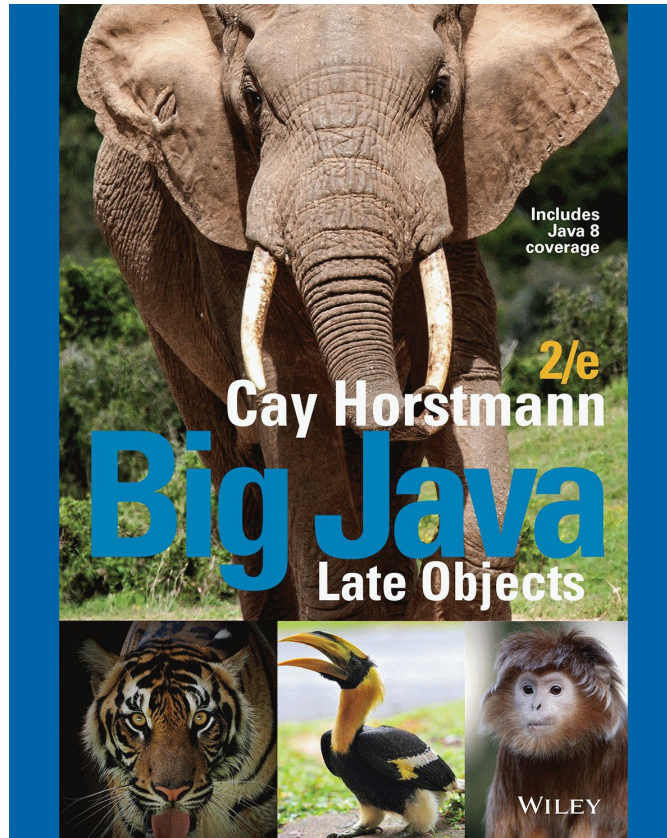


Chapter 2 - Fundamental Data Types



Chapter Goals



FLIGHT	DESTINATION	GATE #
742	LOS ANGELES	A23
801	LONDON	C72
485	MADRID	B34
770	PARIS	A14
54	TOKYO	C83
753	HONG KONG	G1
14	MIAMI	C5
18	NEW YORK	D
4	RIO DEJANEIRO	A
34	SYDNEY	
5	BANGKOK	

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read and process inputs, and display the results
- To learn how to use the Java String type

Variables

- Most computer programs hold temporary values in **named storage locations**
 - Programmers name them for easy access
- There are many different types (sizes) of storage to hold different things
- You '**declare**' a variable by telling the compiler:
 - What **type (size)** of variable you need
 - What **name** you will use to refer to it

Syntax 2.1 Variable Declaration

- When declaring a variable, you often specify an initial value
- This is also where you tell the compiler the size (type) it will hold

Syntax *typeName variableName = value;*
or
typeName variableName;


Types introduced in this chapter are the number types **int** and **double** (see Table 2) and the String type (see Section 2.5).

int cansPerPack = 6;

See Table 3 for rules and examples of valid names.

A variable declaration ends with a semicolon.

Supplying an initial value is optional, but it is usually a good idea. See Common Error 2.1.

 Use a descriptive variable name. See Programming Tip 2.1.



An Example: Soda Deal

- Soft drinks are sold in cans and bottles. A store offers a six-pack of 12-ounce cans for the same price as a two-liter bottle. Which should you buy? (12 fluid ounces equal approximately 0.355 liters.)
- **List of variables:**

Number of cans per pack	Whole number
Ounces per can	Whole number
Ounces per bottle	Number with fraction



Variables and Contents

- Each variable has an identifier (name) and contents
- You can (optionally) set the contents of a variable when you declare it



```
int cansPerPack = 6;
```

- Imagine a parking space in a parking garage
 - Identifier: J053
 - Contents: Bob's Chevy



Example Declarations

Table 1 Variable Declarations in Java

Variable Name	Comment
<code>int cans = 6;</code>	Declares an integer variable and initializes it with 6.
<code>int total = cans + bottles;</code>	The initial value need not be a fixed value. (Of course, <code>cans</code> and <code>bottles</code> must have been previously declared.)
 <code>bottles = 1;</code>	Error: The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.1.4.
 <code>int volume = "2";</code>	Error: You cannot initialize a number with a string.
<code>int cansPerPack;</code>	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
<code>int dollars, cents;</code>	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

Why Different Types?

- There are three different types of variables that we will use in this chapter:

1) A whole number (no fractional part) `int`

2) A number with a fraction part `double`

3) A word (a group of characters) `String`

- Specify the type before the name in the declaration

```
int cansPerPack = 6;
```

```
double canVolume = 12.0;
```


Why Different Variables?

- Back to the garage analogy, parking spaces may be different sizes for different types of vehicles
 - Bicycle
 - Motorcycle
 - Full Size
 - Electric Vehicle





Number Literals in Java

- Sometimes when you just type a number, the compiler has to 'guess' what type it is

```
amt = 6 * 12.0;  
PI = 3.14;  
canVol = 0.335;
```

Table 2 Number Literals in Java

Number	Type	Comment
6	int	An integer has no fractional part.
-6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type double.
1.0	double	An integer with a fractional part .0 has type double.
1E6	double	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type double.
2.96E-2	double	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
 100,000		Error: Do not use a comma as a decimal separator.
 3 1/2		Error: Do not use fractions; use decimal notation: 3.5

Floating-Point Numbers

- Java stores numbers with fractional parts as 'floating point' numbers.
- They are stored in four parts
 - Sign
 - Mantissa
 - Radix
 - Exponent
- A 'double' is a double-precision floating point number: It takes twice the storage (52 bit mantissa) as the smaller 'float' (23 bit mantissa)

Parts of a floating point number -5:






Sign	Mantissa	Radix ^{exponent}
-1	5	10 ⁰

Naming Variables

- Name should describe the purpose
 - `'canVolume'` is better than `'cv'`
- Use These Simple Rules
 - 1) Variable names must start with a letter or the underscore (`_`) character
 - Continue with letters (upper or lower case), digits or the underscore
 - 2) You cannot use other symbols (`?` or `%...`) and spaces are not permitted
 - 3) Separate words with `'camelHump'` notation
 - Use upper case letters to signify word boundaries
 - 4) Don't use reserved `'Java'` words (see Appendix C)

Variable Names in Java

Table 3 Variable Names in Java

Variable Name	Comment
canVolume1	Variable names consist of letters, numbers, and the underscore character.
x	In mathematics, you use short variable names such as x or y . This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1 on page 38).
 CanVolume	Caution: Variable names are case sensitive. This variable name is different from canVolume, and it violates the convention that variable names should start with a lowercase letter.
 6pack	Error: Variable names cannot start with a number.
 can volume	Error: Variable names cannot contain spaces.
 double	Error: You cannot use a reserved word as a variable name.
 1tr/fl.oz	Error: You cannot use symbols such as / or .

The Assignment Statement

- Use the 'assignment statement' (with an '=') to place a new value into a variable

```
int cansPerPack = 6;    // declare & initialize  
cansPerPack = 8;    // assignment
```

- Beware: The = sign is **NOT** used for comparison:
 - It copies the value on the right side into the variable on the left side
 - You will learn about the comparison operator in the next chapter

Assignment Syntax

- The value on the right of the '=' sign is copied to the variable on the left

This is an initialization
of a new variable,
NOT an assignment.

```
double total = 0;
```

This is an assignment.

```
.  
.  
total = bottles * BOTTLE_VOLUME;
```

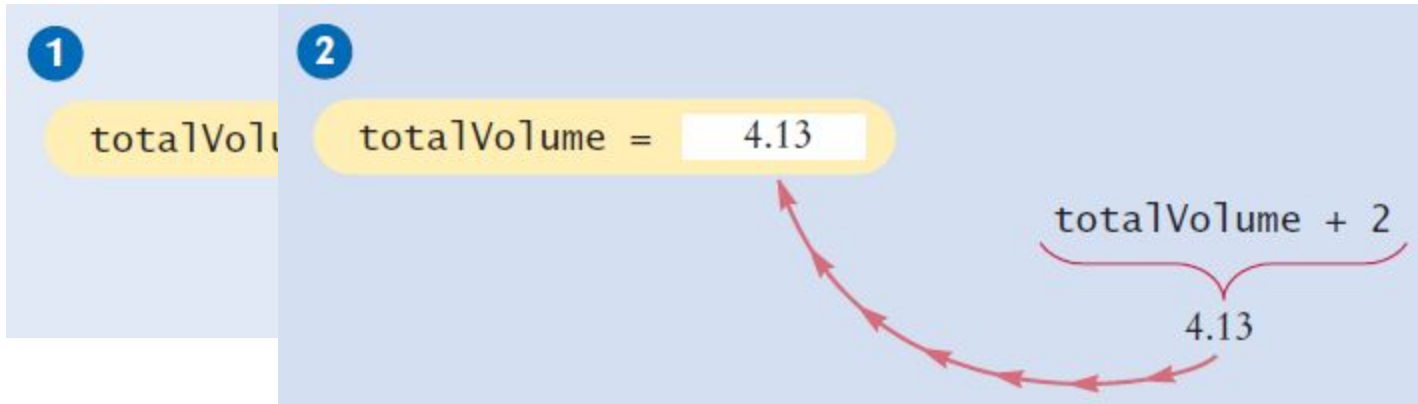
The name of a previously
defined variable

The expression that replaces the previous value

```
.  
.  
.  
total = total + cans * CAN_VOLUME;
```

The same name
can occur on both sides.
See Figure 1.

Updating a Variable



▪Step by Step:

```
totalVolume = totalVolume + 2;
```

1. Calculate the right hand side of the assignment; Find the value of `totalVolume`, and add 2 to it
2. Store the result in the variable named on the left side of the assignment operator (`totalVolume` in this case)

Declarations vs. Assignments

- Variable declarations and an assignment statements are different

`int cansPerPack = 6;` Declaration

...

`cansPerPack = 8;` Assignment statement

- Declarations define a new variable and can give it an initial value
- Assignments modify the value of an existing variable

Constants

- When a variable is defined with the reserved word `final`, its value can never be changed

```
final double BOTTLE_VOLUME = 2;
```

- It is good style to use named constants to explain numerical values to be used in calculations
 - Which is clearer?

```
double totalVolume = bottles * 2;  
double totalVolume = bottles *  
    BOTTLE_VOLUME;
```

- A programmer reading the first statement may not understand the significance of the 2
- Also, if the constant is used in multiple places and needs to be changed, only the initialization changes


Constant Declaration

The `final` reserved word indicates that this value cannot be modified.

```
final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
```



Use uppercase letters for constants.



This comment explains how the value for the constant was determined.

- It is customary (not required) to use all UPPER_CASE letters for constants

Java Comments

- There are three forms of comments:

1: `// single line (or rest of line to right)`

2: `/*`

`multi-line - all comment until matching`

`*/`

3: `/**`

`multi-line Javadoc comments`

`*/`

- Use comments at the beginning of each program, and to clarify details of the code
- Use comments to add explanations for humans who read
- your code
- The compiler ignores comments

Java Comment Example

```
1  /**
2   This program computes the volume (in liters) of a six-pack of soda
3   cans and the total volume of a six-pack and a two-liter bottle.
4   */
5  public class Volume1
6  {
7      public static void main(String[] args)
8      {
9          int cansPerPack = 6;
10         final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
11         double totalVolume = cansPerPack * CAN_VOLUME;
12
13         System.out.print("A six-pack of 12-ounce cans contains ");
14         System.out.print(totalVolume);
15         System.out.println(" liters.");
16
17         final double BOTTLE_VOLUME = 2; // Two-liter bottle
18     }
```

- Lines 1 - 4 are Javadoc comments for the class Volume1
- Lines 10 and 17 use single-line comment to clarify the unit of measurement

Self Check 2.1

Declare a variable suitable for holding the number of bottles in a case.

Answer: One possible answer is

```
int bottlesPerCase = 8;
```

You may choose a different variable name or a different initialization value, but your variable should have type `int`.

Self Check 2.2

What is wrong with the following variable declaration?

```
int ounces per liter = 28.35
```

Answer: There are three errors:

- You cannot have spaces in variable names.
- The variable type should be double because it holds a fractional value.
- There is a semicolon missing at the end of the statement.

Self Check 2.3

Declare and initialize two variables, `unitPrice` and `quantity`, to contain the unit price of a single bottle and the number of bottles purchased. Use reasonable initial values.

Answer:

```
double unitPrice = 1.95;  
int quantity = 2;
```


Self Check 2.4

Use the variables declared in Self Check 3 to display the total purchase price.

Answer:

```
System.out.print("Total price: ");  
System.out.println(unitPrice * quantity);
```

Self Check 2.5

Some drinks are sold in four-packs instead of six-packs. How would you change the `Volume1.java` program to compute the total volume?

Answer: Change the declaration of `cansPerPack` to

```
int cansPerPack = 4;
```

Self Check 2.6

What is wrong with this comment?

```
double canVolume = 0.355; /* Liters in a 12-ounce can //
```

Answer: You need to use a `*/` delimiter to close a comment that begins with a `/*`:

```
double canVolume = 0.355;  
/* Liters in a 12-ounce can */
```

Self Check 2.7

Suppose the type of the `cansPerPack` variable in `Volume1.java` was changed from `int` to `double`. What would be the effect on the program?

Answer: The program would compile, and it would display the same result. However, a person reading the program might find it confusing that fractional cans are being considered.

Self Check 2.8

Why can't the variable `totalVolume` in the `Volume1.java` program be declared as `final`?

Answer: Its value is modified by the assignment statement.

Self Check 2.9

How would you explain assignment using the parking space analogy?

Answer: Assignment would occur when one car is replaced by another in the parking space.

Common Error 2.1

- Undeclared Variables

- You must declare a variable before you use it: (i.e. above in the code)

```
double canVolume = 12 * literPerOunce; // ??  
double literPerOunce = 0.0296;
```

- Uninitialized Variables

- You must initialize (i.e. set) a variable's contents before you use it

```
int bottles;  
int bottleVolume = bottles * 2;    // ??
```

Common Error 2.2

- Overflow means that storage for a variable cannot hold the result

```
int fiftyMillion = 50000000;  
System.out.println(100 * fiftyMillion);  
// Expected: 5000000000
```

- Will print out 705032704
- Why?
 - The result (5 billion) overflowed `int` capacity
 - Maximum value for an `int` is **+2,147,483,647**
- Use a `long` instead of an `int` (or a `double`)

Common Error 2.3

- Roundoff Errors

- Floating point values are not exact

- This is a limitations of binary values (no fractions):

```
double price = 4.35;
```

```
double quantity = 100;
```

```
double total = price * quantity;
```

```
// Should be 100 * 4.35 = 435.00
```

```
System.out.println(total); // Prints 434.99999999999999
```

- You can deal with roundoff errors by rounding to the nearest integer (see Section 2.2.5) or by displaying a fixed number of digits after the decimal separator (see Section 2.3.2).

All of the Java Numeric Types

Type	Description	
int	The integer type, with range $-2,147,483,648$ (<code>Integer.MIN_VALUE</code>) . . . $2,147,483,647$ (<code>Integer.MAX_VALUE</code> , about 2.14 billion)	Whole Numbers (no fractions)
byte	The type describing a byte consisting of 8 bits, with range -128 . . . 127	
short	The short integer type, with range $-32,768$. . . $32,767$	
long	The long integer type, with about 19 decimal digits	
double	The double-precision floating-point type, with about 15 decimal digits and a range of about $\pm 10^{308}$	Floating point Numbers
float	The single-precision floating-point type, with about 7 decimal digits and a range of about $\pm 10^{38}$	
char	The character type, representing code units in the Unicode encoding scheme (see Section 2.6.6)	Characters (no math)

- Each type has a range of values that it can hold

Value Ranges per Type

Integer Types

- `byte`: A very small number (-128 to +127)
- `short`: A small number (-32768 to +32767)
- `int`: A large number (-2,147,483,648 to +2,147,483,647)
- `long`: A huge number

• Floating Point Types





- `float`: A huge number with decimal places
- `double`: Much more precise, for heavy math

• Other Types



- `boolean`: `true` or `false`
- `char`: One symbol in single quotes 'a'

Storage per Type (in bytes)



Integer Types

- `byte:` 
- `short:` 
- `int:` 
- `long:` 

• Floating Point Types

- `float:` 
- `double:` 

• Other Types

- `boolean:` 
- `char:` 

Arithmetic

- Java supports all of the same basic math as a calculator:

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

- You write your expressions a bit differently though

Algebra

$$\frac{a + b}{2}$$

Java

$$(a + b) / 2$$



- Precedence is similar to Algebra:

- PEMDAS

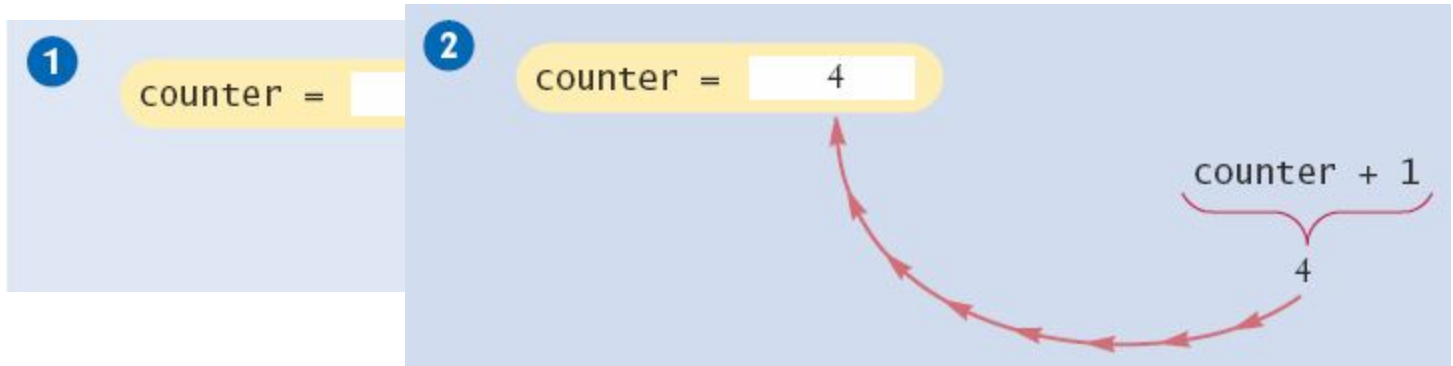
- Parenthesis, Exponent, Multiply/Divide, Add/Subtract

Mixing Numeric Types

- It is safe to convert a value from an integer type to a floating-point type
 - No 'precision' is lost
- But going the other way can be dangerous
 - All fractional information is lost
 - The fractional part is discarded (not rounded)
- If you mix types integer and floating-point types in an expression, no precision is lost:

```
double area, pi = 3.14;  
int radius = 3;  
area = radius * radius * pi;
```

Incrementing a Variable



▪Step by Step:

```
counter = counter + 1;
```

1. Do the right hand side of the assignment first:
Find the value stored in `counter`, and add 1 to it
2. Store the result in the variable named on the left side of the assignment operator (`counter` in this case)

Shorthand for Incrementing

- Incrementing (+1) and decrementing (-1) integer types is so common that there are shorthand version for each

Long Way	Shortcut
<code>counter = counter + 1;</code>	<code>counter++ ;</code>
<code>counter = counter - 1;</code>	<code>counter-- ;</code>

Integer Division and Remainder

- When both parts of division are integers, the result is an integer.

- All fractional information is lost (no rounding)

```
int result = 7 / 4;
```

- The value of result will be 1

- If you are interested in the remainder of dividing two integers, use the % operator (called modulus):

```
int remainder = 7 % 4;
```

- The value of remainder will be 3
 - Sometimes called modulo divide

Powers and Roots

- In Java, there are no symbols for power and roots

$$b \times \left(1 + \frac{r}{100}\right)^n$$

Becomes:

`b * Math.pow(1 + r / 100, n)`

`b * Math.pow(1 + r / 100, n)`

$$\frac{r}{100}$$

$$1 + \frac{r}{100}$$

$$\left(1 + \frac{r}{100}\right)^n$$

$$b \times \left(1 + \frac{r}{100}\right)^n$$

- Analyzing the expression:

- The Java library declares many mathematical functions, such as `Math.sqrt` (square root) and `Math.pow` (raising to a power)

Mathematical Methods

Table 6 Mathematical Methods

Method	Returns
<code>Math.sqrt(x)</code>	Square root of x (≥ 0)
<code>Math.pow(x, y)</code>	x^y ($x > 0$, or $x = 0$ and $y > 0$, or $x < 0$ and y is an integer)
<code>Math.sin(x)</code>	Sine of x (x in radians)
<code>Math.cos(x)</code>	Cosine of x
<code>Math.tan(x)</code>	Tangent of x
<code>Math.toRadians(x)</code>	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
<code>Math.toDegrees(x)</code>	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
<code>Math.exp(x)</code>	e^x
<code>Math.log(x)</code>	Natural log ($\ln(x)$, $x > 0$)
<code>Math.log10(x)</code>	Decimal log ($\log_{10}(x)$, $x > 0$)
<code>Math.round(x)</code>	Closest integer to x (as a long)
<code>Math.abs(x)</code>	Absolute value $ x $
<code>Math.max(x, y)</code>	The larger of x and y
<code>Math.min(x, y)</code>	The smaller of x and y

Floating-Point to Integer Conversion

- The Java compiler does not allow direct assignment of a floating-point value to an integer variable

```
double balance = total + tax;  
int dollars = balance; // Error
```

- You can use the 'cast' operator: `(int)` to force the conversion:

```
double balance = total + tax;  
int dollars = (int) balance; // no Error
```

- You lose the fractional part of the floating-point value (no rounding occurs)

Cast Syntax

This is the type of the expression after casting.

`(int) (balance * 100)`

These parentheses are a part of the cast operator.

Use parentheses here if the cast is applied to an expression with arithmetic operators.

- Casting is a very powerful tool and should be used carefully
- To round a floating-point number to the nearest whole number, use the `Math.round` method
- This method returns a long integer, because large floating-point numbers cannot be stored in an `int`

```
long rounded = Math.round(balance);
```

Arithmetic Expressions

Mathematical Expression	Java Expression	Comments
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	The parentheses are required; <code>x + y / 2</code> computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	<code>x * y / 2</code>	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	<code>Math.pow(1 + r / 100, n)</code>	Use <code>Math.pow(x, n)</code> to compute x^n .
$\sqrt{a^2 + b^2}$	<code>Math.sqrt(a * a + b * b)</code>	<code>a * a</code> is simpler than <code>Math.pow(a, 2)</code> .
$\frac{i + j + k}{3}$	<code>(i + j + k) / 3.0</code>	If i , j , and k are integers, using a denominator of 3.0 forces floating-point division.
π	<code>Math.PI</code>	<code>Math.PI</code> is a constant declared in the <code>Math</code> class.

Self Check 2.10

A bank account earns interest once per year. In Java, how do you compute the interest earned in the first year? Assume variables `percent` and `balance` of type `double` have already been declared.

Answer: `double interest = balance * percent / 100;`

Self Check 2.11

In Java, how do you compute the side length of a square whose area is stored in the variable `area`?

Answer: `double sideLength = Math.sqrt(area);`

Self Check 2.12

The volume of a sphere is given by

$$V = \frac{4}{3}\pi r^3$$

If the radius is given by a variable `radius` of type `double`, write a Java expression for the volume.

Answer: `4 * Math.PI * Math.pow(radius, 3) / 3` or `(4.0 / 3) * Math.PI * Math.pow(radius, 3)`, but not `(4 / 3) * Math.PI * Math.pow(radius, 3)`

Self Check 2.13

What is the value of $1729 / 10$ and $1729 \% 10$?

Answer: 172 and 9

Self Check 2.14

If n is a positive integer, what is $(n / 10) \% 10$?

Answer: It is the second-to-last digit of n . For example, if n is 1729, then $n / 10$ is 172, and $(n / 10) \% 10$ is 2.

Common Error 2.4

- Unintended Integer Division

```
System.out.print("Please enter your last three test  
scores: ");  
int s1 = in.nextInt();  
int s2 = in.nextInt();  
int s3 = in.nextInt();  
double average = (s1 + s2 + s3) / 3; // Error
```

- Why?

- All of the calculation on the right happens first

- Since all are `ints`, the compiler uses integer division

- Then the result (an `int`) is assigned to the `double`

- There is no fractional part of the `int` result, so zero (.0) is assigned to the fractional part of the `double`

Common Error 2.5

- Unbalanced Parenthesis

- Which is correct?

$(-(b * b - 4 * a * c) / (2 * a) // 3 (, 2)$

$-(b * b - (4 * a * c)) / (2 * a) // 3 (, 3)$

- The count of (and) must match

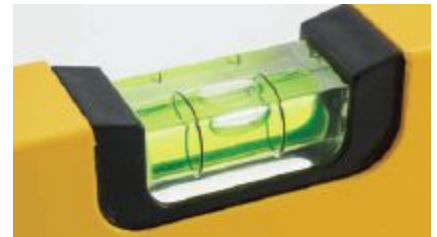
- Unfortunately, it is hard for humans to keep track

- Here's a handy trick

- Count (as +1, and) as -1: Goal: 0

$-(b * b - (4 * a * c))) / 2 * a)$

1 2 1 0 -1 -2



Input and Output

- Reading Input
- You might need to ask for input (aka prompt for input) and then save what was entered
 - We will be reading input from the keyboard
 - For now, don't worry about the details
- This is a three step process in Java
 1. Import the `Scanner` class from its 'package'

```
java.util import java.util.Scanner;
```
 2. Setup an object of the `Scanner` class

```
Scanner in = new Scanner(System.in);
```
 3. Use methods of the new `Scanner` object to get input

```
int bottles = in.nextInt();  
double price = in.nextDouble();
```

Syntax 2.3: Input Statement

- The `Scanner` class allows you to read keyboard input from the user
 - It is part of the Java API `util` package
- Java classes are grouped into packages. Use the `import` statement to use classes from packages

Include this line so you can use the `Scanner` class.

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

Create a `Scanner` object to read keyboard input.

```
.  
.br/>Scanner in = new Scanner(System.in);  
.br/>.
```

Don't use `println` here.

Display a prompt in the console window.

```
System.out.print("Please enter the number of bottles: ");
```

Define a variable to hold the input value.

```
int bottles = in.nextInt();
```

The program waits for user input, then places the input into the variable.

Formatted Output

- Outputting floating point values can look strange:

Price per liter: 1.21997

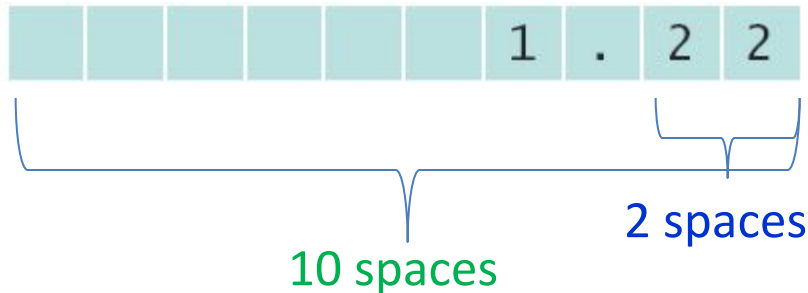
- To control the output appearance of numeric variables, use formatted output tools such as:

```
System.out.printf("%.2f", price);
```

Price per liter: 1.22

```
System.out.printf("%10.2f", price);
```

Price per liter: 1.22



- The `%10.2f` is called a format specifier

Format Types

- Formatting is handy to align columns of output

Table 8 Format Specifier Examples		
Format String	Sample Output	Comments
"%d"	24	Use d with an integer.
"%5d"	24	Spaces are added so that the field width is 5.
"Quantity:%5d"	Quantity: 24	Characters inside a format string but outside a format specifier appear in the output.
"%f"	1.21997	Use f with a floating-point number.
"%.2f"	1.22	Prints two digits after the decimal point.
"%7.2f"	1.22	Spaces are added so that the field width is 7.
"%s"	Hello	Use s with a string.
"%d %.2f"	24 1.22	You can format multiple values at once.
"Hello%nWorld%n"	Hello World	Each %n causes subsequent output to continue on a new line.

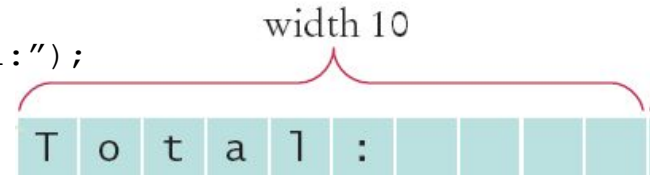
- You can also include text inside the quotes:

```
System.out.printf("Price per liter: %10.2f", price);
```

Formatted Output Examples

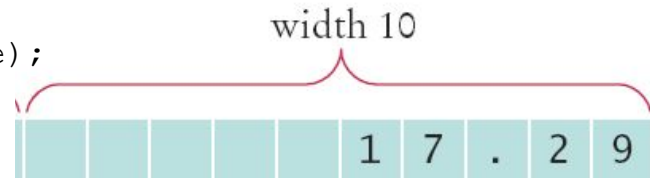
- Left Justify a String:

```
System.out.printf("%-10s", "Total:");
```



- Right justify a number with two decimal places

```
System.out.printf("%10.2f", price);
```

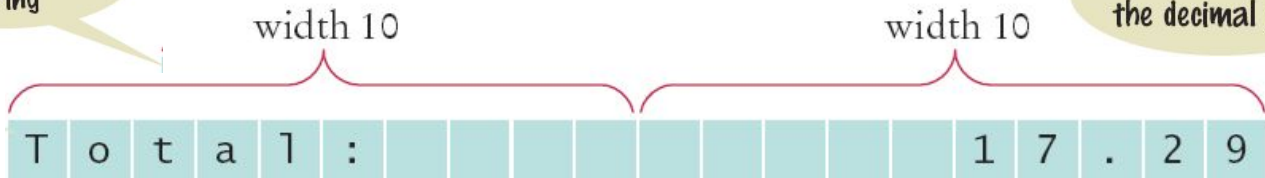


- And you can print multiple values

```
System.out.printf("%-10s%10.2f", "Total:", price);
```

A left-justified
string

Two digits after
the decimal point



Volume2.java

section_3/Volume2.java

```
1  import java.util.Scanner;
2
3  /**
4   * This program prints the price per ounce for a six-pack of cans.
5   */
6  public class Volume2
7  {
8      public static void main(String[] args)
9      {
10         // Read price per pack
11
12         final double CANS_PER_PACK = 6;
13         double packVolume = canVolume * CANS_PER_PACK;
14
15         // Compute and print price per ounce
16
17         double pricePerOunce = packPrice / packVolume;
18
19         System.out.printf("Price per ounce: %8.2f", pricePerOunce);
20         System.out.println();
21     }
22 }
```

Self Check 2.15

Write statements to prompt for and read the user's age using a `Scanner` variable named `in`.

Answer:

```
System.out.print("How old are you? ");  
int age = in.nextInt();
```

Self Check 2.16

What is wrong with the following statement sequence?

```
System.out.print("Please enter the unit price: ");  
double unitPrice = in.nextDouble();  
int quantity = in.nextInt();
```

Answer: There is no prompt that alerts the program user to enter the quantity.

Self Check 2.17

What is problematic about the following statement sequence?

```
System.out.print("Please enter the unit price: ");  
double unitPrice = in.nextInt();
```

Answer: The second statement calls `nextInt`, not `nextDouble`. If the user were to enter a price such as `1.95`, the program would be terminated with an “input mismatch exception”.

Self Check 2.18

What is problematic about the following statement sequence?

```
System.out.print("Please enter the number of cans");  
int cans = in.nextInt();
```

Answer: There is no colon and space at the end of the prompt. A dialog would look like this:

Please enter the number of cans6

Self Check 2.19

What is the output of the following statement sequence?

```
int volume = 10;  
System.out.printf("The volume is %5d", volume);
```

Answer:

The total volume is 10

There are four spaces between `is` and `10`. One space originates from the format string (the space between `s` and `%`), and three spaces are added before `10` to achieve a field width of 5.

Self Check 2.20

Using the `printf` method, print the values of the integer variables `bottles` and `cans` so that the output looks like this:

```
Bottles: 8
```

```
Cans: 24
```

The numbers to the right should line up. (You may assume that the numbers have at most 8 digits.)

Answer: Here is a simple solution:

```
System.out.printf("Bottles: %8d%n", bottles);
```

```
System.out.printf("Cans: %8d%n", cans);
```

Note the spaces after `Cans:`. Alternatively, you can use format specifiers for the strings. You can even combine all output into a single statement:

```
System.out.printf("%-9s%8d%n%-9s%8d%n", "Bottles: ",  
bottles, "Cans:", cans);
```

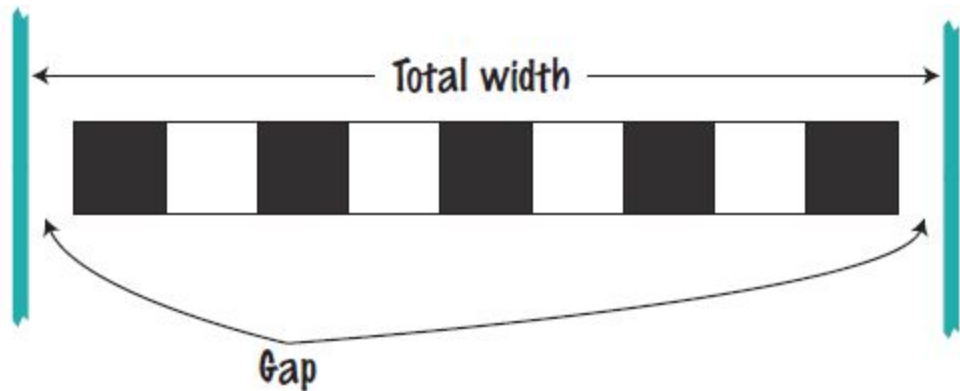
Tip 2.2 Java API Documentation

- Lists the classes and methods of the Java AP
 - On the web at: <http://download.oracle.com/javase/6/docs/api>

The screenshot shows the Java Platform SE 7 API documentation in Mozilla Firefox. The browser window title is "Scanner (Java Platform SE 7) - Mozilla Firefox". The address bar shows the URL <http://download.oracle.com/javase/7/docs/api/>. The page has a navigation bar with tabs: Overview, Package, Class (selected), Use, Tree, Deprecated, Index, and Help. The left sidebar lists various packages, including `javax.net`, `javax.net.ssl`, `javax.print`, `javax.print.attribute`, `javax.print.attribute.st`, `javax.print.event`, `javax.rmi`, `javax.rmi.CORBA`, `javax.rmi.ssl`, `javax.script`, and `javax.security.auth`. A yellow arrow labeled "Packages" points to the `javax` package in the sidebar. The main content area shows the `Class Scanner` page. It includes the package `java.util`, the class `Scanner`, and a list of implemented interfaces: `Closeable`, `AutoCloseable`, and `Iterator<String>`. A yellow arrow labeled "Classes" points to the `Scanner` class in the sidebar. The `Scanner` class is defined as `public final class Scanner extends Object implements Iterator<String>, Closeable`. A yellow arrow labeled "Methods" points to the `Scanner` class definition. The description of the class is: "A simple text scanner which can parse primitive types and strings using regular expressions."

Problem Solving: First By Hand

- A very important step for developing an algorithm is to first carry out the computations *by hand*. Example Problem:
 - A row of black and white tiles needs to be placed along a wall. For aesthetic reasons, the architect has specified that the first and last tile shall be black.
 - Your task is to compute the number of tiles needed and the gap at each end, given the space available and the width of each tile.



Start with Example Values

- Givens:

- Total width: 100 inches

- Tile width: 5 inches

- Test your values

- Let's see... $100/5 = 20$, perfect! 20 tiles. No gap.

- But wait... BW...BW "...first and last tile shall be black."

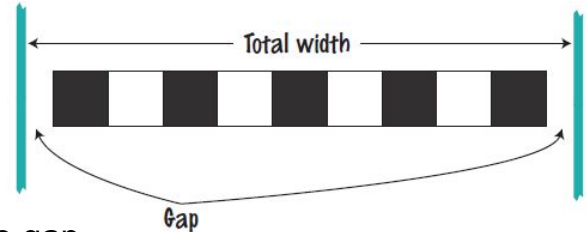
- Look more carefully at the problem...

- Start with one black, then some number of WB pairs



- Observation: each pair is 2x width of 1 tile

- In our example, $2 \times 5 = 10$ inches

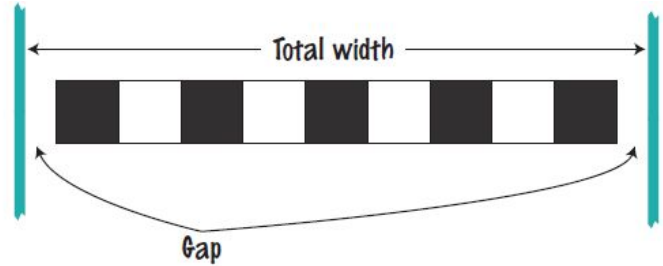


Start with Example Values

Total width: 100 inches

Tile width: 5 inches

- Calculate total width of all tiles
 - One black tile: 5"
 - 9 pairs of BWs: 90"
 - Total tile width: 95"
- Calculate gaps (one on each end)
 - $100 - 95 = 5$ " total gap
 - 5 " gap / 2 = 2.5" at each end



Now Devise an Algorithm

- Use your example to see how you calculated values
- How many pairs?
 - Note: must be a whole number

Integer part of:

$$(\text{total width} - \text{tile width}) / 2 \times \text{tile width}$$

- How many tiles?

$$1 + 2 \times \text{the number of pairs}$$

- Gap at each end

$$(\text{total width} - \text{number of tiles} \times \text{tile width}) / 2$$

Self Check 2.21

Translate the pseudocode for computing the number of tiles and the gap width into Java.

Answer:

```
int pairs = (totalWidth - tileWidth) / (2 * tileWidth);  
int tiles = 1 + 2 * pairs;  
double gap = (totalWidth - tiles * tileWidth) / 2.0;
```

Be sure that `pairs` is declared as an `int`.

Self Check 2.22

Suppose the architect specifies a pattern with black, gray, and white tiles, like this:



Again, the first and last tile should be black. How do you need to modify the algorithm?

Answer: Now there are groups of four tiles (gray/ white/gray/black) following the initial black tile. Therefore, the algorithm is now

number of groups = integer part of $(\text{total width} - \text{tile width}) / (4 \times \text{tile width})$

number of tiles = $1 + 4 \times \text{number of groups}$

The formula for the gap is not changed.

Self Check 2.23

A robot needs to tile a floor with alternating black and white tiles. Develop an algorithm that yields the color (0 for black, 1 for white), given the row and column number. Start with specific values for the row and column, and then generalize.

	1	2	3	4
1	Black	White	Black	White
2	White	Black	White	Black
3	Black	White	Black	
4	White	Black		

Answer: Clearly, the answer depends only on whether the row and column numbers are even or odd, so let's first take the remainder after dividing by 2. Then we can enumerate all expected answers:

Row %2	Column %2	Color
0	0	0
0	1	1
1	0	1
1	1	0

In the first three entries of the table, the color is simply the sum of the remainders. In the fourth entry, the sum would be 2, but we want a zero. We can achieve that by taking another remainder operation:

$$\text{color} = ((\text{row} \% 2) + (\text{column} \% 2)) \% 2$$

Self Check 2.24

For a particular car, repair and maintenance costs in year 1 are estimated at \$100; in year 10, at \$1,500. Assuming that the repair cost increases by the same amount every year, develop pseudocode to compute the repair cost in year 3 and then generalize to year n .

Answer: In nine years, the repair costs increased by \$1,400. Therefore, the increase per year is $\$1,400 / 9 \approx \156 . The repair cost in year 3 would be $\$100 + 2 \times \$156 = \$412$. The repair cost in year n is $\$100 + n \times \156 . To avoid accumulation of roundoff errors, it is actually a good idea to use the original expression that yielded \$156, that is,

$$\text{Repair cost in year } n = 100 + n \times 1400 / 9$$

Self Check 2.25

The shape of a bottle is approximated by two cylinders of radius r_1 and r_2 and heights h_1 and h_2 , joined by a cone section of height h_3 . Using the formulas for the volume of a cylinder, $V = \pi r^2 h$, and a cone section,

$$V = \pi \frac{(r_1^2 + r_1 r_2 + r_2^2) h}{3},$$

develop pseudocode to compute the volume of the bottle. Using an actual bottle with known volume as a sample, make a hand calculation of your pseudocode.

Answer: The pseudocode follows easily from the equations:

bottom volume = $\pi \times r_1^2 \times h_1$

top volume = $\pi \times r_2^2 \times h_2$

middle volume = $\pi \times (r_1^2 + r_1 \times r_2 + r_2^2) \times h_3 / 3$

total volume = bottom volume + top volume + middle volume

Measuring a typical wine bottle yields $r_1 = 3.6$, $r_2 = 1.2$, $h_1 = 15$, $h_2 = 7$, $h_3 = 6$ (all in centimeters). Therefore,

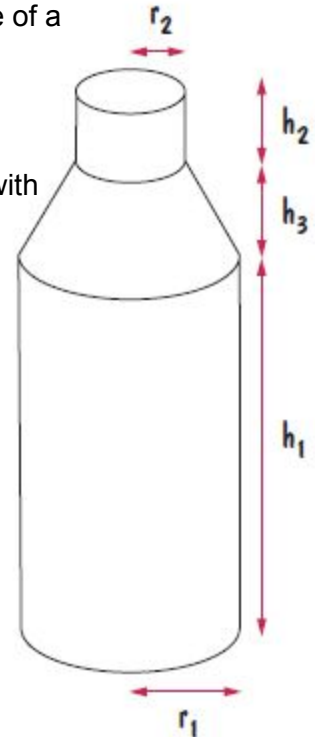
bottom volume = 610.73

top volume = 31.67

middle volume = 135.72

total volume = 778.12

The actual volume is 750 ml, which is close enough to our computation to give confidence that it is correct.



Strings

- The `String` Type:

- Type Variable Literal

- `String name = "Harry"`

- Once you have a `String` variable, you can use methods such as:

- `int n = name.length(); // n will be assigned 5`

- A `String`'s length is the number of characters inside:

- An empty `String` (length 0) is shown as ""

- The maximum length is quite large (an `int`)

String Concatenation (+)

- You can 'add' one `String` onto the end of another

```
String fName = "Harry"  
String lName = "Morgan"  
String name = fName + lName;    // HarryMorgan
```

- You wanted a space in between?

```
String name = fName + " " + lName;    // Harry Morgan
```

- To concatenate a numeric variable to a `String`:

```
String a = "Agent";  
int n = 7;  
String bond = a + n;    // Agent7
```

- Concatenate `Strings` and numerics inside `println`:

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

String Input

- You can read a `String` from the console with:

```
System.out.print("Please enter your name: ");  
String name = in.next();
```

- The `next` method reads one word at a time
- It looks for 'white space' delimiters

- You can read an entire line from the console with:

```
System.out.print("Please enter your address: ");  
String address = in.nextLine();
```

- The `nextLine` method reads until the user hits 'Enter'

- Converting a `String` variable to a number:

```
System.out.print("Please enter your age: ");  
String input = in.nextLine();  
int age = Integer.parseInt(input); // only digits!
```

String Escape Sequences

- How would you print a double quote?

- Preface the " with a \ inside the double quoted String

```
System.out.print("He said \"Hello\"");
```

- OK, then how do you print a backslash?

- Preface the \ with another \!

```
System.out.print("C:\\Temp\\Secret.txt");
```

- Special characters inside Strings

- Output a newline with a '\n'

```
System.out.print("*\n**\n***\n");
```

*

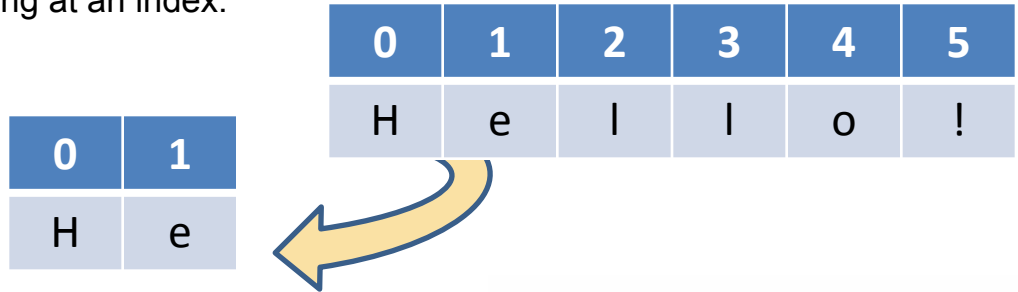
**

Strings and Characters

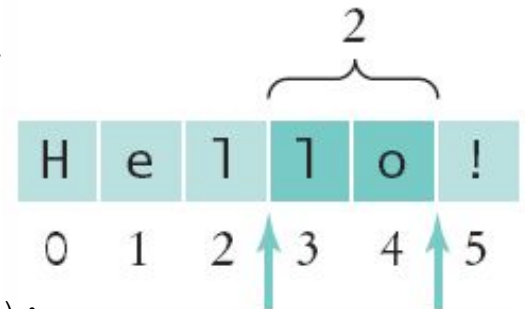
- Strings are sequences of characters
 - Unicode characters to be exact
 - Characters have their own type: `char`
 - Characters have numeric values
 - See the ASCII code chart in Appendix B
 - For example, the letter 'H' has a value of 72 if it were a number
- Use single quotes around a `char`
`char initial = 'B';`
- Use double quotes around a `String`
`String initials = "BRL";`

Copying a char from a String

- A substring is a portion of a `String`
- The `substring` method returns a portion of a `String` at a given index for a number of chars, starting at an index:



```
String greeting = "Hello!";  
String sub = greeting.substring(0, 2);
```



```
String sub2 = greeting.substring(3, 5);
```

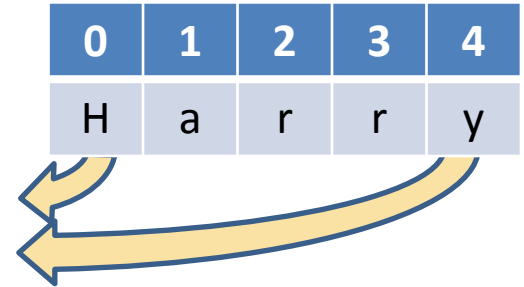
Copying a Portion of a String

- Each `char` inside a `String` has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- The first `char` is index zero (0)
- The `charAt` method returns a `char` at a given index inside a `String`:

```
String greeting = "Harry";  
char start = greeting.charAt(0);  
char last = greeting.charAt(4);
```



String Operations (1)

Table 9 String Operations

Statement	Result	Comment
<pre>string str = "Ja"; str = str + "Va";</pre>	str is set to "Java"	When applied to strings, + denotes concatenation.
<pre>System.out.println("Please" + " enter your name: ");</pre>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<pre>team = 49 + "ers"</pre>	team is set to "49ers"	Because "ers" is a string, 49 is converted to a string.
<pre>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</pre>	first contains "Harry" last contains "Morgan"	The next method places the next word into the string variable.
<pre>String greeting = "H & S"; int n = greeting.length();</pre>	n is set to 5	Each space counts as one character.
<pre>String str = "Sally"; char ch = str.charAt(1);</pre>	ch is set to 'a'	This is a char value, not a String. Note that the initial position is 0.

String Operations (2)

Statement	Result	Comment
<pre>String str = "Sally"; String str2 = str.substring(1, 4);</pre>	str2 is set to "all"	Extracts the substring starting at position 1 and ending before position 4.
<pre>String str = "Sally"; String str2 = str.substring(1);</pre>	str2 is set to "ally"	If you omit the end position, all characters from the position until the end of the string are included.
<pre>String str = "Sally"; String str2 = str.substring(1, 2);</pre>	str2 is set to "a"	Extracts a String of length 1; contrast with <code>str.charAt(1)</code> .
<pre>String last = str.substring(str.length() - 1);</pre>	last is set to the string containing the last character in str	The last character has position <code>str.length() - 1</code> .

Self Check 2.26

What is the length of the string "Java Program"?

Answer: The length is 12. The space counts as a character.

Self Check 2.27

Consider this string variable.

```
String str = "Java Program";
```

Give a call to the `substring` method that returns the substring "gram".

Answer: `str.substring(8, 12)` or `str.substring(8)`

Self Check 2.28

Use string concatenation to turn the string variable `str` from Self Check 27 into "Java Programming".

Answer: `str = str + "ming";`

Self Check 2.29

What does the following statement sequence print?

```
String str = "Harry";  
int n = str.length();  
String mystery = str.substring(0, 1) + str.substring(n - 1, n);  
System.out.println(mystery);
```

Answer: Hy

Self Check 2.30

Give an input statement sequence to read a name of the form “John Q. Public”.

Answer:

```
String first = in.next();  
String middle = in.next();  
String last = in.next();
```