Errors

- An error in a program is called a bug.
- Eliminating errors is called debugging.
- Three kinds or errors
  - Syntax errors
  - Runtime errors
  - Logic errors

Syntax Errors

- Grammatical mistakes in a program
  - The grammatical rules for writing a program are very strict
- The compiler catches syntax errors and prints an error message.
- Example: using a period where a program expects a comma

Runtime Errors

- Errors that are detected when your program is running, but not during compilation
- When the computer detects an error, it terminates the program and prints an error message.
- Example: attempting to divide by 0

Logic Errors

- Errors that are not detected during compilation or while running, but which cause the program to produce incorrect results
- Example: an attempt to find the sum of 2 numbers but you use the operator '-' instead of '+' will yield an undesired output.

Software Reuse

- Programs not usually created entirely from scratch
- Most contain components which already exist
- Reusable classes are used
  - Design class objects which are general
  - Java provides many classes
  - Note documentation on following slide

`nextInt()` & `nextDouble()` Method

- The `nextInt()` method
  - reads an integer

- The `nextDouble()` method
  - reads a decimal number

Keywords or Reserved Words

- Words such as `if` are called keywords or reserved words and have special, predefined meanings.
  - Cannot be used as identifiers.
- Example keywords: `int`, `public`, `class`

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

- 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 Class `String`

- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

Concatenation of Strings

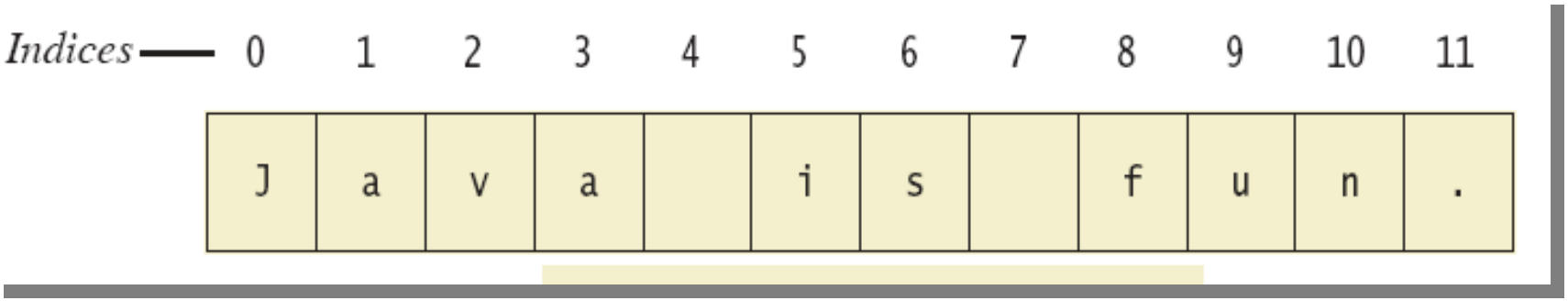
- Two strings are concatenated using the **+** operator.

```
String greeting = "Hello";
String sentence;
sentence = greeting + " doctor";
System.out.println(sentence);
```

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

`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 an an *index.*
  - The `'f'` in `"Java is fun."` is at index 8.

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.

```
String greeting = "Hello";
int n = greeting.length();
```

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.

  `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`**.

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.