Keyboard Input

- Java 5.0 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();

Some Scanner Class Methods

- Figure 2.7a

  Scanner next() - Returns the String value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.
  Scanner nextLine() - Reads the rest of the current keyboard input line and returns the characters read as a value of type String. Note that the line terminator (\n) is read and discarded, it is not included in the string returned.
  Scanner nextInt() - Returns the next keyboard input as a value of type int.
  Scanner nextDouble() - Returns the next keyboard input as a value of type double.
  Scanner nextFloat() - Returns the next keyboard input as a value of type float.
Some **Scanner** Class Methods

- `Scanner.nextInt()`: Returns the next integer input as a value of type int.
- `Scanner.nextLong()`: Returns the next long input as a value of type long.
- `Scanner.nextDouble()`: Returns the next double input as a value of type double.
- `Scanner.nextLine()`: Returns the next keyboard input as a value of type String.
- `Scanner.nextBoolean()`: Returns the next boolean input as a value of type boolean.
- `Scanner.next(char delimiters)`: Returns the next character input based on a given delimiter.
- `Scanner.nextLine()`: Returns the next keyboard input as a value of type String. The remainder of the current line is read, 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
  42
  and don't you forget it.
  ```

- **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.

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.

- 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
```

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

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

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

```java
/**
 * 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

- See Comments.sample.txt

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.
  \[ \text{public static final double INTEREST_RATE} = 6.65; \]
- 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
  \[ \text{public static final Variable_Type = Constant;} \]
- Examples
  \[ \text{public static final double PI} = 3.14159; \]
  \[ \text{public static final String MOTTO} = \text{"The customer is always right."}; \]
- By convention, uppercase letters are used for constants.

Names Constants

- see CircleCalculation2

Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches has an area of 10.349375 square inches.