Objectives

- Describe the Java data types used for simple data
- Write Java statements to declare variables, define named constants
- Write assignment statements, expressions containing variables and constants
- Define strings of characters, perform simple string processing

Objectives

- Write Java statements that accomplish keyboard input, screen output
- Adhere to stylistic guidelines and conventions
- Write meaningful comments
Outline

- Variables and Expressions
- The Class String
- Keyboard and Screen I/O
- Documentation and Style

Variables and Expressions: Outline

- Variables
- Data Types
- Java Identifiers
- Assignment Statements
- Simple Input
- Simple Screen Output
- Constants
- Named Constants

Variables and Expressions: Outline

- Assignment Compatibilities
- Type Casting
- Arithmetic Operations
- Parentheses and Precedence Rules
- Specialized Assignment Operators
- Case Study: Vending Machine Change
- Increment and Decrement Operators
Variables

- Variables store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored by a variable is called its value.
  - The value is stored in the memory location.
  - Its value can be changed.

Variables and Values

- Variables
  - `numberOfBaskets`
  - `eggsPerBasket`
  - `totalEggs`
- Assigning values
  - `eggsPerBasket = 6;`
  - `eggsPerBasket = eggsPerBasket - 2;`
Naming and Declaring Variables

• Choose names that are helpful such as count or speed, but not c or s.
• When you declare a variable, you provide its name and type.
  ```java
  int numberOfBaskets, eggsPerBasket;
  ```
• A variable’s type determines what kinds of values it can hold (int, double, char, etc.).
• A variable must be declared before it is used.

Syntax and Examples

• Syntax
  ```java
  type variable_1, variable_2, …;
  ```
  (variable_1 is a generic variable called a syntactic variable)
• Examples
  ```java
  int styleChoice, numberOfChecks;
  double balance, interestRate;
  char jointOrIndividual;
  ```

Data Types

• A class type is used for a class of objects and has both data and methods.
  • "Java is fun" is a value of class type String
• A primitive type is used for simple, nondecomposable values such as an individual number or individual character.
  • int, double, and char are primitive types.
Primitive Types

- Figure 2.1 Primitive Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Integer</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>2 bytes</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>Integer</td>
<td>8 bytes</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>Floating-point</td>
<td>4 bytes</td>
<td>±3.402823466 × 10^-38 to ±1.79763269 × 10^38</td>
</tr>
<tr>
<td>double</td>
<td>Floating-point</td>
<td>8 bytes</td>
<td>±1.79763269 × 10^38 to ±1.79763269 × 10^38</td>
</tr>
<tr>
<td>char</td>
<td>Single character (Unicode)</td>
<td>2 bytes</td>
<td>All Unicode values from 0 to 65,535</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>True or false</td>
<td></td>
</tr>
</tbody>
</table>

Java Identifiers

- An identifier is a name, such as the name of a variable.
- Identifiers may contain only:
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning
- The first character cannot be a digit.

Java Identifiers

- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:
  - 7-ll netscape.com util.* (not allowed)
- Identifiers can be arbitrarily long.
- Since Java is case sensitive, stuff, Stuff, and STUFF are different identifiers.
Keywords or Reserved Words

- Words such as `if` are called *keywords or reserved words* and have special, predefined meanings.
- Cannot be used as identifiers.
- See Appendix 1 for a complete list of Java keywords.
- Example keywords: `int`, `public`, `class`

Naming Conventions

- Class types begin with an uppercase letter (e.g. `String`).
- Primitive types begin with a lowercase letter (e.g. `int`).
- Variables of both class and primitive types begin with a lowercase letters (e.g. `myName, myBalance`).
- Multiword names are "punctuated" using uppercase letters.

Where to Declare Variables

- Declare a variable
  - Just before it is used or
  - At the beginning of the section of your program that is enclosed in `{}`.

```java
public static void main(String[] args) {
    /* declare variables here */
    ...
}
```
Primitive Types

- Four integer types (byte, short, int, and long)
  - int is most common
- Two floating-point types (float and double)
  - double is more common
- One character type (char)
- One boolean type (boolean)

Examples of Primitive Values

- Integer types
  0  -1  365  12000
- Floating-point types
  0.99  -22.8  3.14159  5.0
- Character type
  'a'  'A'  '#'  '
- Boolean type
  true  false

Assignment Statements

- An assignment statement is used to assign a value to a variable.
  \[ \text{answer} = 42; \]
- The "equal sign" is called the assignment operator.
- We say, "The variable named \text{answer} is assigned a value of 42," or more simply, "\text{answer} is assigned 42."
Assignment Statements

- Syntax
  variable = expression

  where expression can be another variable, a literal or constant (such as a number), or something more complicated which combines variables and literals using operators (such as + and -)

Assignment Examples

amount = 3.99;
firstInitial = 'W';
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;

Initializing Variables

- A variable that has been declared, but no yet given a value is said to be uninitialized.
- Uninitialized class variables have the value null.
- Uninitialized primitive variables may have a default value.
- It's good practice not to rely on a default value.
Initializing Variables

• To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.

• Examples:
  
  int count = 0;
  char grade = 'A';

• syntax

  type variable_1 = expression_1,
  variable_2 = expression_2, ...;

Assignment Evaluation

• The expression on the right-hand side of the assignment operator (=) is evaluated first.

• The result is used to set the value of the variable on the left-hand side of the assignment operator.

  score = numberOfCards + handicap;
  eggsPerBasket = eggsPerBasket - 2;
Simple Input

• Sometimes the data needed for a computation are obtained from the user at run time.
• Keyboard input requires

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

at the beginning of the file.

Simple Input

• Data can be entered from the keyboard using

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

followed, for example, by

```java
eggsPerBasket = keyboard.nextInt();
```

which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`.

Simple Input

• View sample program listing 2.2

```java
class EggBasket2
```

Sample screen output
Simple Screen Output

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

- Outputs the string literal "the count is"
- Followed by the current value of the variable `count`.

Constants

- Literal expressions such as `2`, `3.7`, or `'y'` are called constants.
- Integer constants can be preceded by a `+` or `-` sign, but cannot contain commas.
- Floating-point constants can be written
  - With digits after a decimal point or
  - Using e notation.

E Notation

- e notation is also called scientific notation or floating-point notation.
- Examples
  - `865000000.0` can be written as `8.65e8`
  - `0.000483` can be written as `4.83e-4`
- The number in front of the `e` does not need to contain a decimal point.
Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence $1.0/3.0$ is slightly less than $1/3$.
- $1.0/3.0 + 1.0/3.0 + 1.0/3.0$ is less than $1$.

Named Constants

- Java provides mechanism to …
  - Define a variable
  - Initialize it
  - Fix the value so it cannot be changed

  ```java
  public static final Type Variable = Constant;
  ```

  ```java
  public static final double PI = 3.14159;
  ```

Assignment Compatibilities

- Java is said to be strongly typed.
  - You can’t, for example, assign a floating point value to a variable declared to store an integer.
  - Sometimes conversions between numbers are possible.

  ```java
  doubleVariable = 7;
  ```

  is possible even if `doubleVariable` is of type `double`, for example.
Assignment Compatibilities

- A value of one type can be assigned to a variable of any type further to the right:
  - `byte` --> `short` --> `int` --> `long` --> `float` --> `double`
- But not to a variable of any type further to the left.
- You can assign a value of type `char` to a variable of type `int`.

Type Casting

- A `type cast` temporarily changes the value of a variable from the declared type to some other type.
- For example,
  ```java
  double distance;
  distance = 9.0;
  int points;
  points = (int)distance;
  ```
- Illegal without `(int)`

Type Casting

- The value of `(int)distance` is 9.
- The value of `distance`, both before and after the cast, is 9.0.
- Any nonzero value to the right of the decimal point is truncated rather than rounded.
Arithmetic Operators

• Arithmetic expressions can be formed using the +, -, *, and / operators together with variables or numbers referred to as operands.
  • When both operands are of the same type, the result is of that type.
  • When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.

Arithmetic Operations

• Example
  
  If hoursWorked is an int to which the value 40 has been assigned, and payRate is a double to which 8.25 has been assigned

  hoursWorked * payRate

  is a double with a value of 500.0.

Arithmetic Operations

• Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  • The result of one step produces one of the operands to be used in the next step.

  example

  balance + (balance * rate)
Arithmetic Operations

- If at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.
- The result is the rightmost type from the following list that occurs in the expression.
  - byte --> short --> int --> long
  - float --> double

The Division Operator

- The division operator (/) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, 99/100 has a value of 0.

The mod Operator

- The mod (%) operator is used with operators of integer type to obtain the remainder after integer division.
  - 14 divided by 4 is 3 with a remainder of 2.
    - Hence, 14 % 4 is equal to 2.
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer.
Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed.
- Examples:
  - $(\text{cost} + \text{tax}) \times \text{discount}$
  - $(\text{cost} + (\text{tax} \times \text{discount})$
- Without parentheses, an expression is evaluated according to the rules of precedence.

Principles of Precedence

- Figure 2.2 Precedence Rules

- The binary arithmetic operators *, /, and % have lower precedence than the unary operators +, -, ++, --, and !, but have higher precedence than the binary arithmetic operators + and -.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.
Precedence Rules

- When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.
- Even when parentheses are not needed, they can be used to make the code clearer.
  \[ \text{balance} + (\text{interestRate} \times \text{balance}) \]
- Spaces also make code clearer
  \[ \text{balance} + \text{interestRate} \times \text{balance} \]
  but spaces do not dictate precedence.

Sample Expressions

- Figure 2.3 Some Arithmetic Expressions in Java

<table>
<thead>
<tr>
<th>Ordinary Math</th>
<th>Java (Prefixed Form)</th>
<th>Java (Parenthesized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate + delta</td>
<td>rate + rate + delta</td>
<td>(rate + rate) + delta</td>
</tr>
<tr>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
</tr>
<tr>
<td>1 / (time + 3 * mass)</td>
<td>1 / (time + 3 * mass)</td>
<td>1 / (time + 3 * mass)</td>
</tr>
<tr>
<td>( \frac{a - 7}{t + 9 \times v} )</td>
<td>( \frac{a - 7}{t + 9 \times v} )</td>
<td>( \frac{a - 7}{t + 9 \times v} )</td>
</tr>
</tbody>
</table>

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including -, *, /, and %, discussed later).
  \[ \text{amount} = \text{amount} + 5; \]
  can be written as
  \[ \text{amount} += 5; \]
  yielding the same results.
Case Study: Vending Machine Change

• Requirements
  • The user enters an amount between 1 cent and 99 cents.
  • The program determines a combination of coins equal to that amount.
  • For example, 55 cents can be two quarters and one nickel.

Case Study

• Sample dialog
  
Enter a whole number from 1 to 99.
The machine will determine a combination of coins.

87
87 cents in coins:
  3 quarters
  1 dime
  0 nickels
  2 pennies

Case Study

• Variables needed
  
  int amount, quarters, dimes, nickels, pennies;
Case Study

• Algorithm – first version
1. Read the amount.
2. Find the maximum number of quarters in the amount.
3. Subtract the value of the quarters from the amount.
4. Repeat the last two steps for dimes, nickels, and pennies.
5. Print the original amount and the quantities of each coin.

Case Study, cont.

• The algorithm doesn’t work properly
  • Original amount is changed by the intermediate steps.
  • Original value of amount is lost.
• Change the list of variables
  int amount, originalAmount, quarters, dimes, nickles, pennies;
  • Update the algorithm.

Case Study

• Algorithm – second version
1. Read the amount.
2. Make a copy of the amount.
3. Find the maximum number of quarters in the amount.
4. Subtract the value of the quarters from the amount.
5. Repeat the last two steps for dimes, nickels, and pennies.
6. Print the original amount and the quantities of each coin.
Case Study

• View Java code that implements the algorithm written in pseudocode – listing 2.3

Enter a whole number from 1 to 99, I will find a combination of coins that equals that amount of change.
87
87 cents in coins can be given as:
3 quarters
1 dime
0 nickels and
2 pennies

Case Study

• How do we determine the number of quarters (or dimes, nickels, or pennies) in an amount?
• There are 2 quarters in 55 cents, but there are also 2 quarters in 65 cents.
• That’s because
  \[
  55 \div 2 = 2 \text{ and } 65 \div 25 = 2.
  \]

Case Study

• How do we determine the remaining amount?
• The remaining amount can be determined using the mod operator
  \[
  55 \% 25 = 5 \text{ and } 65 \% 25 = 15
  \]
• Similarly for dimes and nickels.
• Pennies are simply \( amount \% 5 \).
Case Study

• The program should be tested with several different amounts.
• Test with values that give zero values for each possible coin denomination.
• Test with amounts close to
  • extreme values such as 0, 1, 98 and 99
  • coin denominations, such as 24, 25, and 26.

Increment and Decrement Operators

• Used to increase (or decrease) the value of a variable by 1
• Easy to use, important to recognize
• The increment operator
count++ or ++count
• The decrement operator
count-- or --count

• equivalent operations
count++; ++count;
count = count + 1;

count--;
--count;
count = count - 1;
Increment and Decrement Operators in Expressions

- after executing
  ```java
  int m = 4;
  int result = 3 * (++m)
  result has a value of 15 and m has a value of 5
  ```
- after executing
  ```java
  int m = 4;
  int result = 3 * (m++)
  result has a value of 12 and m has a value of 5
  ```

The Class String

- We've used constants of type String already.
  "Enter a whole number from 1 to 99."
- A value of type String is a
  - Sequence of characters
  - Treated as a single item.

String Constants and Variables

- Declaring
  ```java
  String greeting;
  greeting = "Hello!";
  ```
  ```java
  or
  String greeting = "Hello!";
  ```
  ```java
  or
  String greeting = new
  String("Hello!");
  ```
- Printing
  ```java
  System.out.println(greeting);
  ```
Concatenation of Strings

- Two strings are **concatenated** using the + operator.
  
  ```java
  String greeting = "Hello";
  String sentence;
  sentence = greeting + " officer";
  System.out.println(sentence);
  ```

- Any number of strings can be concatenated using the + operator.

Concatenating Strings and Integers

```java
String solution;
solution = "The answer is " + 42;
System.out.println(solution);
```

String Methods

- An object of the String class stores data consisting of a sequence of characters.
- Objects have methods as well as data.
- The length() method returns the number of characters in a particular String object.
  
  ```java
  String greeting = "Hello";
  int n = greeting.length();
  ```
The Method `length()`

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```java
int count = command.length();
System.out.println("Length is "+command.length());
count = command.length() + 3;
```

---

String Indices

- Figure 2.4
- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
  - A position is referred to as an index.
  - The 'f' in "Java is fun." is at index 8.

---

String Methods

- `charAt(int index)`
  - Returns the character at `index` in this string. Index numbers begin at 0.
- `compareTo(A_String)`
  - Compares this string with `A_String` to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if `A_String` is first.
- `concat(A_String)`
  - Returns a new string having the same characters as this string concatenated with the characters in `A_String`. You can use the `+` operator instead of `concat`.
- `equals(A_String)`
  - Returns true if this string and `A_String` are equal. Otherwise, return false.

---
String Methods

equalsIgnoreCase(other_string)
Behaves like the method equals, but considers uppercase and lowercase versions of a letter to be the same.

indexOf(A_String)
Returns the index of the first occurrence of the substring A_String within this string. Returns -1 if A_String is not found. Index numbers begin at 0.

lastIndexOf(A_String)
Returns the index of the last occurrence of the substring A_String within this string. Returns -1 if A_String is not found. Index numbers begin at 0.

Figure 2.5b

String Methods

length()
Returns the length of this string.

toLowerCase()
Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.

toUpperCase()
Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.

Figure 2.5c

String Methods

replace(OldChar, NewChar)
Returns a new string having the same characters as this string, but with each occurrence of OldChar replaced by NewChar.

substring(start)
Returns a new string having the same characters as the substring that begins at index start of this string through to the end of the string. Index numbers begin at 0.

substring(start, end)
Returns a new string having the same characters as the substring that begins at index start of this string through, but not including, index end of the string. Index numbers begin at 0.

trim()
Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

Figure 2.5d
String Processing

• No methods allow you to change the value of a String object.
• But you can change the value of a String variable.
• View sample program StringDemo listing 2.4

Escape Characters

• How would you print "Java" refers to a language. ?
• The compiler needs to be told that the quotation marks ("") do not signal the start or end of a string, but instead are to be printed.

```java
System.out.println("\"Java\" refers to a language.");
```

• Figure 2.6
• Each escape sequence is a single character even though it is written with two symbols.
Examples

```java
System.out.println("abc\ndef");
'
```

```java
System.out.println("new\nline");
'
```

```java
char singleQuote = '\';
System.out.println(singleQuote);
'
```

The Unicode Character Set

- Most programming languages use the ASCII character set.
- Java uses the Unicode character set which includes the ASCII character set.
- The Unicode character set includes characters from many different alphabets (but you probably won’t use them).

Keyboard and Screen I/O: Outline

- Screen Output
- Keyboard Input
Screen Output

- We’ve seen several examples of screen output already.
- `System.out` is an object that is part of Java.
- `println()` is one of the methods available to the `System.out` object.

Screen Output

- The concatenation operator (+) is useful when everything does not fit on one line.
  ```java
  System.out.println("Lucky number = " + 13 + "Secret number = " + number);
  ```
- Do not break the line except immediately before or after the concatenation operator (+).

Screen Output

- Alternatively, use `print()`
  ```java
  System.out.print("One, two,");
  System.out.print(" buckle my shoe.");
  System.out.println(" Three, four,");
  System.out.println(" shut the door.");
  ```
- ending with a `println()`.
Keyboard Input

• Java has reasonable facilities for handling keyboard input.
• These facilities are provided by the Scanner class in the java.util package.
  • A package is a library of classes.

Using the Scanner Class

• Near the beginning of your program, insert
  import java.util.Scanner;
• Create an object of the Scanner class
  Scanner keyboard =
    new Scanner (System.in)
• Read data (an int or a double, for example)
  int n1 = keyboard.nextInt();
  double d1 = keyboard.nextDouble();

Keyboard Input Demonstration

• View sample program
  class ScannerDemo, listing 2.5

  Enter two whole numbers separated by one or more spaces:
  4 43
  You entered 42 and 43.
  Next enter two numbers.
  A decimal point is OK.
  2.0 23
  You entered 9.99 and 21.0
  Next enter two words:
  plastic spoon
  You entered "Plastic" and "spoon".
  Next enter a line of text:
  May the hair on your toes grow long and curly.
  You entered "May the hair on your toes grow long and curly."
Some **Scanner** Class Methods

- **Figure 2.7a**

  - `Scanner` Object `next()` method
    - Returns the string value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are white space characters.

  - `Scanner` Object `nextInt()` method
    - Reads the rest of the current keyboard input line and returns the characters read as a value of type `int`. Note that the line terminator (\n) is read and discarded; it is not included in the string returned.

  - `Scanner` Object `nextLine()` method
    - Returns the next keyboard input as a value of type `String`.

  - `Scanner` Object `nextDouble()` method
    - Returns the next keyboard input as a value of type `double`.

  - `Scanner` Object `nextFloat()` method
    - Returns the next keyboard input as a value of type `float`.

---

Some **Scanner** Class Methods

- **Figure 2.7b**

  - `Scanner` Object `nextLong()` method
    - Returns the next keyboard input as a value of type `long`.

  - `Scanner` Object `nextByte()` method
    - Returns the next keyboard input as a value of type `byte`.

  - `Scanner` Object `nextShort()` method
    - Returns the next keyboard input as a value of type `short`.

  - `Scanner` Object `nextBoolean()` method
    - Returns the next keyboard input as a value of type `boolean`. The values of true and false are returned as the words `true` and `false`. Any combination of uppercase or lowercase letters follows the rules for `true` and `false`.

  - `Scanner` Object `useDelimiter(String, RegEx)` method
    - Makes the string `Delimiters` the only delimiter used to separate input. Only the exact string will be a delimiter. In particular, blanks, line breaks, and other white space will no longer be delimiters unless they are part of `Delimiters`. `RegEx` is a simple case of the use of `useDelimiter` method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.

---

**nextLine()** Method Caution

- The **nextLine()** method reads
  - The remainder of the current line,
  - Even if it is empty.
nextLine() Method Caution

- Example – given following declaration.
  ```java
  int n;
  String s1, s2;
  n = keyboard.nextInt();
  s1 = keyboard.nextLine();
  s2 = keyboard.nextLine();
  ```
- Assume input shown
  ```java
  n is set to 42
  but s1 is set to the empty string.
  ```

The Empty String

- A string can have any number of characters, including zero.
- The string with zero characters is called the empty string.
- The empty string is useful and can be created in many ways including
  ```java
  String s3 = "";
  ```

Other Input Delimiters (optional)

- Almost any combination of characters and strings can be used to separate keyboard input.
- To change the delimiter to "##"
  ```java
  keyboard2.useDelimiter("##");
  ```
- Whitespace will no longer be a delimiter for keyboard2 input.
Other Input Delimiters

- View sample program
  class DelimitersDemo, listing 2.6

```java
Enter a line of text with two words:
funny word 1#
The two words are "funny" and "word #1"  
Enter a line of text with two words
delimited by #:
funny word 2#
The two words are "funny" and "word #2"
```

Printing to the console

- `System.out` is an object that is part of the Java language
- `println` is a method invoked by the `System.out` object that can be used for console output
  - The data to be output is given as an argument in parentheses
  - A plus sign is used to connect more than one item
  - Every invocation of `println` ends a line of output
  ```java
  System.out.println("The answer is " + 42);
  ```

`println` Versus `print`

- Another method that can be invoked by the `System.out` object is `print`
- The `print` method is like `println`, except that it does not end a line
  - With `println`, the next output goes on a new line
  - With `print`, the next output goes on the same line
Starting with version 5.0, Java includes a method named `printf` that can be used to produce output in a specific format. The Java method `printf` is similar to the `print` method:

- Like `print`, `printf` does not advance the output to the next line.
- `System.out.printf` can have any number of arguments.
- The first argument is always a `format string` that contains one or more `format specifiers` for the remaining arguments.
- All the arguments except the first are values to be output to the screen.

**printf Format Specifier**

- The code:
  ```java
double price = 19.8;
double value = 12.123;
System.out.print("\$");
System.out.printf("%.2f", price);
System.out.println(" each");
```

will output the line:

$ 19.80 each

- The format string `"%.2f"` indicates the following:
  - End any text to be output and start the format specifier (`%`)
  - Display up to 6 right-justified characters, pad fewer than six characters on the left with blank spaces (i.e., field width is 6)
  - Display exactly 2 digits after the decimal point (`.2`)
  - Display a floating point number, and end the format specifier (i.e., the conversion character is `f`)

**Right and Left Justification in printf**

- The code:
  ```java
double value = 12.123;
double value = 12.123;
System.out.printf("Start%8.2fEnd", value);
System.out.println();
System.out.printf("Start%-8.2fEnd", value);
System.out.println();
```

will output the following:

Start   12.12End
Start12.12   End

- The format string `"Start%8.2fEnd"` produces output that is right justified with three blank spaces before the 12.12.
- The format string `"Start%-8.2fEnd"` produces output that is left justified with three blank spaces after the 12.12.
Multiple arguments with `printf`

- The following code contains a `printf` statement having three arguments:
  - The code:
    ```java
double price = 19.8;
String name = "magic apple";
System.out.printf("$%6.2f for each %s.",
price, name);
System.out.println();
System.out.println("Wow");
```
  - Will output:
    ```text
    $ 19.80 for each magic apple.
    Wow
    ```
  - Note that the first argument is a format string containing two format specifiers (`%6.2f` and `%s`).
  - These format specifiers match up with the two arguments that follow (`price` and `name`).

Line Breaks with `printf`

- Line breaks can be included in a format string using `%n`
  - The code:
    ```java
double price = 19.8;
String name = "magic apple";
System.out.printf("$%6.2f for each %s.%n",
price, name);
System.out.println("Wow");
```
  - Will output:
    ```text
    $ 19.80 for each magic apple.
    Wow
    ```

Format Specifiers for `System.out.printf`

<table>
<thead>
<tr>
<th>Character</th>
<th>Type of Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal/integer</td>
<td>123</td>
</tr>
<tr>
<td>f</td>
<td>Float/double</td>
<td>12.34</td>
</tr>
<tr>
<td>e</td>
<td>Exponential</td>
<td>1e-3</td>
</tr>
<tr>
<td>g</td>
<td>General</td>
<td>12.34f</td>
</tr>
<tr>
<td>s</td>
<td>String</td>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>c</td>
<td>Character</td>
<td>%c</td>
</tr>
</tbody>
</table>
### The printf Method (Part 1 of 3)

```java
1. public class PrintDemo
2. {
3.   public static void main(String[] args)
4.   {
5.     String strInt = "123";
6.     System.out.println("String output:");
7.     System.out.printf("%s\\n", strInt);
8.     System.out.printf("%s\\n", args[0]);
9.     System.out.printf("%s\\n", args[1]);
10.    System.out.printf("%s\\n", args[2]);
11.    System.out.println();
```

### The printf Method (Part 2 of 3)

```java
12.     char oneCharacter = 'Z';
13.     System.out.printf("Character output:\n", oneCharacter);
14.     System.out.printf("%c\n", oneCharacter);
15.     System.out.printf("%c\n", oneCharacter);
16.     System.out.printf("%c\n", oneCharacter);
17.     System.out.printf("\n");
18.     double d = 123.456789;
19.     System.out.printf("Floating-point output:\n", d);  // Note: printf does not support floating-point output.
20.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
21.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
22.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
23.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
24.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
25.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
26.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
27.     System.out.printf("%f\n", d);  // Use DecimalFormat for formatting.
28. }
```

### The printf Method (Part 3 of 3)

Note that the output is rounded, not truncated, when digits are discarded.
Legacy Code

• Code that is "old fashioned" but too expensive to replace is called legacy code
• Sometimes legacy code is translated into a more modern language
• The Java method printf is just like a C language function of the same name
• This was done intentionally to make it easier to translate C code into Java

Documentation and Style: Outline

• Meaningful Names
• Comments
• Indentation
• Named Constants

Documentation and Style

• Most programs are modified over time to respond to new requirements.
• Programs which are easy to read and understand are easy to modify.
• Even if it will be used only once, you have to read it in order to debug it.
Meaningful Variable Names

- A variable's name should suggest its use.
- Observe conventions in choosing names for variables:
  - Use only letters and digits.
  - "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  - Start variables with lowercase letters.
  - Start class names with uppercase letters.

Comments

- The best programs are self-documenting.
  - Clean style
  - Well-chosen names
- Comments are written into a program as needed to explain the program.
  - They are useful to the programmer, but they are ignored by the compiler.

Comments

- A comment can begin with `//`.
- Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

```java
double radius; // in centimeters
```
Comments

• A comment can begin with /* and end with */
• Everything between these symbols is treated as a comment and is ignored by the compiler.

/*
This program should only
be used on alternate Thursdays,
except during leap years, when it should
only be used on alternate Tuesdays.
*/

Comments

• A javadoc comment, begins with /** and ends with */.
• It can be extracted automatically from Java software.

/**
method change requires the
number of coins to be nonnegative
*/

When to Use Comments

• Begin each program file with an explanatory comment
  • What the program does
  • The name of the author
  • Contact information for the author
  • Date of the last modification.
• Provide only those comments which the expected reader of the program file will need in order to understand it.
Comments Example

- View sample program
  class CircleCalculation, listing 2.7

Sample Screen Output

Enter the radius of a circle in inches: 2.5
A circle of radius 2.5 inches has an area of 19.629375 square inches.

Indentation

- Indentation should communicate nesting clearly.
- A good choice is four spaces for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.

Indentation

- Indentation does not change the behavior of the program.
- Proper indentation helps communicate to the human reader the nested structures of the program.
Using Named Constants

• To avoid confusion, always name constants (and variables).
  \[ \text{area} = \pi \times \text{radius} \times \text{radius}; \] 
  is clearer than 
  \[ \text{area} = 3.14159 \times \text{radius} \times \text{radius}; \]

• Place constants near the beginning of the program.

Named Constants

• Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.
  \[ \text{public static final} \ \text{double \ INTEREST\_RATE} = 6.65; \]

• If a literal (such as 6.65) is used instead, every occurrence must be changed, with the risk than another literal with the same value might be changed unintentionally.

Declaring Constants

• Syntax
  \[ \text{public static final} \ \text{Variable\_Type} = \text{Constant}; \]

• Examples
  \[ \text{public static final} \ \text{double} \ \pi = 3.14159; \]
  \[ \text{public static final} \ \text{String} \ \text{MOTTO} = \ "\text{The customer is always right.}"; \]

• By convention, uppercase letters are used for constants.
Named Constants

• View sample program
class CircleCalculation2, listing 2.8

Sample Screen Output

Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.

Summary

• You have become familiar with Java primitive types (numbers, characters, etc.).
• You have learned about assignment statements and expressions.
• You have learned about strings.
• You have become familiar with classes, methods, and objects.
• You have learned about simple keyboard input and screen output.