

Java Fundamentals

Lecture 2a

Topics (1 of 2)

- The Parts of a Java Program
- The `print` and `println` Methods, and the Java API
- Variables and Literals
- Primitive Data Types
- Arithmetic Operators
- Combined Assignment Operators
- Conversion between Primitive data types

Topics (2 of 2)

- Creating named constants with `final`
- The `String` class
- Scope
- Comments
- Programming style
- Using the `Scanner` class for input
- Dialog boxes
- Common Errors to Avoid

Parts of a Java Program (1 of 3)

- A Java source code file contains one or more Java classes.
- If more than one class is in a source code file, only one of them may be public.
- The public class and the filename of the source code file must match.
 - ex: A class named `Simple` must be in a file named `Simple.java`
- Each Java class can be separated into parts.

Parts of a Java Program (2 of 3)

- **Example:** `Simple.java`

```
1 // This is a simple Java program.
2
3 public class Simple
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Programming is great fun!");
8     }
9 }
```

Parts of a Java Program (3 of 3)

- To compile the example:
 - `javac Simple.java`
 - Notice the `.java` file extension is needed.
 - This will result in a file named `Simple.class` being created.
- To run the example:
 - `java Simple`
 - Notice there is no file extension here.
 - The *java* command assumes the extension is `.class`.

Analyzing the Example (1 of 3)

```
// This is a simple Java program.
```

This is a Java comment. It is ignored by the compiler.

```
public class Simple
```

This is the class header for the class Simple

```
{
```

This area is the body of the class Simple. All of the data and methods for this class will be between these curly braces.

```
}
```

Analyzing the Example (2 of 3)

```
// This is a simple Java program.
```

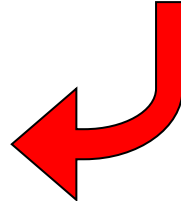
```
public class Simple  
{
```

```
    public static void main(String [] args)  
    {
```


```
    }
```

```
}
```

This is the method header for the main method. The main method is where a Java application begins.



This area is the body of the main method. All of the actions to be completed during the main method will be between these curly braces.



Analyzing the Example (3 of 3)

```
// This is a simple Java program.
```

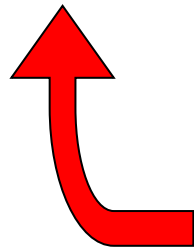
```
public class Simple  
{
```

```
    public static void main(String [] args)  
    {
```

```
        System.out.println("Programming is great fun!");
```

```
    }
```

```
}
```



**This is the Java Statement that
is executed when the program runs.**

Parts of a Java Program (1 of 7)

- Comments
 - The line is ignored by the compiler.
 - The comment in the example is a single-line comment.
- Point to ponder #1

How many comments can you add to your Java program?

As many as you want to make your code more understandable for you and others!

Parts of a Java Program (2 of 7)

- Comments

What is this Java program doing?

```
int f=1, n=5;
for(int i=1;i<=n;i++){
    f=f*i;
}
return f;
```

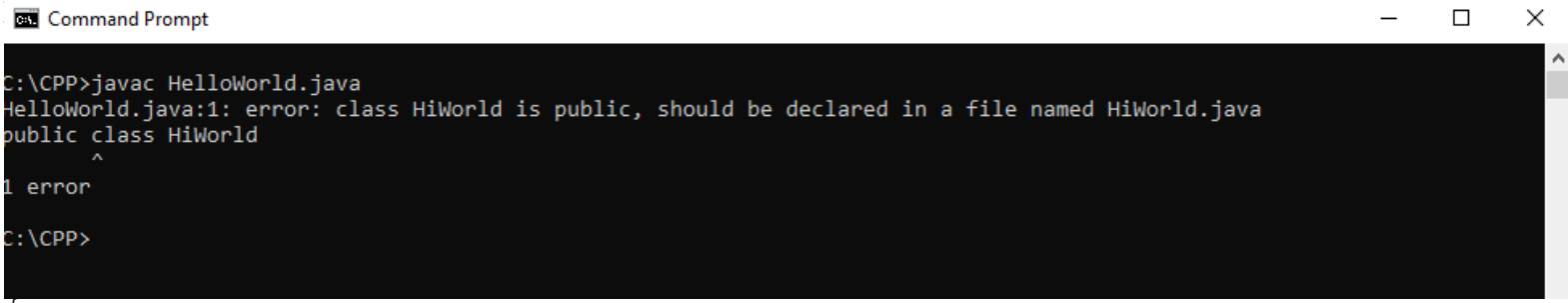
How about now?

```
// Calculating and returning the factorial of 5
int fact=1;                                // Initializing the variable fact for the factorial
for(int i=5;i>0;i--){                      // looping from 1 until 5 (5 iterations)
    fact=fact*i;                           // accumulating the products in fact
}
return fact;                               // return the calculated factorial
```

Parts of a Java Program (3 of 7)

- Class Header
 - The class header tells the compiler things about the class such as what other classes can use it (**public**) and that it is a Java class (**class**), and the name of that class (**Simple**).
- Point to ponder #2

What happen if the name of the public class is different from the file name that includes the class, and you try to compile this program?



```
Command Prompt
C:\CPP>javac HelloWorld.java
HelloWorld.java:1: error: class HiWorld is public, should be declared in a file named HiWorld.java
public class HiWorld
      ^
1 error
C:\CPP>
```

Parts of a Java Program (4 of 7)

- Curly Braces

- When associated with the class header, they define the scope of the class.
- When associated with a method, they define the scope of the method.

- Point to ponder #3

Is this a valid Java program?

```
public class Simple  
{
```

```
}}
```



How about that?

```
public class Simple  
{}
```



Parts of a Java Program (5 of 7)

- The `main` Method

- This line must be exactly as shown in the example (except the `args` variable name can be programmer defined).
- This is the line of code that the `java` command will run first.
- This method starts the Java program.
- Every Java application must have a `main` method.

- Java Statements

- When the program runs, the statements within the `main` method will be executed.

Parts of a Java Program (6 of 7)

- Point to ponder #4

What is the output of this application?

```
public class Number
{
    public static void begin(String [] args)
    {
        System.out.println("1");
    }

    public static void main(String [] args)
    {
        System.out.println("2");
    }

    public static void start(String [] args)
    {
        System.out.println("3");
    }
}
```



Parts of a Java Program (7 of 7)

- Point to ponder #5

Is this a valid Java program?



```
public class Number
{
    public static void begin(String [] args)
    {
        System.out.println("1");
    }
}
```

- Is it a Java application?
- Why?



No main method.

Java Statements (1 of 2)

- If we look back at the `Simple.java` example, we can see that there is only one line that ends with a semi-colon.

```
System.out.println("Programming is great fun!");
```

- This is because it is the only Java statement in the program.
- The rest of the code is either a comment or other Java framework code.

Java Statements (2 of 2)

- Comments are ignored by the Java compiler, so they need no semi-colons.
- Other Java code elements that do not need semi colons include:
 - class headers
 - Terminated by the code within its curly braces.
 - method headers
 - Terminated by the code within its curly braces.
 - curly braces
 - Part of framework code that needs no semi-colon termination.

Short Review (1 of 2)

- Java is a case-sensitive language.
- All Java programs must be stored in a file with a `.java` file extension.
- Comments are ignored by the compiler.
- A `.java` file may contain many classes but may only have one public class.
- If a `.java` file has a public class, the class must have the same name as the file.

Short Review (2 of 2)

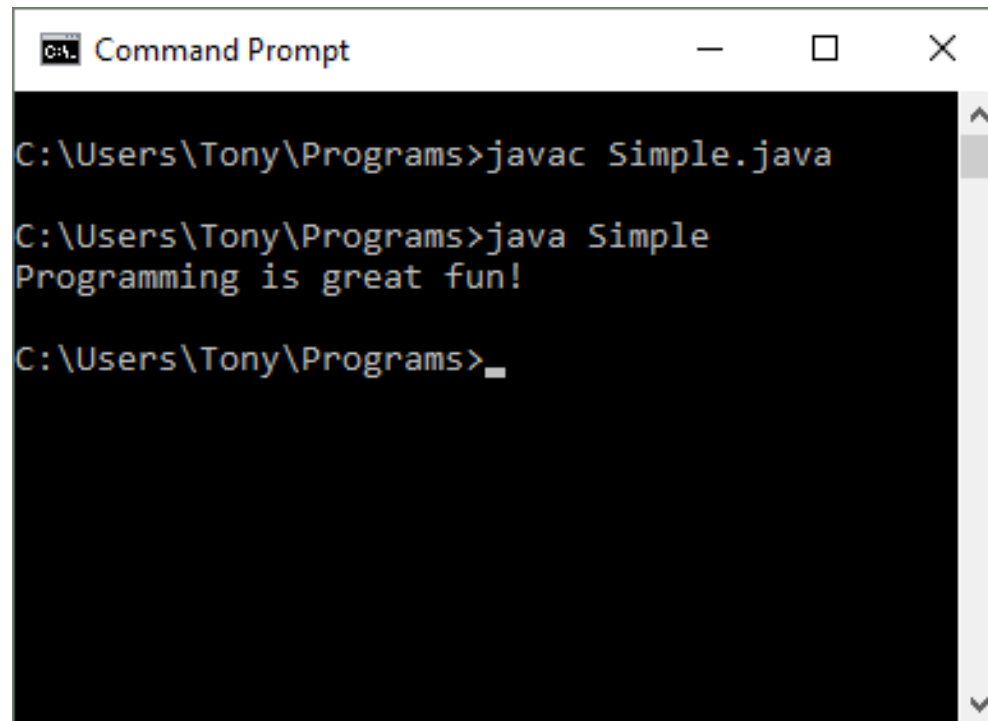
- Java applications must have a `main` method.
- For every left brace, or opening brace, there must be a corresponding right brace, or closing brace.
- Statements are terminated with semicolons.
 - Comments, class headers, method headers, and braces are not considered Java statements.

Special Characters

//	double slash	Marks the beginning of a single line comment.
()	open and close parenthesis	Used in a method header to mark the <i>parameter list</i> .
{ }	open and close curly braces	Encloses a group of statements, such as the contents of a class or a method.
“ ”	quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen
;	semi-colon	Marks the end of a complete programming statement

Console Output (1 of 9)

- Many of the programs that you will write will run in a console window.



```
C:\Users\Tony\Programs>javac Simple.java

C:\Users\Tony\Programs>java Simple
Programming is great fun!

C:\Users\Tony\Programs>_
```

Console Output (2 of 9)

- The console window, which displayed only text, is known as *the standard output* device.
- The *standard input* device is typically the keyboard.
- Java sends information to the standard output device by using a Java class stored in the standard Java library.

Console Output (3 of 9)

- Java classes in the standard Java library are accessed using the Java Application Programming Interface (API).
- The standard Java library is commonly referred to as the *Java API*.

Console Output (4 of 9)

- The previous example uses the line:

```
System.out.println("Programming is great fun!");
```

- This line uses the `System` class from the standard Java library.
- The `System` class contains methods and objects that perform system level tasks.
- The `out` object, a member of the `System` class, contains the methods `print` and `println`.

Console Output (5 of 9)

- The `print` and `println` methods actually perform the task of sending characters to the output device.

- The line:

```
System.out.println("Programming is great fun!");
```

is pronounced: System dot out dot println ...

- The value inside the parenthesis will be sent to the output device (in this case, a string).

Console Output (6 of 9)

- The `println` method places a newline character at the end of whatever is being printed out.
- The following lines:

```
System.out.println("This is being printed out");  
System.out.println("on two separate lines.");
```

will output:

```
This is being printed out  
on two separate lines
```

Console Output (7 of 9)

- The `print` statement works very similarly to the `println` statement.
- However, the `print` statement does not put a newline character at the end of the output.
- The lines:

```
System.out.print("These lines will be");  
System.out.print("printed on");  
System.out.println("the same line.");
```

will output:

```
These lines will beprinted onthe same line.
```

Notice the odd spacing? Why are some words together?

Console Output (8 of 9)

- For all of the previous examples, we have been printing out strings of characters.
- Later, we will see that much more can be printed.
- There are some special characters that can be put into the output.

```
System.out.print("This line will have a newline at the end.\n");
```

- The `\n` in the string is an escape sequence that represents the newline character.
- Escape sequences allow the programmer to print characters that otherwise would be unprintable.

Console Output (9 of 9)

- The old way - typewriters



Just use `println` or add `\n` at the end of your string!

Java Escape Sequences (1 of 4)

<code>\n</code>	newline	Advances the cursor to the next line for subsequent printing
<code>\t</code>	tab	Causes the cursor to skip over to the next tab stop
<code>\b</code>	backspace	Causes the cursor to back up, or move left, one position
<code>\r</code>	carriage return	Causes the cursor to go to the beginning of the current line, not the next line
<code>\\</code>	backslash	Causes a backslash to be printed
<code>\'</code>	single quote	Causes a single quotation mark to be printed
<code>\"</code>	double quote	Causes a double quotation mark to be printed

Java Escape Sequences (2 of 4)

- Even though the escape sequences are comprised of two characters, they are treated by the compiler as a single character.

```
System.out.print("These are our top sellers:\n");  
System.out.print("\tComputer games\n\tCoffee\n");  
System.out.println("\tAspirin");
```

Would result in the following output:

```
These are our top sellers:  
    Computer games  
    Coffee  
    Aspirin
```

- With these escape sequences, complex text output can be achieved.

Java Escape Sequences (3 of 4)

- Point to ponder #6

How to print the quote “To be, or not to be, that is the question”?

```
System.out.print("\nTo be, or not to be,");  
System.out.print("that is the question");
```



```
System.out.print("To be, or not to be, ");  
System.out.print("that is the question");
```



```
System.out.print("\nTo be, or not to be, ");  
System.out.print("that is the question");
```



Java Escape Sequences (4 of 4)

- Point to ponder #7

How to print the sequence “\n is a line break”?

```
System.out.print("\n is a line break");
```



```
System.out.print("\\n is a line break");
```



Java Fundamentals

Lecture 2b

Topics (1 of 2)

- The Parts of a Java Program
- The `print` and `println` Methods, and the Java API
- Variables and Literals
- Primitive Data Types
- Arithmetic Operators
- Combined Assignment Operators
- Conversion between Primitive data types

Topics (2 of 2)

- Creating named constants with `final`
- The `String` class
- Scope
- Comments
- Programming style
- Using the `Scanner` class for input
- Dialog boxes
- Common Errors to Avoid

Variables and Literals (1 of 5)

- A variable is a named storage location in the computer's memory.
- A literal is a value that is written into the code of a program.
- Programmers determine the number and type of variables a program will need.
- See example: `Variable.java`

Variables and Literals (2 of 5)

```
1 // This program has a variable.
2
3 public class Variable
4 {
5     public static void main(String[] args)
6     {
7         int value;
8
9         value = 5;
10        System.out.print("The value is ");
11        System.out.println(value);
12    }
13 }
```

Variables and Literals (3 of 5)

This line is called
a *variable declaration*.

```
int value;
```

The following line is known
as an *assignment statement*.

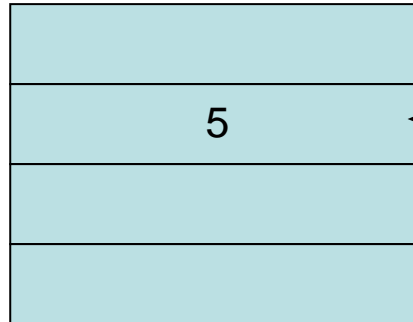
```
value = 5;
```

0x000

0x001

0x002

0x003



The value 5
is stored in
memory.

This is a string *literal*. It will be printed **as is**.

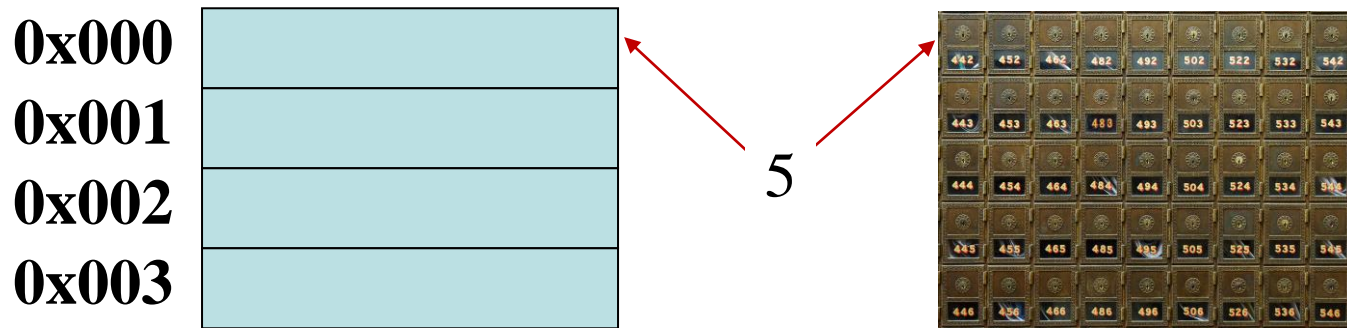
```
System.out.print("The value is ");  
System.out.println(value);
```

The integer 5 will
be printed out here.
Notice no quote marks?

Variables and Literals (4 of 5)

- Point to ponder #1

What is the similarity between a variable in the computer's memory and a Po Box in the Post Office?



- Both designate a space (memory address or box number) where you can store content that can be changed over time.

Variables and Literals (5 of 5)

- Point to ponder #2

What is the output of this program?

```
1 public class Variable
2 {
3     public static void main(String[] args)
4     {
5         int value;
6
7         value = 5;
8         System.out.print("The value is ");
9         System.out.println("value");
10    }
11 }
```

The value is value

```
1 public class Variable
2 {
3     public static void main(String[] args)
4     {
5         int value;
6
7         value = 5;
8         System.out.print("The value is ");
9         System.out.println(value);
10    }
11 }
```

The value is 5

The + Operator

- The + operator can be used in two ways.
 - as a concatenation operator
 - as an addition operator
- If either side of the + operator is a string, the result will be a string.

```
System.out.println("Hello " + "World");
```

```
-> Hello World
```

```
System.out.println("The value is: " + 5);
```

```
-> The value is: 5
```

```
System.out.println("The value is: " + value);
```

```
-> The value is: 5
```

```
System.out.println("The value is: " + "\n" + 5);
```

```
-> The value is: /n5
```

```
System.out.println("5" + 5);
```

```
-> 55
```

String Concatenation (1 of 3)

- Java commands that have string literals must be treated with care.
- A string literal value cannot span lines in a Java source code file.

```
System.out.println("This line is too long  
and now it has spanned more than one  
line, which will cause a syntax error to  
be generated by the compiler. ");
```

String Concatenation (2 of 3)

- The String concatenation operator can be used to fix this problem.

```
System.out.println("These lines are " +  
                    "are now ok and will not " +  
                    "cause the error as before.");
```

- String concatenation can join various data types.

```
System.out.println("We can join a string to " +  
                    "a number like this: " + 5);
```

String Concatenation (3 of 3)

- Be careful with Quotation Marks. Placing double quotation marks around anything that is not intended to be a string literal will create an error of some type.

```
1 public class Variable
2 {
3     public static void main(String[] args)
4     {
5         int value;
6
7         value = "5";
8         System.out.print("The value is " + value);
9
10    }
11 }
```

- Point to ponder #3

Is this a compile-time error or a run-time error in Java?

Compile-time error.

Identifiers (1 of 6)

- Identifiers are programmer-defined names that represent:
 - classes
 - variables
 - methods
- Identifiers may not be any of the Java reserved keywords.

Identifiers (2 of 6)

- Choose names for your variables that give an indication of what they are used for.
- For instance: `int numberStudents` is better than `int nS` and much better than `int x`
- The name `numberStudents` gives anyone reading the program an idea of what the variable is used for creating:

self-documenting programs!

Identifiers (3 of 6)

- Point to ponder #4

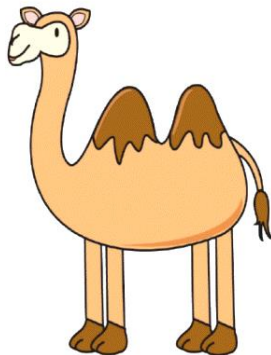
So, what is a self-documenting program?

One that allows the programmer to understand it just by reading its code!

Identifiers (4 of 6)

- Point to ponder #5

What is this convention that chooses names for variables that begin with a lowercase letter, and after that, the first letter of each individual word that makes up the variable name is capitalized?



Lower Camel Case!

`numberStudents`

instead of

`numberstudents`

`numberGraduateStudents`

instead of

`numbergraduatestudents`

Identifiers (5 of 6)

- Identifiers must follow certain rules:
 - An identifier may only contain:
 - letters a–z or A–Z,
 - the digits 0–9,
 - underscores (`_`), or
 - the dollar sign (`$`)
 - The first character may not be a digit.
 - Identifiers are case sensitive.
 - `numberStudents` is not the same as `numberstudents`.
 - Identifiers cannot include spaces.

Identifiers (6 of 6)

- Point to ponder #6

Variable Name	Legal or Illegal?
dayOfWeek	
3dGraph	
june1997	
mixture#3	
week day	

Overall Java Naming Conventions

- Variable and methods names should follow the lower camel case convention:

Ex: `int caTaxRate, findAverage()`

- Class names should follow the upper camel case convention.

Ex: `public class BigLittle`

- More Java naming conventions can be found at:

<http://java.sun.com/docs/codeconv/html/CodeConvention.s.doc8.html>

- A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.

Java Reserved Keywords

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

Primitive Data Types (1 of 2)

- Each variable has a data type, which is the type of data that the variable can hold.
- Selecting the proper data type is important because a variable's data type determines the amount of memory the variable uses.

Primitive Data Types (2 of 2)

- Point to ponder #7

Why are they called primitive?

You cannot use them to create objects. You can only create variables to hold a single value, with no attributes or methods.

- There are 8 Java primitive data types.

Numeric Data Types

byte	1 byte	Integers in the range -128 to $+127$
short	2 bytes	Integers in the range of $-32,768$ to $+32,767$
int	4 bytes	Integers in the range of $-2,147,483,648$ to $+2,147,483,647$
long	8 bytes	Integers in the range of $-9,223,372,036,854,775,808$ to $+9,223,372,036,854,775,807$
float	4 bytes	Floating-point numbers in the range of $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$, with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of $\pm 1.7 \times 10^{-308}$ to $\pm 3.4 \times 10^{308}$, with 15 digits of accuracy

Variable Declarations

- Variable Declarations take the following form:
 - *DataType VariableName;*
- Examples of variable declarations:
 - `byte inches;`
 - `short month;`
 - `int speed;`
 - `long timeStamp;`
 - `float salesCommission;`
 - `double distance;`

Integer Data Types (1 of 2)

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.

See Example: `IntegerVariables.java`

Integer Data Types (2 of 2)

- Point to ponder #8

Can I declare several variables of the same type simply by separating their names with commas?

For instance: `int length, width, area;`



This is the same of:

```
int length;  
int width;  
int area;
```

Integer Literals

- When whole numbers are embedded into Java source code, they are called integer literals. The default type for integer literals is `int`.
 - -20, 105, 120, and 189000 are `int` data types.
- You can force an integer literal to be treated as a long, by suffixing it with the letter L. For example, the value 57L would be treated as a long.

```
long meters = 1000L;
```

Java Fundamentals

Lecture 2c

Topics (1 of 2)

- The Parts of a Java Program
- The `print` and `println` Methods, and the Java API
- Variables and Literals
- Primitive Data Types
- Arithmetic Operators
- Combined Assignment Operators
- Conversion between Primitive data types

Topics (2 of 2)

- Creating named constants with `final`
- The `String` class
- Scope
- Comments
- Programming style
- Using the `Scanner` class for input
- Dialog boxes
- Common Errors to Avoid

Floating Point Data Types

- Data types that allow fractional values are called *floating-point* numbers.
 - 1.7 and -45.316 are floating-point numbers.
- In Java there are two data types that can represent floating-point numbers.
 - `float` - also called *single precision* (7 decimal points).
 - `double` - also called *double precision* (15 decimal points).

Floating Point Literals (1 of 4)

- When floating point numbers are embedded into Java source code, they are called *floating point literals*.
- The default type for floating point literals is `double`.
 - 29.75, 1.76, and 31.51 are `double` data types.

Floating Point Literals (2 of 4)

- See example: `Sale.java`

```
1 // This program demonstrates the double data type.
2
3 public class Sale
4 {
5     public static void main(String[] args)
6     {
7         double price, tax, total;
8
9         price = 29.75;
10        tax = 1.76;
11        total = 31.51;
12        System.out.println("The price of the item " +
13        "is " + price);
14        System.out.println("The tax is " + tax);
15        System.out.println("The total is " + total);
16    }
17 }
```

Floating Point Literals (3 of 4)

- A `double` value is not compatible with a `float` variable because of its size and precision.
 - `float number;`
 - `number = 23.5; // Error!`
- A `double` can be forced into a `float` by appending the letter `F` or `f` to the literal.
 - `float number;`
 - `number = 23.5F; // This will work.`
- Remember, Java is a *strongly-typed* language!

Floating Point Literals (4 of 4)

- Literals cannot contain embedded currency symbols or commas.
 - `grossPay = $1,257.00; // ERROR!`
 - `grossPay = 1257.00; // Correct.`
- Floating-point literals can be represented in *scientific notation*.
 - $47,281.97 == 4.728197 \times 10^4$.
- Java uses *E notation* to represent values in scientific notation.
 - $4.728197 \times 10^4 == 4.728197\text{E}4$.

Scientific and E Notation (1 of 3)

Decimal Notation	Scientific Notation	E Notation (Java)
247.91	2.4791×10^2	2.4791E2
0.00072	7.2×10^{-4}	7.2E-4
2,900,000	2.9×10^6	2.9E6

Scientific and E Notation (2 of 3)

See example: `SunFacts.java`

```
1 // This program uses E notation.
2
3 public class SunFacts
4 {
5     public static void main(String[] args)
6     {
7         double distance, mass;
8
9         distance = 1.495979E11;
10        mass = 1.989E30;
11        System.out.println("The sun is " + distance +
12        " meters away.");
13        System.out.println("The sun's mass is " + mass +
14        " kilograms.");
15    }
16 }
```

Scientific and E Notation (3 of 3)

- Point to ponder #1

What is the name of the number 10^{100} or

[illegible]

1 googol. That was the inspiration of Larry Page and Sergey Brin to name "Google Inc.", when the name was accidentally misspelled by one Stanford's graduate student. It was picked to signify that the search engine was intended to provide large quantities of information.



The `boolean` Data Type (1 of 2)

- The Java `boolean` data type can have two possible values.
 - `true`
 - `false`
- The value of a `boolean` variable may only be copied into a `boolean` variable (not into a variable of any type other than `boolean`).

The boolean Data Type (2 of 2)

See example: `TrueFalse.java`

```
1 // A program for demonstrating boolean variables
2
3 public class TrueFalse
4 {
5     public static void main(String[] args)
6     {
7         boolean bool;
8
9         bool = true;
10        System.out.println(bool);
11        bool = false;
12        System.out.println(bool);
13    }
14 }
```

The boolean Data Type (2 of 2)

- Point to ponder #2

How many bits/bytes can we store in a boolean variable?

1 bit

The `char` Data Type (1 of 2)

- The Java `char` data type provides access to *single characters*.
- `char` literals are enclosed in *single quotation marks*.
 - `'a'`, `'Z'`, `'1'`, `'x'`
- Don't confuse `char` literals with string literals.
 - `char` literals are enclosed in single quotes.
 - `String` literals are enclosed in double quotes.

```
char letter;  
letter = "A"; // Error!
```

The char Data Type (2 of 2)

See example: Letters.java

```
1 // This program demonstrates the char data type.
2
3 public class Letters
4 {
5     public static void main(String[] args)
6     {
7         char letter;
8
9         letter = 'A';
10        System.out.println(letter);
11        letter = 'B';
12        System.out.println(letter);
13    }
14 }
```

Unicode (1 of 6)

- Internally, characters are stored as numbers.
- Character data in Java is stored as Unicode (standard) characters.
- Each Unicode character requires 2 bytes in memory.
- Thus, the Unicode character set (basic multilingual) can consist of 65536 (2^{16}) individual characters.
- The first 256 characters in the Unicode character set are compatible with the ASCII (American Standard Code for Information Interchange) character set.

Unicode (2 of 6)

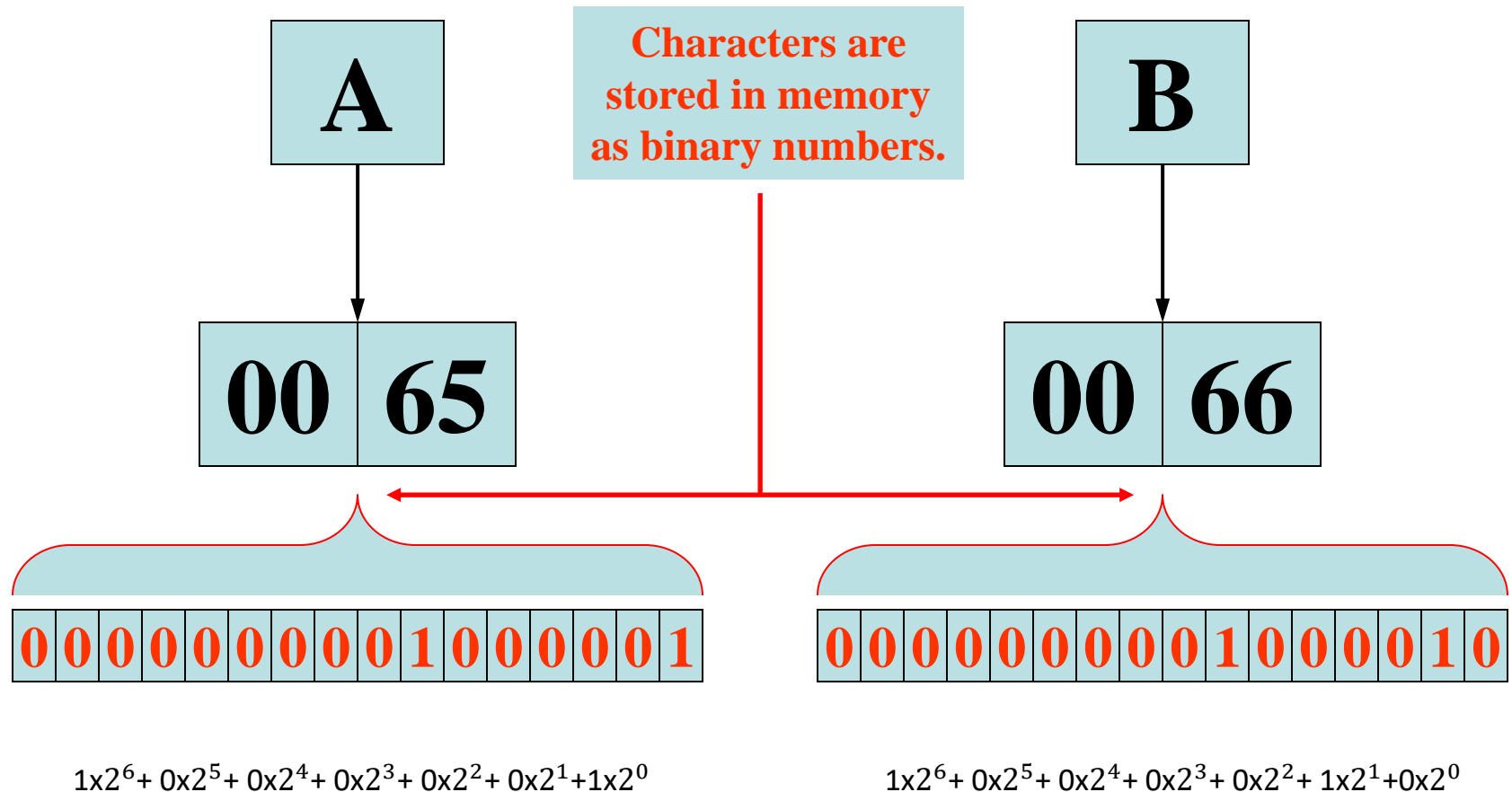
Code	Character	Code	Character	Code	Character	Code	Character	Code	Character
0	NUL	26	SUB	52	4	78	N	104	h
1	SOH	27	Escape	53	5	79	O	105	i
2	STX	28	FS	54	6	80	P	106	j
3	ETX	29	GS	55	7	81	Q	107	k
4	EOT	30	RS	56	8	82	R	108	l
5	ENQ	31	US	57	9	83	S	109	m
6	ACK	32	(Space)	58	:	84	T	110	n
7	BEL	33	!	59	;	85	U	111	o
8	Backspace	34	"	60	<	86	V	112	p
9	HTab	35	#	61	=	87	W	113	q
10	Line Feed	36	\$	62	>	88	X	114	r
11	VTAB	37	%	63	?	89	Y	115	s
12	Form Feed	38	&	64	@	90	Z	116	t
13	CR	39	'	65	A	91	[117	u
14	SO	40	(66	B	92	\	118	v
15	SI	41)	67	C	93]	119	w
16	DLE	42	*	68	D	94	^	120	x
17	DC1	43	+	69	E	95	_	121	y
18	DC2	44	,	70	F	96	`	122	z
19	DC3	45	-	71	G	97	a	123	{
20	DC4	46	.	72	H	98	b	124	
21	NAK	47	/	73	I	99	c	125	}
22	SYN	48	0	74	J	100	d	126	~
23	ETB	49	1	75	K	101	e	127	DEL
24	CAN	50	2	76	L	102	f		
25	EM	51	3	77	M	103	g		

Unicode (3 of 6)

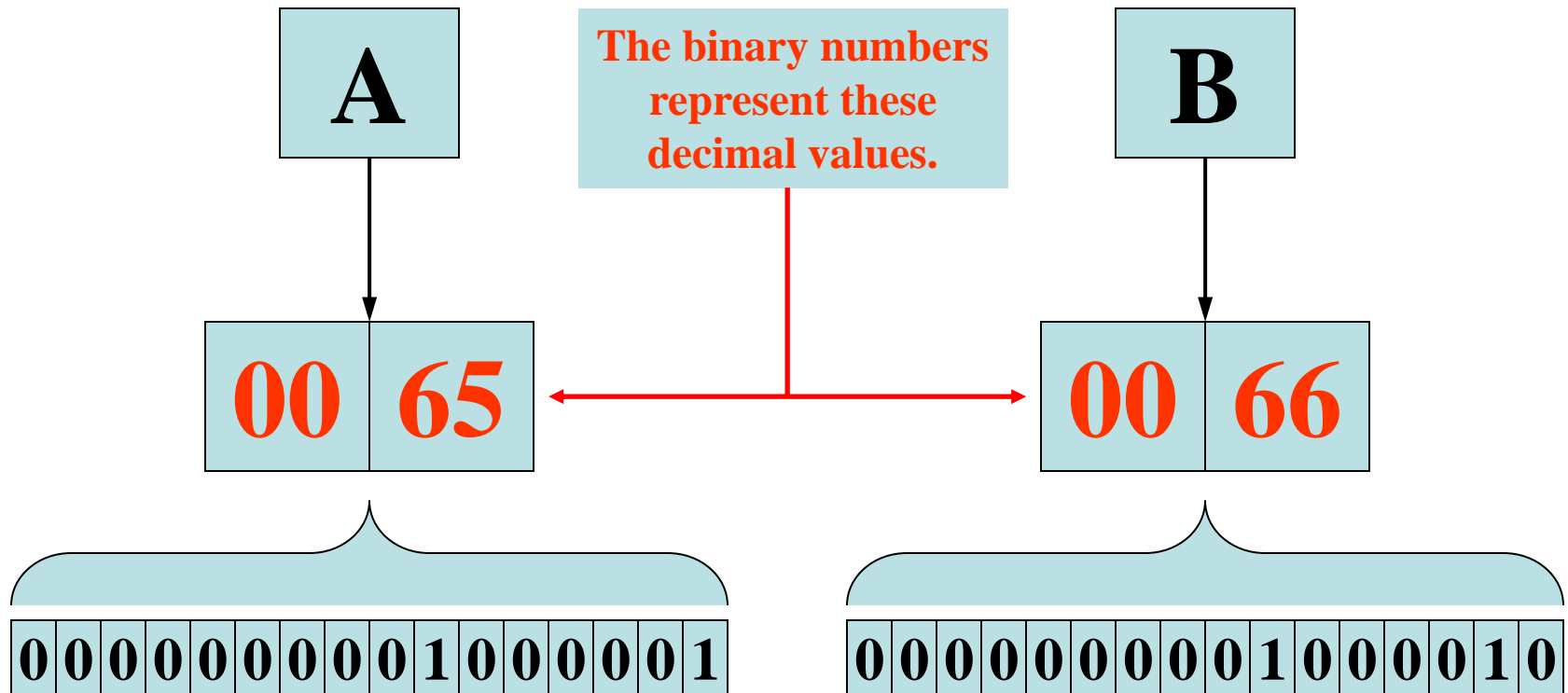
See example: Letters2.java

```
1 // This program demonstrates the close relationship between
2 // characters and integers.
3
4 public class Letters2
5 {
6     public static void main(String[] args)
7     {
8         char letter;
9
10        letter = 65;
11        System.out.println(letter);
12        letter = 66;
13        System.out.println(letter);
14    }
15 }
```

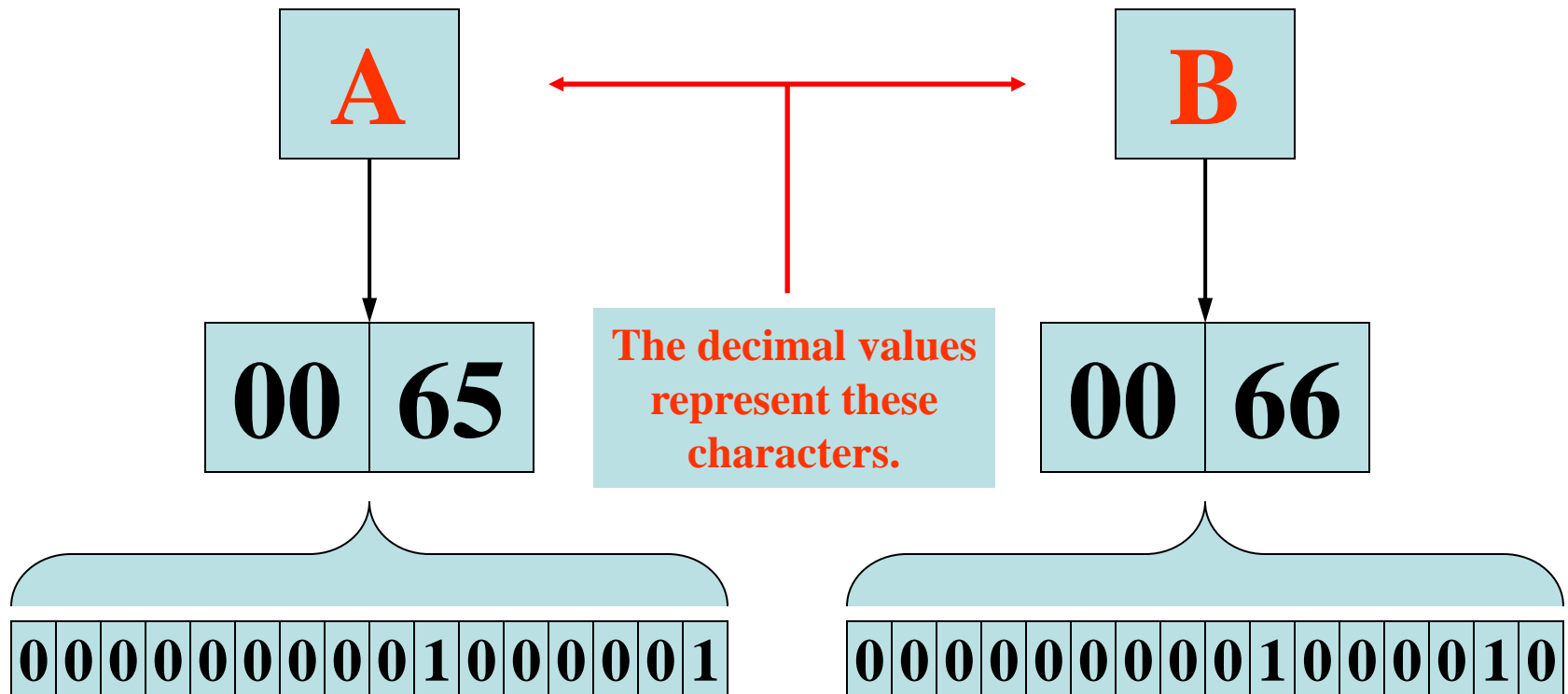

Unicode (4 of 6)



Unicode (5 of 6)



Unicode (6 of 6)



Creating Constants with `final` (1 of 3)

- Many programs have data that does not need to be changed.
- Littering programs with literal values can make the program hard to read and maintain.
- Replacing literal values with constants remedies this problem.
- Constants are variables whose value cannot be changed (read only).

Creating Constants with `final` (2 of 3)

- Constants are declared using the keyword `final`.
- By convention, constants are all uppercase and words are separated by the underscore character.

```
final double INTEREST_RATE;
```

- Once initialized with a value, constants cannot be changed programmatically (compile-time error!).

```
final double INTEREST_RATE = 0.069;  
INTEREST_RATE = 0.089;
```



Creating Constants with `final` (3 of 3)

- Consider the code:

```
amount = balance * 0.069;
```

- The code below is much clear and maintainable.

```
amount = balance * INTEREST_RATE;
```

- Point to ponder #3. Why?
 - Constants allow the programmer to use a name rather than a value throughout the program.
 - Constants also give a singular point for changing those values when needed.

The Scanner Class (1 of 4)

- To read input from the keyboard we can use the `Scanner` class.
- The `Scanner` class is defined in `java.util`, so we will use the following statement at the top of our programs:

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

The Scanner Class (2 of 4)

- Scanner objects work with `System.in` which refers to the standard input device. Those objects retrieve the input formatted as primitive values or strings.
- To create a Scanner object:

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

- The Scanner class methods:

- `nextByte()` // returns input as a byte.
- `nextDouble()` // returns input as a double.
- `nextFloat()` // returns input as a float.
- `nextInt()` // returns input as an int.
- `next()` // returns input as a String (until the space only).
- `nextLine()` // returns input as a String (the entire input).
- `nextLong()` // returns input as a long.
- `nextShort()` // returns input as a short.

The Scanner Class (3 of 4)

See example: Payroll.java

```
import java.util.Scanner; // Needed for the Scanner class

public class Payroll
{
    public static void main(String[] args)
    {
        String name; // To hold a name
        int hours; // Hours worked
        double payRate; // Hourly pay rate
        double grossPay; // Gross pay

        // Create a Scanner object to read input.
        Scanner keyboard = new Scanner(System.in);

        // Get the user's name.
        System.out.print("What is your name? ");
        name = keyboard.nextLine();

        // Get the number of hours worked this week.
        System.out.print("How many hours did you work this week? ");
        hours = keyboard.nextInt();

        . . .

        . . .

        // Get the user's hourly pay rate.
        System.out.print("What is your hourly pay rate? ");
        payRate = keyboard.nextDouble();

        // Calculate the gross pay.
        grossPay = hours * payRate;

        // Display the resulting information.
        System.out.println("Hello, " + name);
        System.out.println("Your gross pay is $" + grossPay);
    }
}
```

The Scanner Class (4 of 4)

- Closing a Scanner object after using it

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

```
.  
.   
.
```

```
input.close();
```

It is important to call `close()` to free up resources.

Variable Assignment and Initialization

(1 of 8)

- In order to store a value in a variable, an *assignment statement* must be used.
- The *assignment operator* is the equal (=) sign.
- The operand on the left side of the assignment operator must be a variable name.
- The operand on the right side must be either a literal, another variable or expression that evaluates to a type that is compatible with the type of the variable.

Variable Assignment and Initialization (2 of 8)

- Point to ponder #4

Valid or invalid?

- `int 12 = width;`



- `int length = 12;`



- `int width = 12;`



- `width = length;`



- `width = 10 + 3;`



- `length = "Hi";`



Variable Assignment and Initialization (3 of 8)

```
// This program shows variable assignment.

public class Initialize
{
    public static void main(String[] args)
    {
        int month, days;

        month = 2;
        days = 28;
        System.out.println("Month " + month + " has " +
                           days + " Days.");
    }
}
```

The variables must be declared before they can be used.

Variable Assignment and Initialization (4 of 8)

```
// This program shows variable assignment.

public class Initialize
{
    public static void main(String[] args)
    {
        int month, days;

        month = 2;
        days = 28;
        System.out.println("Month " + month + " has " +
                           days + " Days.");
    }
}
```

**Once declared, they can then receive a value (assignment);
However, the value must be compatible with the variable's
declared type.**

Variable Assignment and Initialization (5 of 8)

```
// This program shows variable assignment.

public class Initialize
{
    public static void main(String[] args)
    {
        int month, days;

        month = 2;
        days = 28;
        System.out.println("Month " + month + " has " +
                           days + " Days.");
    }
}
```

After receiving a value, the variables can then be used in output statements or in other calculations.

Variable Assignment and Initialization (6 of 8)

```
// This program shows variable initialization.

public class Initialize
{
    public static void main(String[] args)
    {
        int month = 2, days = 28;
        System.out.println("Month " + month + " has " +
                           days + " Days.");
    }
}
```

Local variables can be declared and initialized on the same line.

Variable Assignment and Initialization

(7 of 8)

- Variables can only hold one value at a time.
- Local variables (those declared inside a method) do not receive a default value.
- Local variables must have a valid type in order to be used.

```
public static void main(String [] args)
{
    int month, days; //No value given...
    System.out.println("Month " + month + " has " +
                       days + " Days.");
}
```

Trying to use uninitialized variables will generate a Syntax Error when the code is compiled.

Variable Assignment and Initialization (8 of 8)

- Point to ponder #5

What is the value of `x` that will be printed after the execution of line 2 and 4?

```
1 int x = 5;  
2 System.out.println(x);      5  
3 x = 99;  
4 System.out.println(x);      99
```

Arithmetic Operators (1 of 4)

- Java has five (5) arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	<code>total = cost + tax;</code>
-	Subtraction	Binary	<code>cost = total - tax;</code>
*	Multiplication	Binary	<code>tax = cost * rate;</code>
/	Division	Binary	<code>salePrice = original / 2;</code>
%	Modulus	Binary	<code>remainder = value % 5;</code>

Arithmetic Operators (2 of 4)

- The binary operators must have two operands.
- Each operator must have a left and right operand.
- It is an error to try to divide any number by zero.

Arithmetic Operators (3 of 4)

See example: `Wages.java`

```
public class Wages
{
    public static void main(String[] args)
    {
        double regularWages; // The calculated regular wages.
        double basePay = 25; // The base pay rate.
        double regularHours = 40; // The hours worked less overtime.
        double overtimeWages; // Overtime wages
        double overtimePay = 37.5; // Overtime pay rate
        double overtimeHours = 10; // Overtime hours worked
        double totalWages; // Total wages

        regularWages = basePay * regularHours;
        overtimeWages = overtimePay * overtimeHours;
        totalWages = regularWages + overtimeWages;
        System.out.println("Wages for this week are $" + totalWages);
    }
}
```

Arithmetic Operators (4 of 4)

- The arithmetic operators work as one would expect.

```
amount = 4 + 8;           // Assigns 12 to amount
temperature = 112 - 14;    // Assigns 98 to temperature
markUp = 12 * 0.25;        // Assigns 3 to markUp
points = 100 / 20;         // Assigns 5 to points
leftOver = 17 % 3;         // Assigns 2 to leftOver
```

Unary and Ternary Operators

- Unary operators require only a single operand.

Operator	Meaning	Type	Example
-	Negation	Unary	-5; -number;

- Other unary operators and the Java ternary operator will be discussed later.

Integer Division

- When working with two integer operands, the division operator requires special attention.
- Division can be tricky.
 - In a Java program, what is the value of $5/2$?
- You might think the answer is 2.5.
- But, that's wrong.
- The answer is simply 2.
- Integer division will throw away the fractional part of the result (it will be truncated).

Integer Division

- It does not matter that the variable that will be assigned the result is declared as a `double`

```
double number = 5/2;
```

```
System.out.println(number)    2.0
```

- To return a floating-point value, one of the division operands must be of a floating-point data type.

```
double number = 5.0/2;
```

```
System.out.println(number)    2.5
```

Integer Division

- Point to ponder #5
Valid or invalid?

```
int number = 5.0/2;
```



```
int number = 5/2;
```



```
float number = 5.0/2;
```



Operator Precedence (1 of 2)

- Mathematical expressions can be very complex.
- There is a set order in which arithmetic operations will be carried out.

Higher Priority	Operator	Associativity	Example	Result
	- (unary negation)	Right to left	$x = -4 + 3;$	-1
	* / %	Left to right	$x = -4 + 4 \% 3 * 13 + 2;$	11
Lower Priority	+ -	Left to right	$x = 6 + 3 - 4 + 6 * 3;$	23

Grouping with Parenthesis

- When parenthesis are used in an expression, the inner most parenthesis are processed first.
- If two sets of parenthesis are at the same level, they are processed left to right.

The diagram illustrates the order of evaluation for the expression `x = ((4*5) / (5-2)) - 25;`. It uses four numbered curly braces to show the sequence: 1. `4*5`, 2. `5-2`, 3. `((4*5) / (5-2))`, and 4. `((4*5) / (5-2)) - 25`. The numbers 1, 2, 3, and 4 are written in red below their respective braces.

```
x = ((4*5) / (5-2)) - 25; // result = -19
```

- Point to ponder #6
What would be the result without the parentheses?

-23

The Math Class (1 of 2)

- Java provides a class named `Math` which contains useful methods for complex mathematical operations;

- The `Math.pow()` method. It takes two double arguments and raises the first argument to the power of the second argument, returning the result as a double.

```
result = Math.pow(4.0, 2.0); //result = 16.0
```

- The `Math.sqrt()` method. It accepts a double value as its argument and returns the square root of the value as a double.

```
result = Math.sqrt(9.0); //result = 3.0
```

- The `Math.max()` method. It accepts two integer, long, float or double arguments and returns the greater number among them. The type of the result is the same of the arguments.

```
result = Math.max(8,5); //result = 8
```

- Others: `Math.abs()`, `Math.min()`, `Math.log10()`, etc.

The Math.PI Named Constant

- Java provides a predefined constant `Math.PI`, which is assigned the value `3.14159265358979323846` (approximation of the mathematical value pi).
- To calculate the area of the circle:

```
double area = Math.PI * radius * radius;
```

Java Fundamentals

Lecture 2d

Combined Assignment Operators (1 of 3)

- Java has some combined assignment operators.
- These operators allow the programmer to perform an arithmetic operation and assignment with a single operator.
- Although not required, these operators are popular since they shorten simple equations.

Combined Assignment Operators (2 of 3)

Operator	Example	Equivalent	Value of variable after operation
+=	<code>x += 5;</code>	<code>x = x + 5;</code>	The old value of x plus 5.
-=	<code>y -= 2;</code>	<code>y = y - 2;</code>	The old value of y minus 2
*=	<code>z *= 10;</code>	<code>z = z * 10;</code>	The old value of z times 10
/=	<code>a /= b;</code>	<code>a = a / b;</code>	The old value of a divided by b .
%=	<code>c %= 3;</code>	<code>c = c % 3;</code>	The remainder of the division of the old value of c divided by 3.

Combined Assignment Operators (3 of 3)

- Point to ponder #1

```
int x = 5;  
System.out.println(x);           5  
x += 3;  
System.out.println(x);           8  
x -= 1;  
System.out.println(x);           7  
x %= 3;  
System.out.println(x);           1
```

Conversion between Primitive Data Types

(1 of 6)

- Java performs some conversions between data types automatically if the conversion will not result in the loss of data
- Remember, Java is a *strongly typed language*
- Point to ponder #2

What does that mean?

Java checks the data types of the variable and the value being assigned to it to determine whether they are compatible.

Conversion between Primitive Data Types

(2 of 6)

- When the Java compiler encounters this line of code, it will respond with an error message

```
int x;  
double y = 2.5;  
x = y;
```



- However, not all assignment statements that mix data types are rejected by the compiler

```
int x;  
short y = 2;  
x = y;
```



Conversion between Primitive Data Types (3 of 6)

- The primitive (numeric) data types are ranked. When values of lower-ranked data types are assigned to variables of higher-ranked data types, Java automatically performs the conversion.

double

Highest Rank



float

long

int

short

byte



widening conversion
(automatically performed by Java)

Lowest Rank

Conversion between Primitive Data Types (4 of 6)

- When values of higher-ranked data types are assigned to variables of lower-ranked data types, Java does not automatically perform the conversion, and an error is generated.

double

float

long

int

short

byte

Highest Rank



Lowest Rank



narrowing conversion
(not automatically done by Java)

Conversion between Primitive Data Types

(5 of 6)

- In case you still need to perform the narrowing conversion, you should use the *cast operator*

data_type_1 x = (data_type_1) number;

Statement	Description
<code>littleNum = (short)bigNum;</code>	The cast operator returns the value in <code>bigNum</code> , converted to a <code>short</code> . The converted value is assigned to the variable <code>littleNum</code> .
<code>x = (long)3.7;</code>	The cast operator is applied to the expression <code>3.7</code> . The operator returns the value <code>3</code> , which is assigned to the variable <code>x</code> .
<code>number = (int)72.567;</code>	The cast operator is applied to the expression <code>72.567</code> . The operator returns <code>72</code> , which is used to initialize the variable <code>number</code> .
<code>value = (float)x;</code>	The cast operator returns the value in <code>x</code> , converted to a <code>float</code> . The converted value is assigned to the variable <code>value</code> .
<code>value = (byte)number;</code>	The cast operator returns the value in <code>number</code> , converted to a <code>byte</code> . The converted value is assigned to the variable <code>value</code> .

Conversion between Primitive Data Types

(6 of 6)

- Java also performs *promotion* automatically when operands are of different data types
- For example, if `sum` is a `float` and `count` is an `int`, the value of `count` is converted to a `float` to perform the following calculation:

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

- If `sum` is a `double/long`, the value of `count` will be converted to a `double/long`.

The `String` Class

- Java has no primitive data type that holds a series of characters.
- The `String` class from the Java standard library is used for this purpose.
- In order to be useful, a variable must be created to reference a `String` object.

```
String number;
```

- Notice the `S` in `String` is upper case.
- By convention, class names should always begin with an upper-case character.

Primitive vs. Class Type Variables (1 of 2)

- Primitive variables actually contain the value that they have been assigned.

```
int number = 25;
```

- The value 25 will be stored in the memory location associated with the variable `number`.

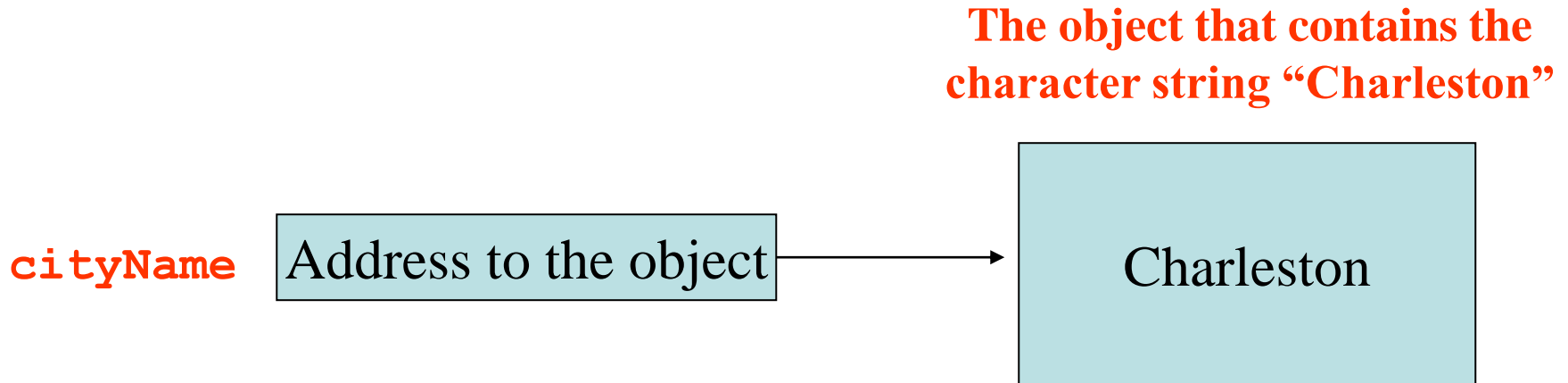
number

25

Primitive vs. Class Type Variables (2 of 2)

- Objects (instances of classes) are not stored in variables. Objects are *referenced* by variables.
- When a variable references an object, it contains the memory address of the object's location.
- Then, it is said that the variable *references* the object.

```
String cityName = "Charleston";
```



String Objects (1 of 4)

- Any time you write a string literal in your program, Java will create a String object in memory to hold it.
- You can create a String object in memory and store its address in a String variable with a simple assignment of a String literal.

```
String value = "Hello";
```

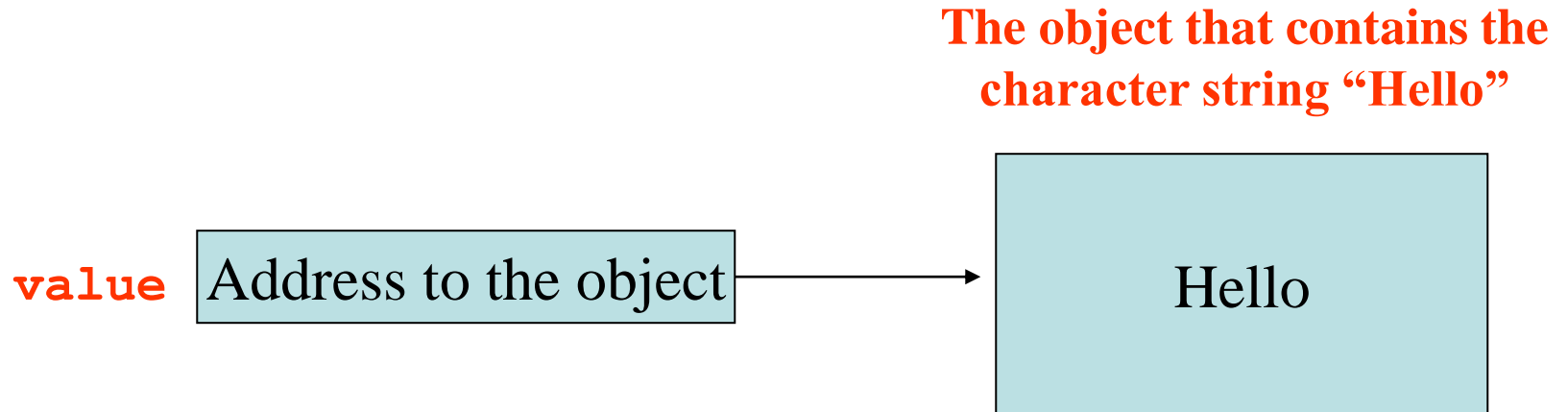
- Strings are the only objects that can be created in this way.

String Objects (2 of 4)

- A String variable can be created using the *new* keyword.

```
String value = new String("Hello");
```

- This is the method that all other objects must use when they are created.



String Objects (3 of 4)

See example: `StringDemo.java`

```
1 // A simple program demonstrating String objects.
2
3 public class StringDemo
4 {
5     public static void main(String[] args)
6     {
7         String greeting = "Good morning, ";
8         String name = "Herman";
9
10        System.out.println(greeting + name);
11    }
12 }
```

String Objects (4 of 4)

- `String` objects are *immutable*, meaning that they cannot be changed.

```
String name = "Hello";  
name = " World"
```

A red oval callout containing the text "2 objects here!".

2 objects here!

- For efficiency, Java maintains a String constant pool with String objects that will eventually be referenced by multiple variables.

```
String otherName = "Hello";
```

- `name` and `otherame` will reference the same String object.

String Methods (1 of 2)

- Since `String` is a class, objects that are instances of it have methods.
- One of those methods is the `length` method.

```
String name = "Mary Ann";  
int stringSize = name.length();
```

- The statement `name.length()` runs the `length` method on the object pointed to by the `name` variable and returns an `int` result.
- Point to ponder #3.

What is the result here? 8 (including spaces!)

String Methods (2 of 2)

- Other important methods

- `charAt(index)`: The argument `index` is an `int` value and specifies a character position in the string. For instance:

```
char letter;  
String name = "Herman";  
letter = name.charAt(3);  
System.out.println(letter);
```

'm'. The first character is at position 0.

- `toLowerCase()`: returns a new string that is the lowercase equivalent of the string contained in the calling object.

```
String bigName = "HERMAN";  
String littleName = bigName.toLowerCase();  
System.out.println(littleName);
```

"herman"

- `toUpperCase()`: returns a new string that is the uppercase equivalent of the string contained in the calling object.

```
String littleName = "herman";  
String bigName = littleName.toUpperCase();  
System.out.println(bigName);
```

"HERMAN"

See example: `StringMethods.java`

Scope (1 of 3)

- *Scope* refers to the part of a program that has access to a variable's contents.
- Variables declared inside a method (like the main method) are called *local variables*.
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

Scope (2 of 3)

See example: `Scope.java`

```
1 // This program can't find its variable.
2
3 public class Scope
4 {
5     public static void main(String[] args)
6     {
7         System.out.println(value); // ERROR!
8         int value = 100;
9     }
10 }
```

**Tried to use a variable before
the variable is declared.**

Scope (3 of 3)

See example: `Scope.java`

```
public static void main(String[] args)
{
    // Declare a variable named number and
    // display its value.
    int number = 7;
    System.out.println(number);
    // Declare another variable named number and
    // display its value.
    int number = 100; // ERROR!!!
    System.out.println(number); // ERROR!!!
}
```

Tried to create two local variables with the same name in the same scope.

Commenting Code (1 of 4)

- Java provides three methods for commenting code.

Comment Style	Description
//	Single line comment. Anything after the // on the line will be ignored by the compiler.
/* ... */	Block comment (Multi-Line Comments). Everything beginning with /* and ending with the first */ will be ignored by the compiler.
/** ... */	Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler.

Commenting Code (2 of 4)

- Javadoc comments can be built into HTML documentation.
- To create the documentation:
 - Run the `javadoc` program with the source file as an argument
 - Ex: `javadoc Comment3.java`
- The `javadoc` program will create `index.html` and several other documentation files in the same directory as the input file.

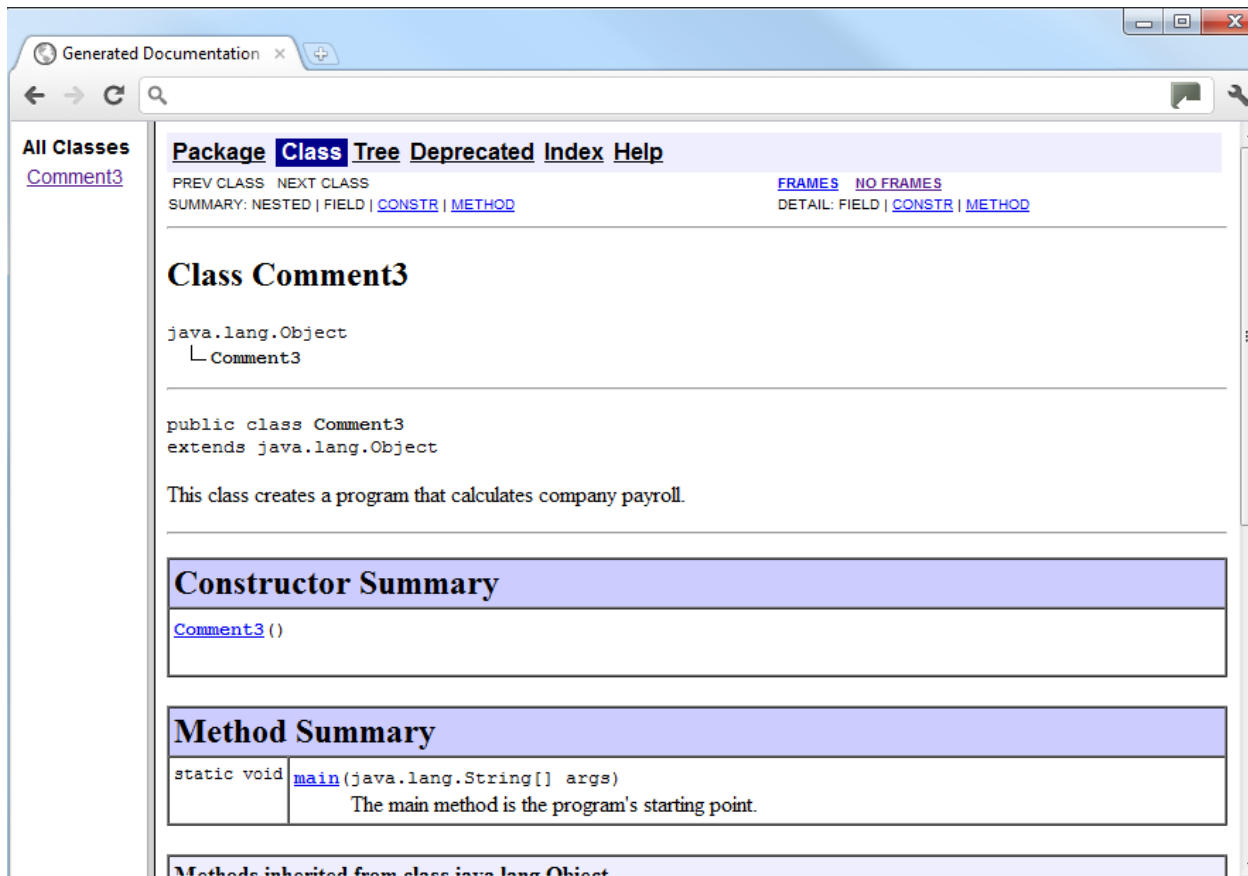
Commenting Code (3 of 4)

- See example: `Comment3.java`

```
1 /**
2  This class creates a program that calculates company payroll.
3  */
4
5  public class Comment3
6  {
7  /**
8   The main method is the program's starting point.
9   */
10
11     public static void main(String[] args)
12     {
13         double payRate; // Holds the hourly pay rate
14         double hours; // Holds the hours worked
15         int employeeNumber; // Holds the employee number
16
17     // The Remainder of This Program is Omitted.
18     }
19 }
```

Commenting Code (4 of 4)

- Example [index.html](#):



Programming Style (1 of 3)

- The way a programmer uses spaces, indentations, blank lines, and punctuation characters to visually arrange source code.
- Java compiler doesn't care that each statement is on a separate line, or that spaces separate operators from operands. Humans, care!

Programming Style (2 of 3)

See example: `Compact.java`

```
1 public class Compact {public static void main(String []  
args){int  
2 shares=220; double averagePrice=14.67;  
System.out.println(  
3 "There were "+shares+" shares sold at $" +averagePrice+  
4 " per share.");}}
```

- The program output: There were 220 shares sold at \$14.67 per share.

Programming Style (3 of 3)

See example: `Readable.java`

```
1 /**
2  This example is much more readable than Compact.java.
3  */
4
5 public class Readable
6 {
7     public static void main(String[] args)
8     {
9         int shares = 220;
10        double averagePrice = 14.67;
11
12        System.out.println("There were " + shares +
13                            " shares sold at $" + averagePrice + " per share.");
14    }
15 }
```

- The program output: There were 220 shares sold at \$14.67 per share.

Indentation

- Programs should use proper indentation.
- Each block of code should be indented a few spaces from its surrounding block.
- Two to four spaces are sufficient.

```
/**
    This example is much more readable than Compact.java.
 */

public class Readable
{
    INDENT public static void main(String[] args)
    INDENT {
    INDENT INDENT int shares = 220;
    INDENT INDENT double averagePrice = 14.67;

    INDENT INDENT System.out.println("There were " + shares +
    INDENT INDENT Extra spaces inserted here " shares sold at $" +
    INDENT INDENT Extra spaces inserted here averagePrice + " per share.");
    INDENT }
}
```

Dialog Boxes

- A *dialog box* is a small graphical window that displays a message to the user or requests input.
- A variety of dialog boxes can be displayed using the `JOptionPane` class.
- Two of the dialog boxes are:
 - Message Dialog - a dialog box that displays a message.
 - Input Dialog - a dialog box that prompts the user for input.
- `JOptionPane` class must be imported:

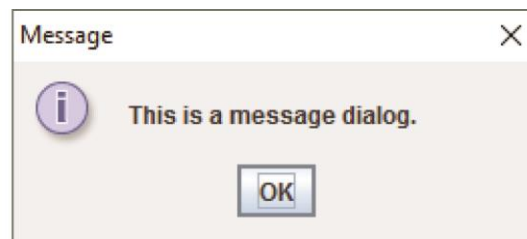
```
import javax.swing.JOptionPane;
```

The JOptionPane

The JOptionPane class provides methods to display each type of dialog box.

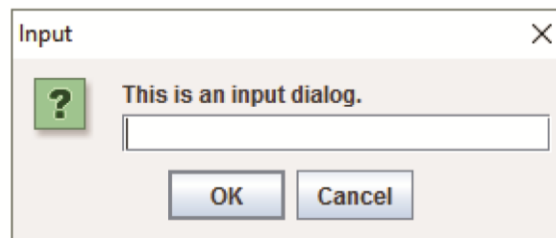
```
JOptionPane.showMessageDialog(null, "Hello World");
```

Message dialog



```
String name = JOptionPane.showInputDialog("Enter your name.");
```

Input dialog



The Parse Methods (1 of 2)

- A `String` containing a number, such as “127.89”, can be converted to a numeric data type.
- Each of the numeric wrapper classes, (covered in Chapter 10) has a method that converts a `String` to a number.
 - The `Integer` class has a method that converts a `String` to an `int`,
 - The `Double` class has a method that converts a `String` to a `double`, and
 - etc.
- These methods are known as *parse methods* because their names begin with the word “parse.”

The Parse Methods (2 of 2)

```
// Store 1 in bVar.  
byte bVar = Byte.parseByte("1");  
  
// Store 2599 in iVar.  
int iVar = Integer.parseInt("2599");  
  
// Store 10 in sVar.  
short sVar = Short.parseShort("10");  
  
// Store 15908 in lVar.  
long lVar = Long.parseLong("15908");  
  
// Store 12.3 in fVar.  
float fVar = Float.parseFloat("12.3");  
  
// Store 7945.6 in dVar.  
double dVar = Double.parseDouble("7945.6");
```