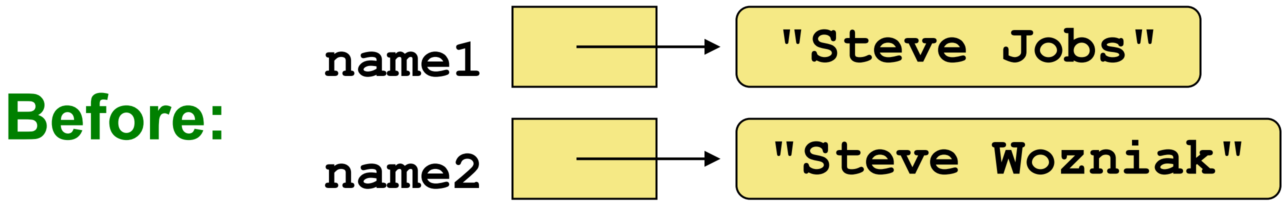
After:

num1 38

num2 38

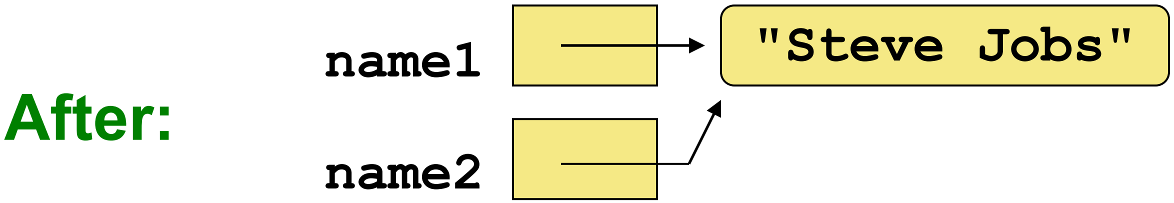
Reference Assignment

- For object references, assignment copies the address:



Before:

```
name2 = name1;
```



After:

Aliases

- Two or more references that refer to the same object are called *aliases* of each other.
- That creates an interesting situation: one object can be accessed using multiple reference variables.
- Aliases can be useful but should be managed carefully.
- Changing an object through one reference changes it for all its aliases, because there is really only one object.

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program.
- The object is useless, and therefore is called *garbage.*
- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use.
- In other languages, the programmer is responsible for performing garbage collection.

The String Class

The String Class

- Because strings are so common, we don't have to use the `new` operator to create a `String` object.

```
title = "Java Software Solutions";
```

- This is special syntax that works only for strings.

- Each string literal (enclosed in double quotes) represents a `String` object.

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed.
- Thus, we say that an object of the `String` class is *immutable.*
- However, several methods of the `String` class return **new** `String` objects that are modified versions of the original.
  - The original object is unchanged.

length()

- Returns the number of characters contained in the string object .

```
 String palindrome = "Dot saw I was Tod";
int len = palindrome.length(); //17
```

concat (String str)

- Concatenates two strings.

  ```
  string1.concat(string2);
  ```

  - This returns a new string that is string1 with string2 added to it at the end.

- For example,

  ```
  My name is Rumplestiltskin
  ```

  ```
  "My name is ".concat("Rumplestiltskin")

  "My name is " + "Rumplestiltskin“
  ```

  - String concatenator `+` is equivalent to `concat`.

String Indexes

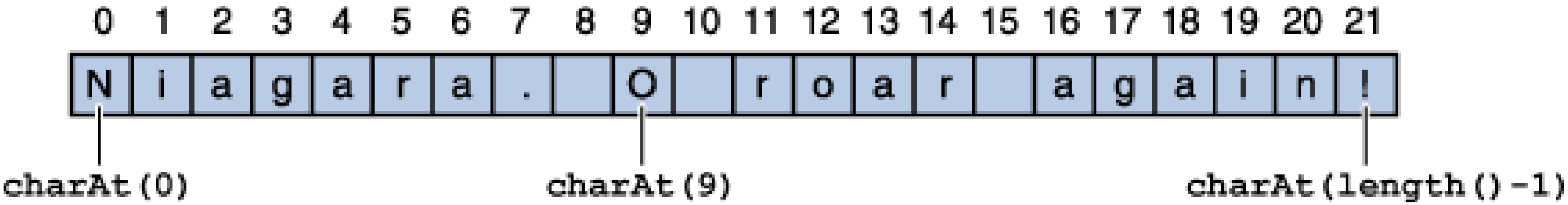
- It is occasionally helpful to refer to a particular character within a string.
- This can be done by specifying the character's numeric *index.*
- The indexes begin at zero in each string.
- In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4.

0	1	2	3	4
H	**e**	**l**	**L**	**o**

charAt(int index)

- Gets the character at a particular index within a string.

```
String  anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

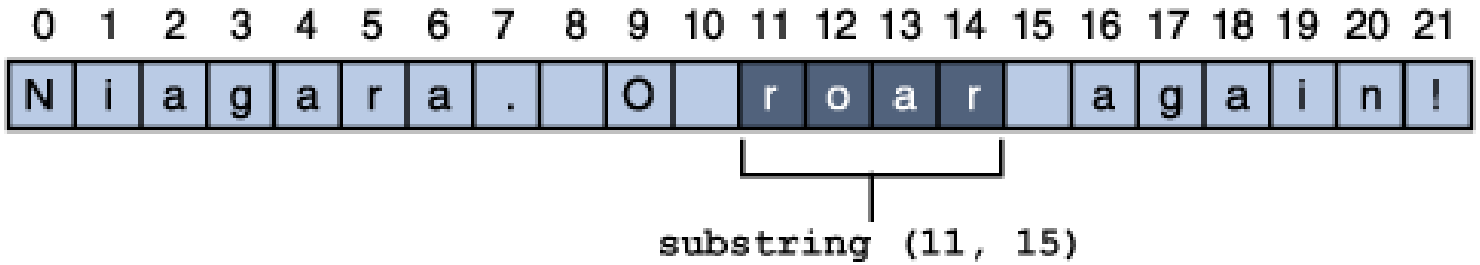


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
N	i	a	g	a	r	a	.		O		r	o	a	r		a	g	a	i	n	!

substring (int beginIndex, int endIndex)

- Returns a new string that is a substring of this string.
  - The first integer argument specifies the index of the first character.
  - The second integer argument is the index of the last character + 1 – the index to stop.

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11, 15);
```

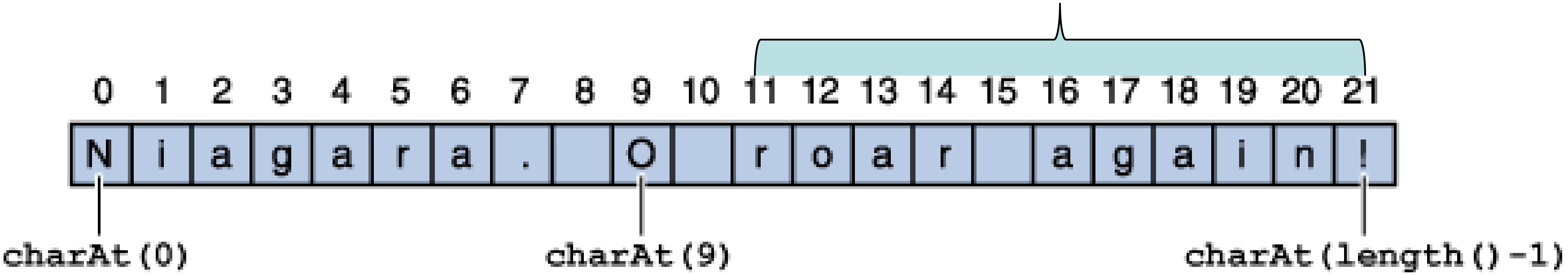


substring (int beginIndex)

- Returns a new string that is a substring of this string.
  - The integer argument specifies the index of the first character. Here, the returned substring extends to the end of the original string.

roar again!

```
String anotherPalindrome = "Niagara. O roar again!";
String roar = anotherPalindrome.substring(11);
```



toLowerCase()

- Returns a copy of this string converted to lowercase. If no conversions are necessary, this method returns the original string.

```
String name = “SMITH";
System.out.println(name.toLowerCase());
```

toUpperCase()

- Returns a copy of this string converted to uppercase. If no conversions are necessary, this method returns the original string.

```
String name = “smith";
 System.out.println(name.toUpperCase());
```

replace(char oldChar, char newChar)

- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

```
String name = “smith";
System.out.println(name.replace(‘s’, ‘S’));
```

equals(String str)

- Returns true if and only if the argument `str` is a String object that represents the same sequence of characters or has the same content as this object.

```
String name = “smith";
String name2 = “Smith";
System.out.println(name.equals(name2));
```

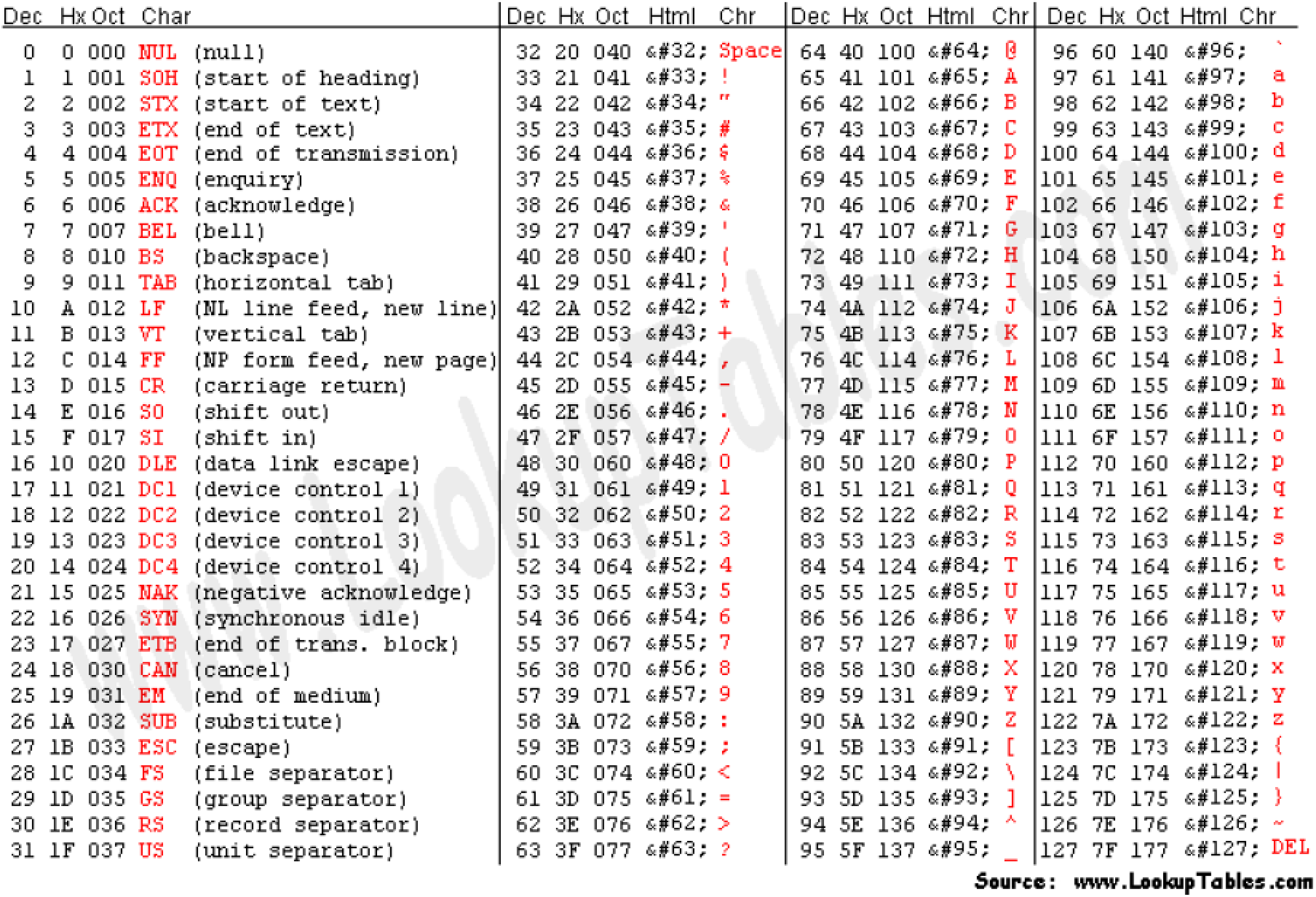
False

  - The corresponding pair of characters are compared. Stop at the first difference.

s	m	i	t	h
S	m	i	t	h

integer

character



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

compareTo(String anotherString)

- Compares two strings lexicographically(similar to dictionary order).
- Returns an integer result. This string is
  - greater than another string if the result is > 0.
  - equal to another string is the result is = 0.
  - less than another string if the result is < 0.

```
 String name = “smith";
 String name2 = “Smith";
```

115-83 = 32

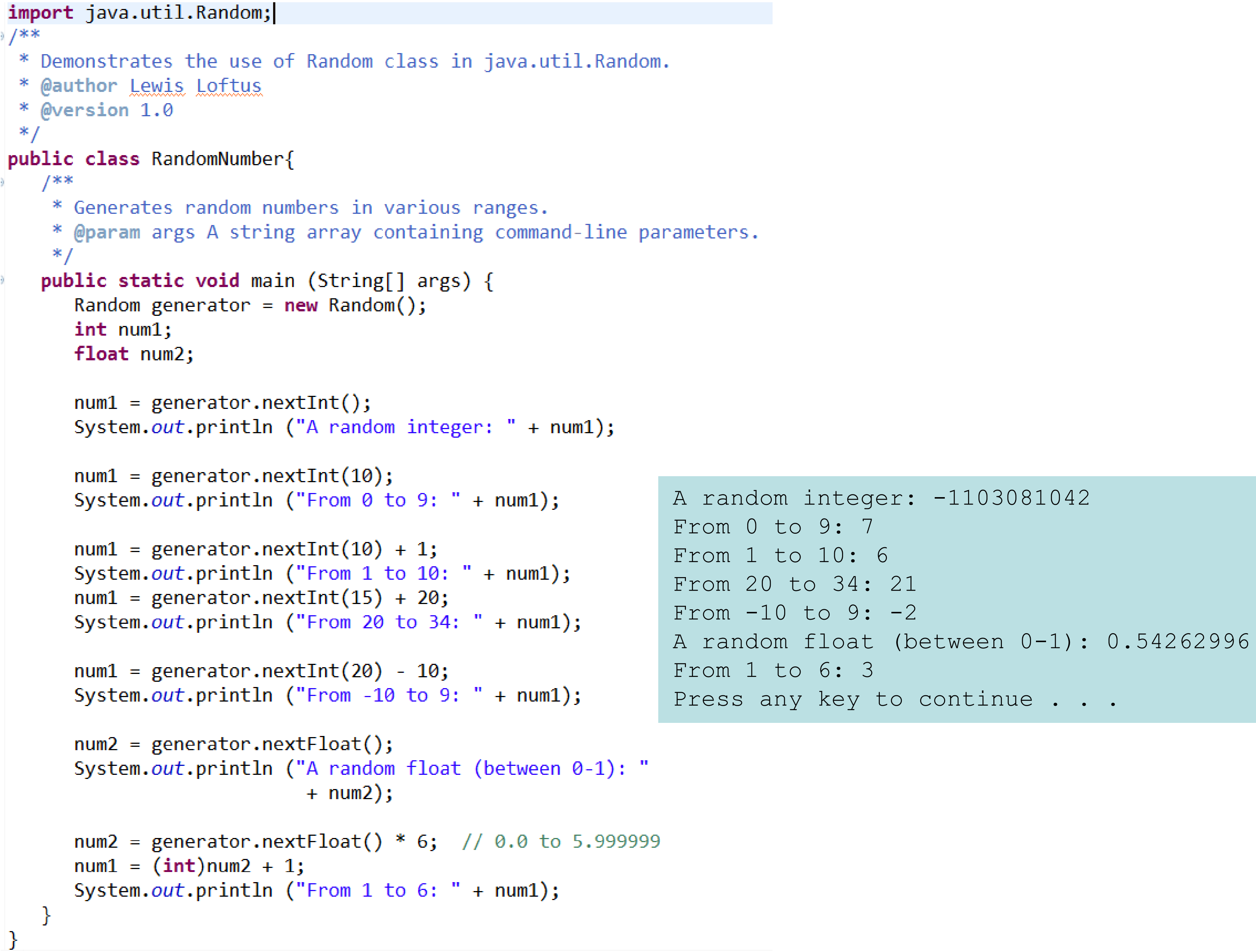
32

```
System.out.println(name.compareTo(name2));
```

The Random Class

The Random Class

- The `Random` class is part of the `java.util` package.
- It provides methods that generate pseudorandom numbers.
- A `Random` object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values.



```java
import java.util.Random;
/**
 * Demonstrates the use of Random class in java.util.Random.
 * @author Lewis Loftus
 * @version 1.0
 */
public class RandomNumber{
   /**
    * Generates random numbers in various ranges.
    * @param args A string array containing command-line parameters.
    */
   public static void main (String[] args) {
      Random generator = new Random();
      int num1;
      float num2;

      num1 = generator.nextInt();
      System.out.println ("A random integer: " + num1);

      num1 = generator.nextInt(10);
      System.out.println ("From 0 to 9: " + num1);

      num1 = generator.nextInt(10) + 1;
      System.out.println ("From 1 to 10: " + num1);
      num1 = generator.nextInt(15) + 20;
      System.out.println ("From 20 to 34: " + num1);

      num1 = generator.nextInt(20) - 10;
      System.out.println ("From -10 to 9: " + num1);

      num2 = generator.nextFloat();
      System.out.println ("A random float (between 0-1): "
                          + num2);

      num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
      num1 = (int)num2 + 1;
      System.out.println ("From 1 to 6: " + num1);
   }
}
```

```
A random integer: -1103081042
From 0 to 9: 7
From 1 to 10: 6
From 20 to 34: 21
From -10 to 9: -2
A random float (between 0-1): 0.54262996
From 1 to 6: 3
Press any key to continue . . .
```

Formatting Output

- DecimalFormat Class
- NumberFormat Class

Formatting Output

- It is often necessary to format values in certain ways so that they can be presented properly.
- The Java standard class library contains classes that provide formatting capabilities.
- The `NumberFormat` class allows you to format values as currency or percentages.
- The `DecimalFormat` class allows you to format values based on a pattern.
- Both are part of the `java.text` package.

Formatting Output (`DecimalFormat`)

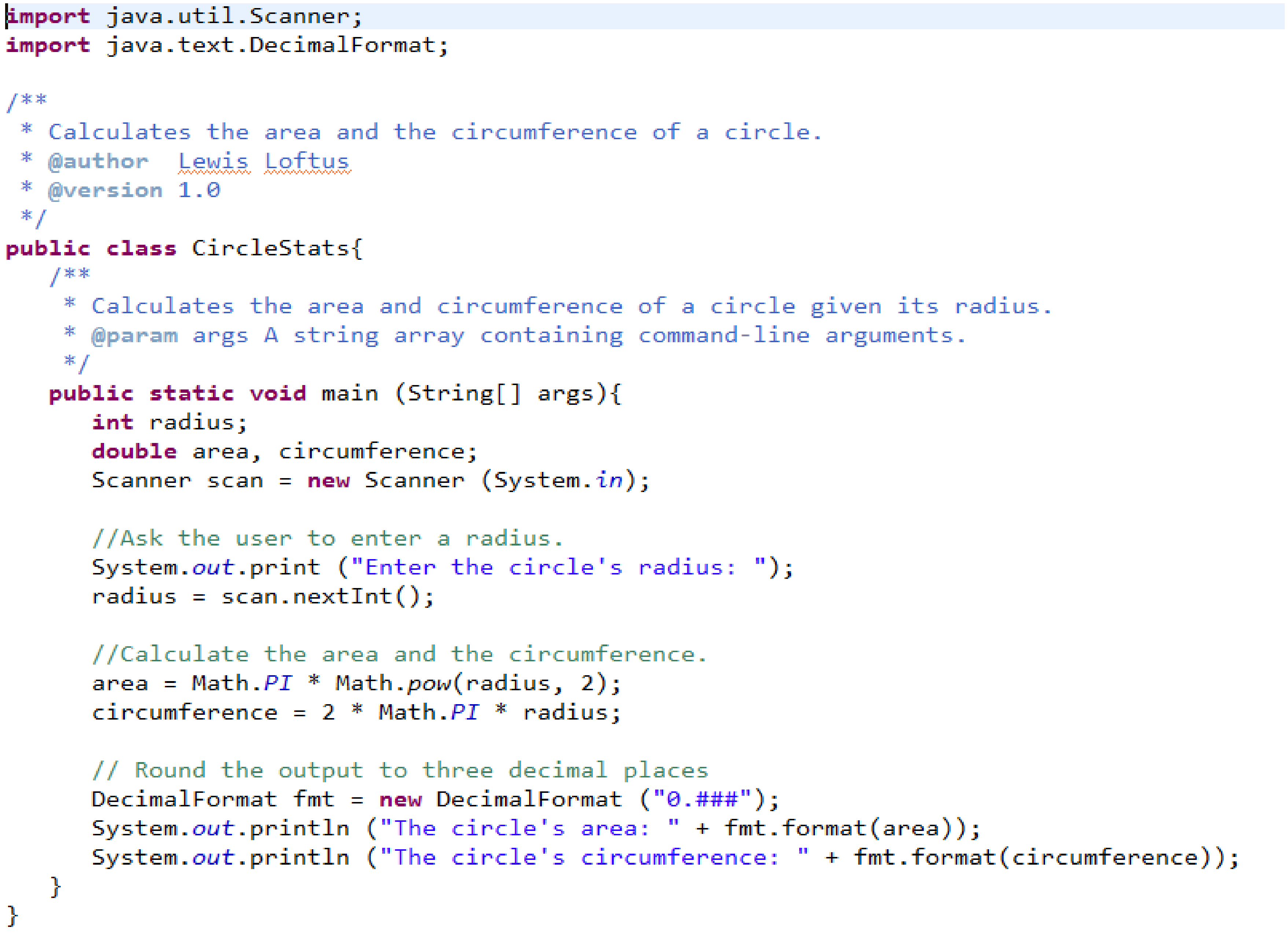
- The `DecimalFormat` class can be used to format a floating point value in various ways.
- For example, you can specify that the number should be truncated to three decimal places.
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number.

Formatting Output (`DecimalFormat`)

Special Pattern Characters		
Symbol	Location	Meaning
0	Number	Digit
#	Number	Digit, zero shows as absent
.	Number	Decimal separator or monetary decimal separator
-	Number	Minus sign
,	Number	Grouping separator
%	Prefix or suffix	Multiply by 100 and show as percentage
‘	Prefix or suffix	Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock".

Math class

- ***pow*** is static method from Math class in `java.lang` package. It must be called via the class name.
- For example, `pow(a,b)` returns a to the power of b - a^b.
  - `Math.pow(12,34)`
- **PI**: a static variable in the same class. A static variable must be called via the class name.
  - `Math.PI`



```java
import java.util.Scanner;
import java.text.DecimalFormat;

/**
 * Calculates the area and the circumference of a circle.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class CircleStats{
   /**
    * Calculates the area and circumference of a circle given its radius.
    * @param args A string array containing command-line arguments.
    */
   public static void main (String[] args){
      int radius;
      double area, circumference;
      Scanner scan = new Scanner (System.in);

      //Ask the user to enter a radius.
      System.out.print ("Enter the circle's radius: ");
      radius = scan.nextInt();

      //Calculate the area and the circumference.
      area = Math.PI * Math.pow(radius, 2);
      circumference = 2 * Math.PI * radius;

      // Round the output to three decimal places
      DecimalFormat fmt = new DecimalFormat ("0.###");
      System.out.println ("The circle's area: " + fmt.format(area));
      System.out.println ("The circle's circumference: " + fmt.format(circumference));
   }
}
```

Formatting Output (`NumberFormat`)

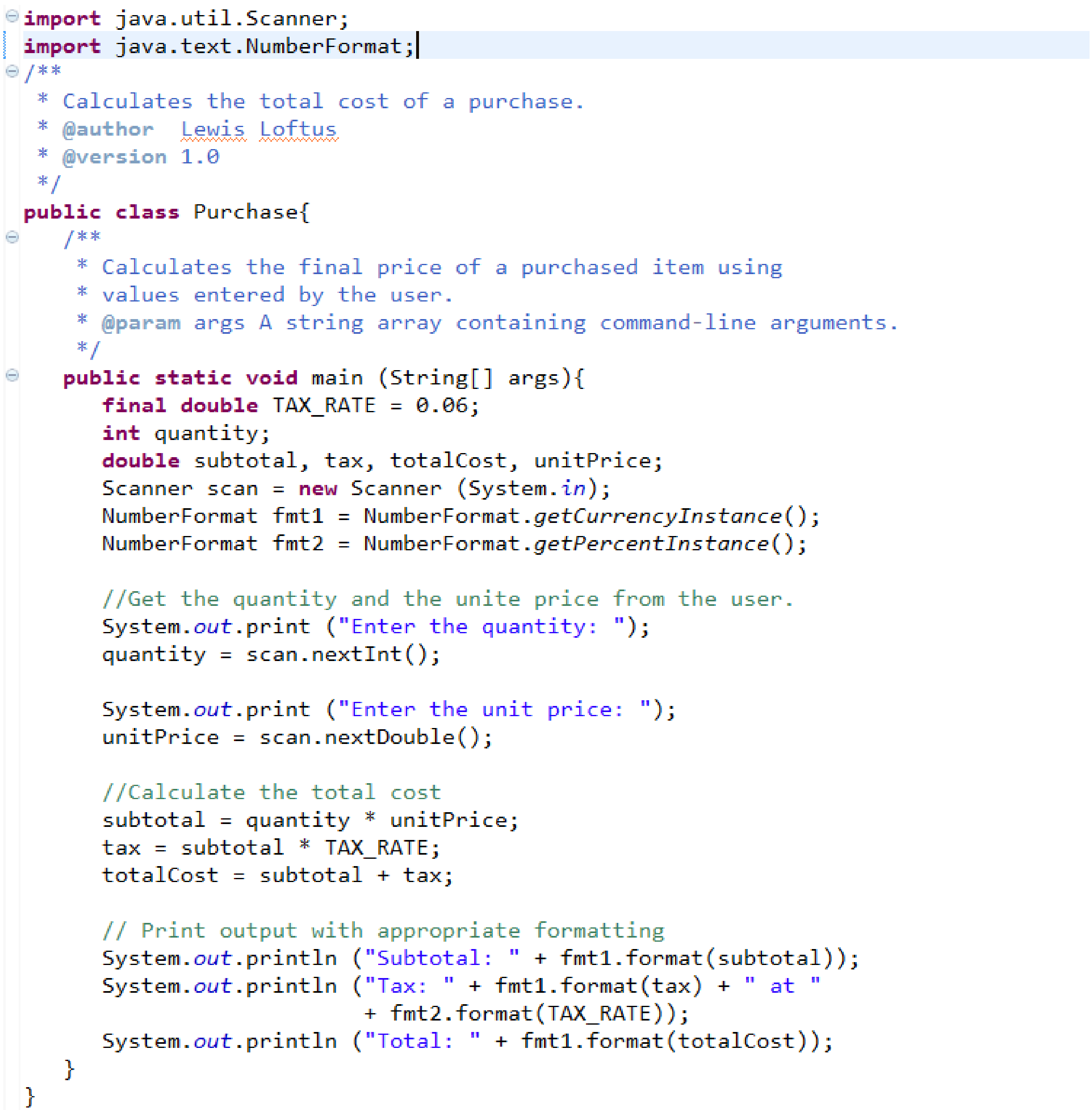
- The `NumberFormat` class has static methods that return a formatter object.

```
getCurrencyInstance()

getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format.

Formatting Output (`NumberFormat`)

- A number to be formatted into a currency is rounded to the hundredth.
- A number to formatted into a percent is multiplied by 100, then rounded to one.



```java
import java.util.Scanner;
import java.text.NumberFormat;
/**
 * Calculates the total cost of a purchase.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class Purchase{
   /**
    * Calculates the final price of a purchased item using
    * values entered by the user.
    * @param args A string array containing command-line arguments.
    */
   public static void main (String[] args){
      final double TAX_RATE = 0.06;
      int quantity;
      double subtotal, tax, totalCost, unitPrice;
      Scanner scan = new Scanner (System.in);
      NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
      NumberFormat fmt2 = NumberFormat.getPercentInstance();

      //Get the quantity and the unite price from the user.
      System.out.print ("Enter the quantity: ");
      quantity = scan.nextInt();

      System.out.print ("Enter the unit price: ");
      unitPrice = scan.nextDouble();

      //Calculate the total cost
      subtotal = quantity * unitPrice;
      tax = subtotal * TAX_RATE;
      totalCost = subtotal + tax;

      // Print output with appropriate formatting
      System.out.println ("Subtotal: " + fmt1.format(subtotal));
      System.out.println ("Tax: " + fmt1.format(tax) + " at "
                          + fmt2.format(TAX_RATE));
      System.out.println ("Total: " + fmt1.format(totalCost));
   }
}
```

Summary

- Character Strings
- Expressions
- Primitive Data Types
- Variables and Assignment
- Interactive Programs
- Data Conversion
- Packages
- Creating Objects
- The String Class, the Random class
- Format output