CHAPTER 2

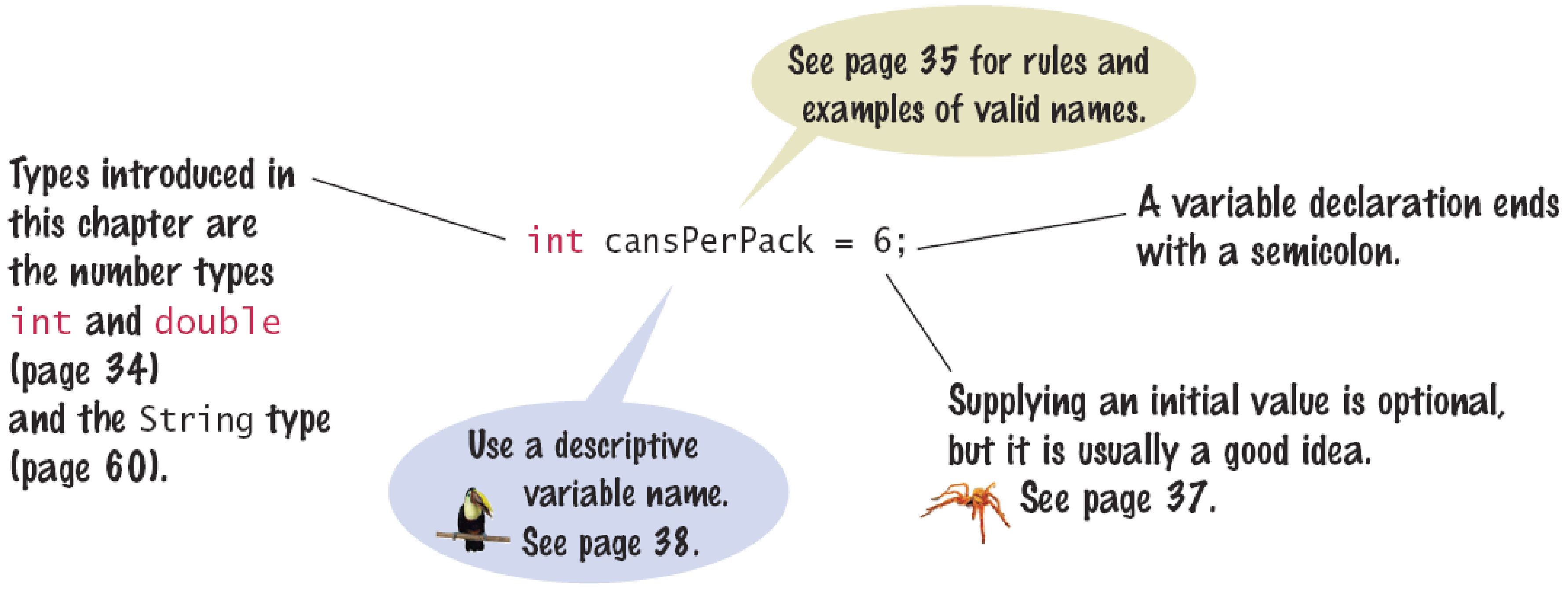
FUNDAMENTAL DATA TYPES

Slides by Donald W. Smith
TechNeTrain.com

Final Draft
Oct. 15, 2011

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


See page 35 for rules and examples of valid names.
Types introduced in this chapter are the number types int and double (page 34) and the String type (page 60).
int cansPerPack = 6;
A variable declaration ends with a semicolon.
Use a descriptive variable name. See page 38.
Supplying an initial value is optional, but it is usually a good idea. See page 37.

Example Declarations

Table 1 Variable Declarations in Java

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

Variable Types

- Common Types

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;
```

Number Literals in Java

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

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

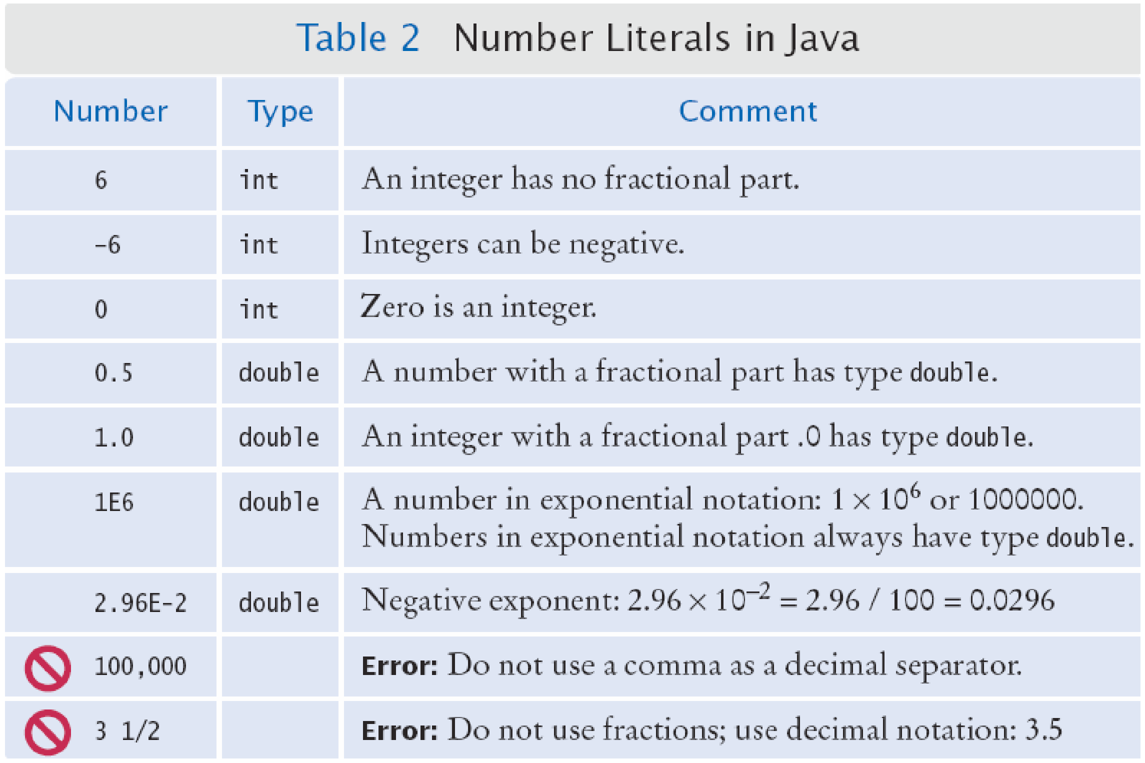


Table 2 Number Literals in Java

	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

Use the double type for floating-point numbers.

Constants

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

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

- 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

  */

Common Errors Examples

- 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 Errors

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

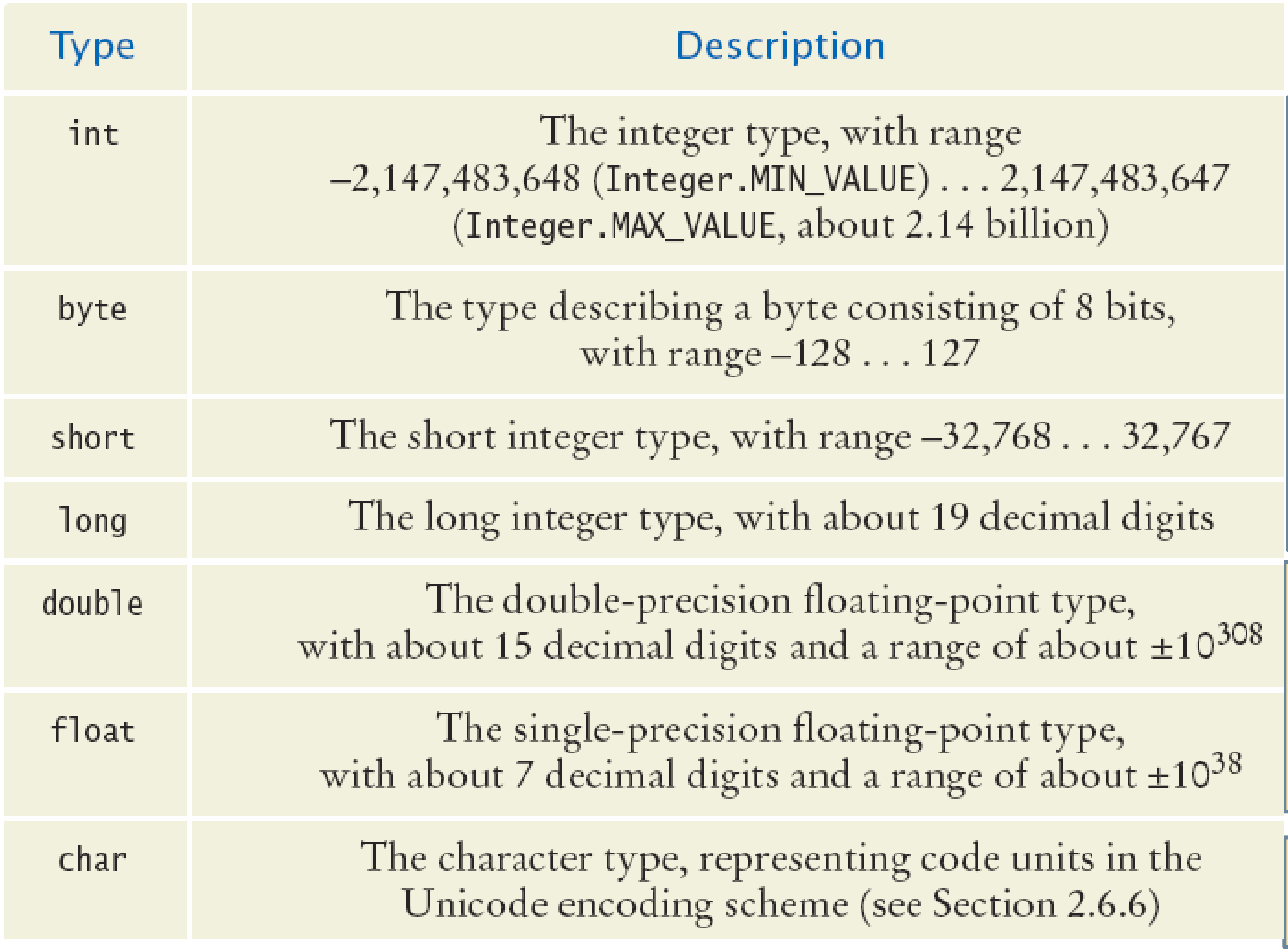
```
int fiftyMillion = 50000000;
System.out.println(100 * fiftyMillion);// Expected: 5000000000
```

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`)

All of the Java Numeric Types



Type	Description
`int`	The integer type, with range −2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)
`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 $\pm 10^{308}$
`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)

Whole Numbers
(no fractions)

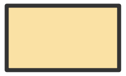
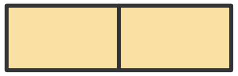
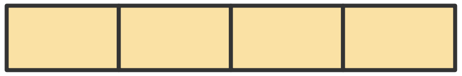
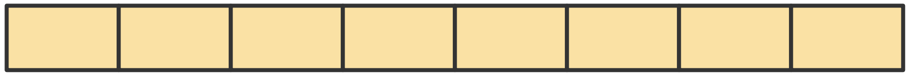
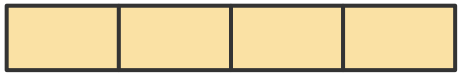
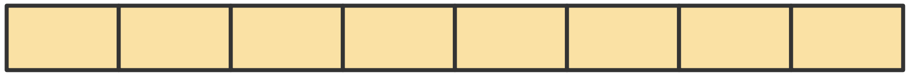
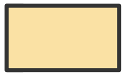
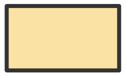
Floating point
Numbers

Characters
(no math)

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:` 

Shorthand for Incrementing

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

Long Way	Shortcut
`counter = counter + 1;`	`counter++ ;`
`counter = counter - 1;`	`counter-- ;`

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;
```

Integer division loses all fractional parts of the result and does not round

  - 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

Integer Division and Remainder Examples

Expression (where n = 1729)	Value	Comment
`n % 10`	9	`n % 10` is always the last digit of `n`.
`n / 10`	172	This is always `n` without the last digit.
`n % 100`	29	The last two digits of `n`.
`n / 10.0`	172.9	Because 10.0 is a floating-point number, the fractional part is not discarded.
`-n % 10`	-9	Because the first argument is negative, the remainder is also negative.
`n % 2`	1	`n % 2` is 0 if `n` is even, 1 or –1 if `n` is odd.

- Handy to use for making change:

```
int pennies = 1729;
int dollars = pennies / 100;  // 17
int cents   = pennies % 100;  // 29
```

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)
  ```

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

Mathematical Methods

Method	Returns
`Math.sqrt(x)`	Square root of x (≥ 0)
`Math.pow(x, y)`	x^y ($x > 0$, or $x = 0$ and $y > 0$, or $x < 0$ and y is an integer)
`Math.sin(x)`	Sine of x (x in radians)
`Math.cos(x)`	Cosine of x
`Math.tan(x)`	Tangent of x
`Math.toRadians(x)`	Convert x degrees to radians (i.e., returns $x \cdot \pi/180$)
`Math.toDegrees(x)`	Convert x radians to degrees (i.e., returns $x \cdot 180/\pi$)
`Math.exp(x)`	e^x
`Math.log(x)`	Natural log ($\ln(x)$, $x > 0$)

Floating-Point to Integer Conversion

- Compiler does not allow direct assignment of a floating-point value to an integer variable

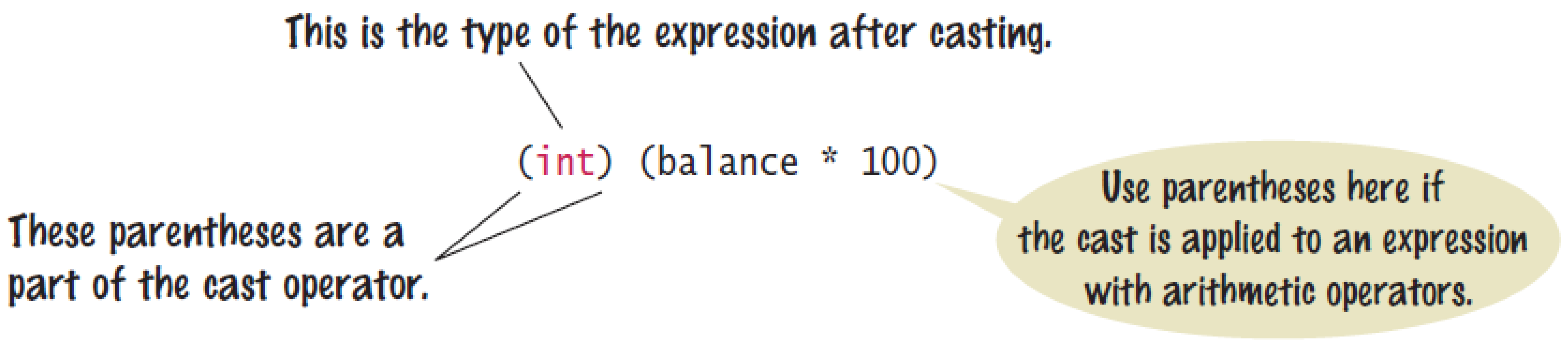
```
double balance = 245.73;
int dollars = balance; // Error
```

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

```
double balance = 245.73;
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
$\frac{x+y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	`Math.pow(1 + r / 100, n)`	Use `Math.pow(x, n)` to compute x^n.
$\sqrt{a^2 + b^2}$	`Math.sqrt(a * a + b * b)`	`a * a` is simpler than `Math.pow(a, 2)`.
$\frac{i+j+k}{3}$	`(i + j + k) / 3.0`	If i, j, and k are integers, using a denominator of 3.0 forces floating-point division.
π	`Math.PI`	`Math.PI` is a constant declared in the `Math` class.

Common Errors

- 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 `int`s, 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`

Input and Output

Input

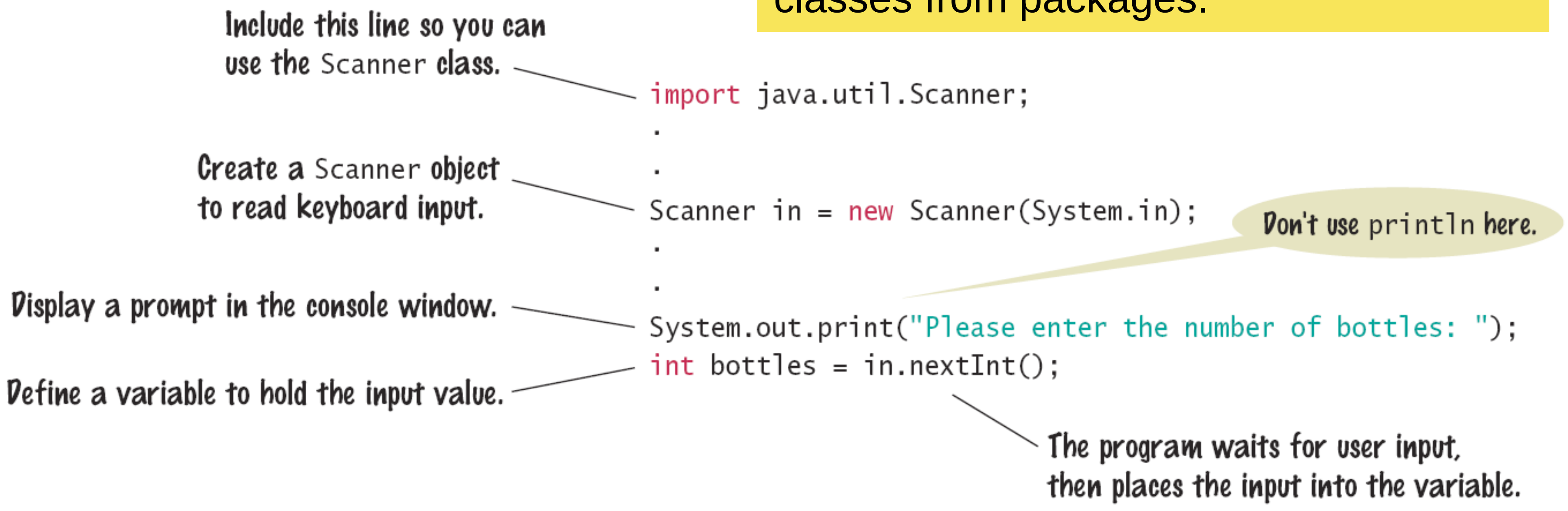
- Reading input from the keyboard
  - For now, don't worry about the details

- 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 of 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.



```
import java.util.Scanner;
.
.
Scanner in = new Scanner(System.in);
.
.
System.out.print("Please enter the number of bottles: ");
int bottles = in.nextInt();
```

Include this line so you can use the Scanner class.
Create a Scanner object to read keyboard input.
Don't use println here.
Display a prompt in the console window.
Define a variable to hold the input value.
The program waits for user input, then places the input into the variable.

Formatted Output

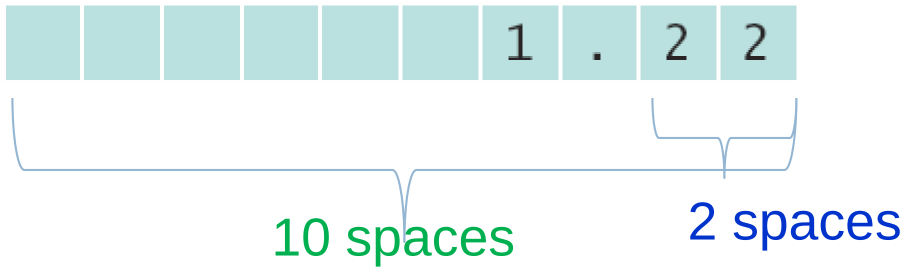
- Outputting floating point values can look strange:

```
    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

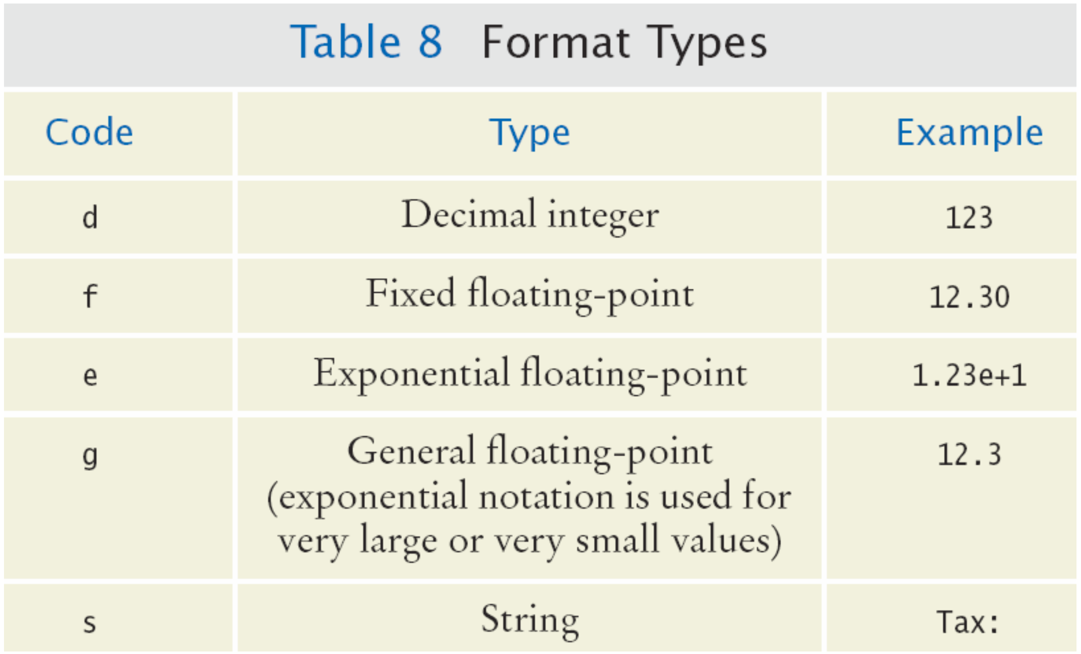


Table 8 Format Types

Code	Type	Example
d	Decimal integer	123
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation is used for very large or very small values)	12.3
s	String	Tax:

- You can also include text inside the quotes:

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

Format Flags

- You can also use format flags to change the way text and numeric values are output:

Table 9 Format Flags

Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(	Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

Strings

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;    // Agent 7
```

- 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();
```

  - next method reads <u>*one word*</u> 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();
```

  - 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

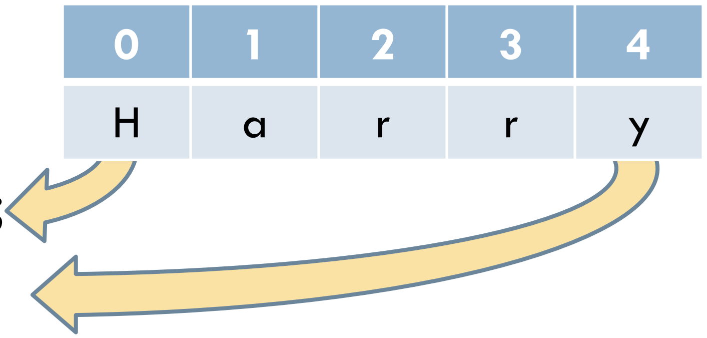
- Each char inside a String has an index number:

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);
```

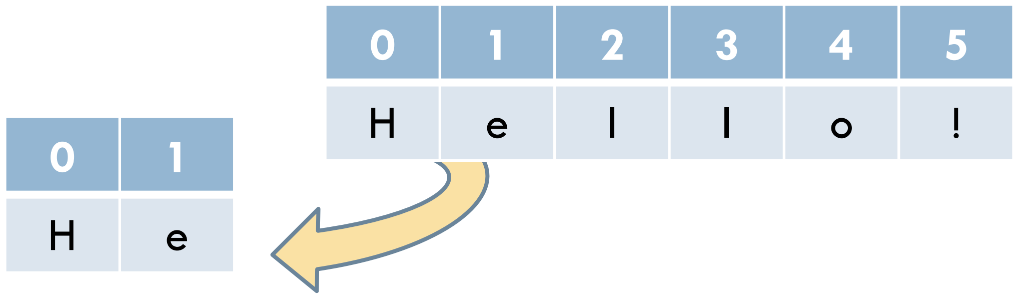


0	1	2	3	4
H	a	r	r	y

Copying portion of 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);
```

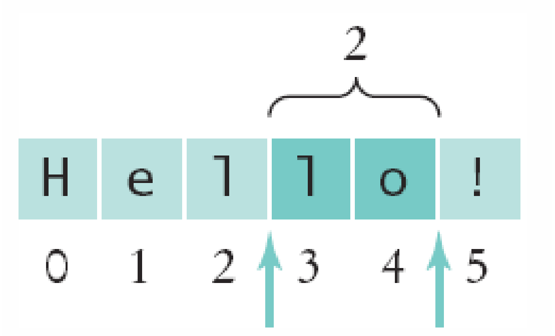


Table 9: String Operations (1)

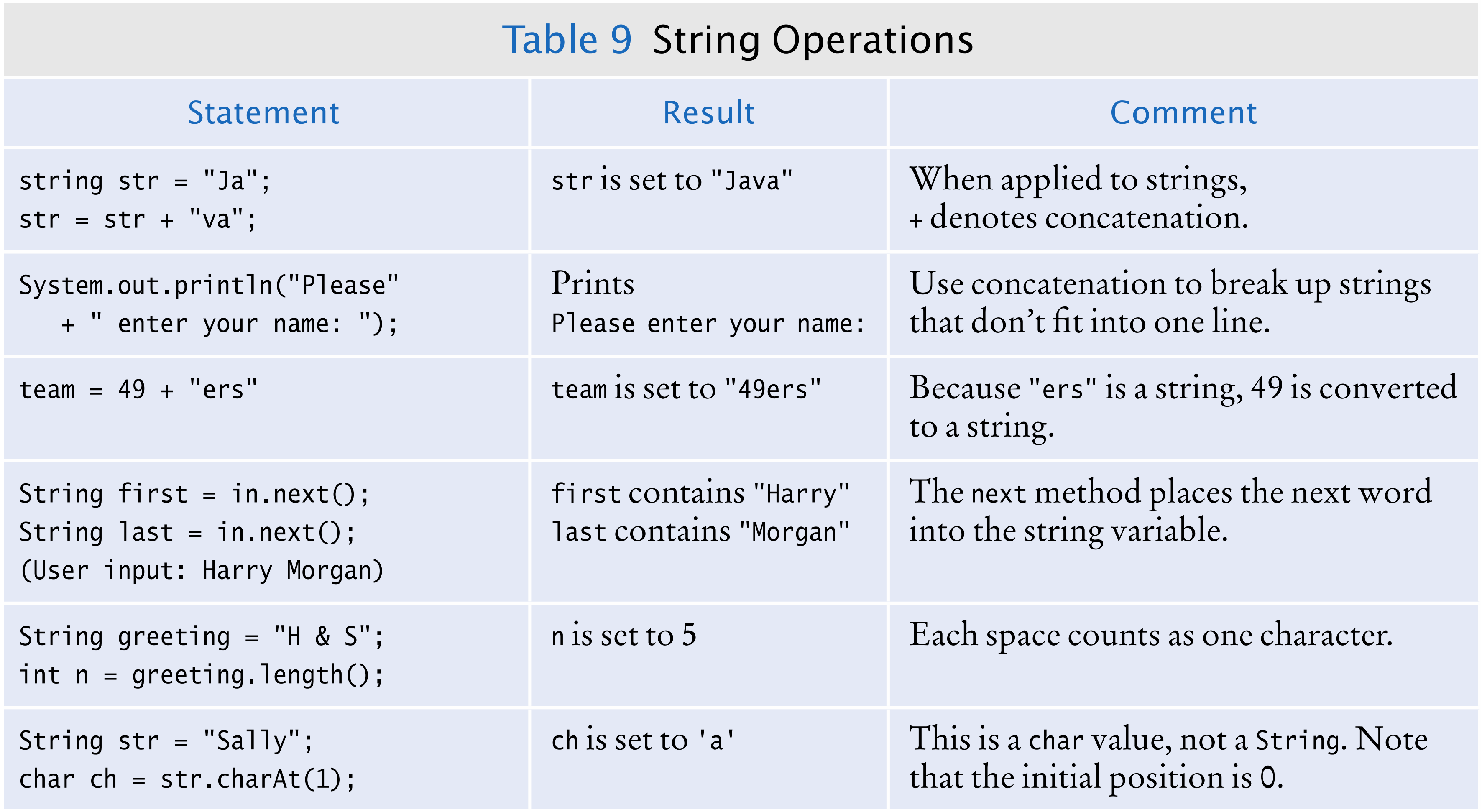


Table 9 String Operations

Statement	Result	Comment

`string str = "Ja";`
`str = str + "va";`
	`str` is set to `"Java"`	When applied to strings,
		`+` denotes concatenation.

`System.out.println("Please"`
`    + " enter your name: ");`
	Prints
	`Please enter your name:`
		Use concatenation to break up strings that don't fit into one line.

`team = 49 + "ers"`	`team` is set to `"49ers"`	Because `"ers"` is a string, 49 is converted to a string.

`String first = in.next();`
`String last = in.next();`
`(User input: Harry Morgan)`
	`first` contains `"Harry"`
	`last` contains `"Morgan"`
		The `next` method places the next word into the string variable.

`String greeting = "H & S";`
`int n = greeting.length();`
	`n` is set to 5	Each space counts as one character.

`String str = "Sally";`
`char ch = str.charAt(1);`
	`ch` is set to `'a'`	This is a `char` value, not a `String`. Note that the initial position is `0`.

Table 9: String Operations (2)

Statement	Result	Comment

`String str = "Sally";`
`String str2 = str.substring(1, 4);`
	`str2` is set to `"all"`	Extracts the substring starting at position 1 and ending before position 4.

`String str = "Sally";`
`String str2 = str.substring(1);`
	`str2` is set to `"ally"`	If you omit the end position, all characters from the position until the end of the string are included.

`String str = "Sally";`
`String str2 = str.substring(1, 2);`
	`str2` is set to `"a"`	Extracts a `String` of length 1; contrast with `str.charAt(1)`.

`String last = str.substring(`
`   str.length() - 1);`
	`last` is set to the string containing the last character in `str`	The last character has position `str.length() - 1`.