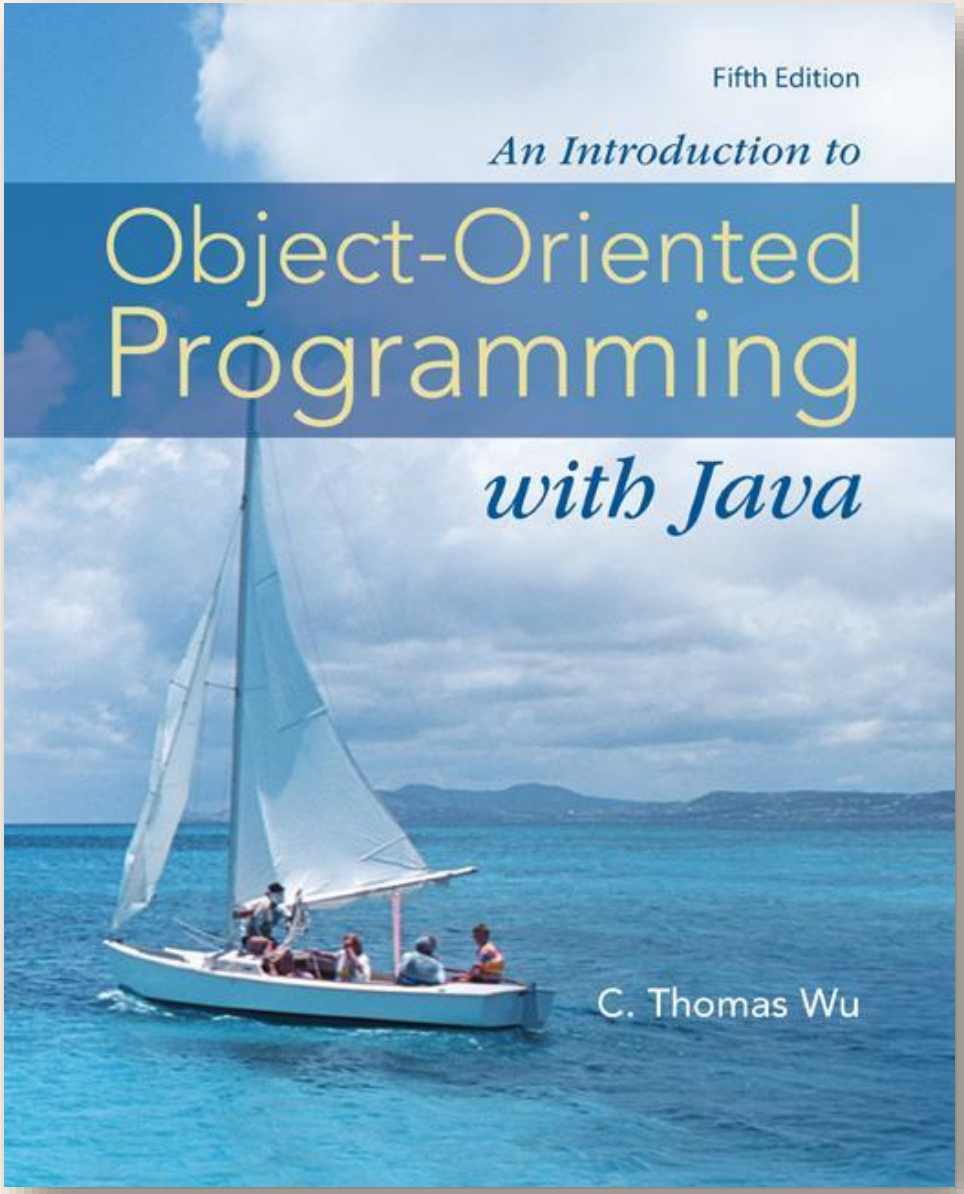


Chapter 9

Characters and Strings

Animated Version

String Class

- A `String` is a sequence (technically, an array) of characters – therefore, formally, “String” is not a built-in data type (unlike `int` and `float`)
- A `String` can be created using a string literal. Example:

```
String name = "Jennifer";
String emtpy = "";
```

- Java `Strings` are *immutable.* This means that it is not possible to change the values of the characters within a `String`.

Regular expressions

- Regular expression ("regex"): a description of a pattern of text
- Can test whether a string matches the expression's pattern
- Regular expressions are extremely powerful but tough to read
- Regular expressions are used in all languages:
  - Java, PHP ,JavaScript, HTML, C#, and other languages
- Many IDEs allow regexes in search/replace

Pattern Example

- Patterns for valid URLs and Email addresses
- Patterns for zip codes, phone numbers etc.
- Patterns for validating password strengths.
- Patterns for search and replace.

Basic regular expressions

The simplest regexes simply matches any string that contains that text.

```
abc
```

The above regular expression matches any string containing "abc":

- YES: "abc", "abcdef", "defabc", ".=.abc.=.", ...
- NO: " ABC" , " fedcba", "ab c", "PHP", ...

- Regular expressions are case-sensitive by default.

Anchors: ^ and $

^ represents the beginning of the string or line;

$ represents the end

- **Jess** matches all strings that contain Jess;
- **^Jess** matches all strings that start with Jess;
- **Jess$** matches all strings that end with Jess;
- **^Jess$** matches the exact string "Jess" only

Wildcards `.`

A dot `.` matches exactly **one-character** except a `\n` (line break)

`.oo.y` matches "Doocy", "goofy", "LooNy", ...

Special characters: |, ()

| means OR

`abc|def|g` matches "abc", "def", or "g"

There's no AND symbol. Why not?

() are for grouping

```
(Homer|Marge) Simpson
```

matches "Homer Simpson" or "Marge Simpson"

Quantifiers: *, +, ?

* means 0 or more occurrences

- **a*** matches "", "a", "aa", "aaa", ...
- **abc*** matches "ab", "abc", "abcc", "abccc", ...
- **a(bc)*** matches "a", "abc", "abcbc", "abcbcbc", ...

+ means 1 or more occurrences

- **a(bc)+** matches "abc", "abcbc", "abcbcbc", ...
- **Goo+gle** matches "Google", "Gooogle", "Goooogle", ...

? means 0 or 1 occurrences

- **a(bc)?** matches "a" or "abc"

More quantifiers: `{min,max}`

`{min,max}` means between min and max occurrences (inclusive)

- `a(bc){2,4}` matches "abcbc", "abcbcbc", or "abcbcbcbc"

`min` or `max` may be omitted to specify any number

- `{2,}` means 2 or more
- `{,6}` means up to 6
- `{3}` means exactly 3

Character sets: [ ]

`[]` group characters into a character set, will match any **single character** from the set

`[bcd]art` matches strings containing "bart", "cart", and "dart"

equivalent to `(b|c|d)art` but shorter

inside [], many of the modifier keys act as normal characters

`what[!*?]*` matches "what", "what!", "what?**!", "what??!", ...

Character ranges: `[start-end]`

inside a character set, specify a range of characters with hyphen -

- `[a-z]` matches any lowercase letter
- `[a-zA-Z0-9]` matches any lower- or uppercase letter or digit

an initial `^` inside a character set negates it

- `[^abcd]` matches any character other than a, b, c, or d

inside a character set, - must be escaped to be matched

- `[+\-]?[0-9]+` matches an optional + or -, followed by at least one digit

What regular expression matches letter grades such as A, B+, or D- ?

Escape sequences

Special escape sequence character sets:

`\d` matches any digit (same as [0-9])

`\D` any non-digit ([^0-9])

`\w` matches any word character
(same as [a-zA-Z_0-9])

`\W` any non-word char

`\s` matches any whitespace character (, \t, \n, etc.)

`\S` any non-whitespace

What regular expression matches dollar amounts of at least $100.00 ?

Regular Expressions in Java

- Java provides the `java.util.regex` package for pattern matching with regular expressions.
- String methods like `replaceFirst()`, `replaceAll()` and `split()` can make use of regular expression.

The replace(All|First) Method

- The replace(All | First) method replaces all or first occurrence(s) of a substring that matches a given regular expression with a given replacement string.

Replace all vowels with the symbol @

```
String originalText, modifiedText;

originalText = ...;     //assign string

modifiedText =
          originalText.replaceAll("[aeiou]","@");
```

The split Method

- The split method breaks a given string around matches of the given regular expression.

```
Stirng s = "202 546-28374 455-434 345";
System.out.println(Arrays.toString(s.split("-|\s",3)));
```

The Pattern and Matcher Classes

- The matches and replaceAll methods of the String class are shorthand for using the Pattern and Matcher classes from the java.util.regex package.
- If str and regex are String objects, then

```
str.matches(regex);
```

is equivalent to

```
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(str);
matcher.matches();
```

The compile Method

- The compile method of the Pattern class converts the stated regular expression to an internal format to carry out the pattern-matching operation.
- This conversion is carried out every time the matches method of the String class is executed, so it is more efficient to use the compile method when we search for the same pattern multiple times.
- See the sample programs Ch9MatchJavaIdentifier2 and Ch9PMCountJava

The find Method

- The find method is another powerful method of the Matcher class.
  - It searches for the next sequence in a string that matches the pattern, and returns true if the pattern is found.
- When a matcher finds a matching sequence of characters, we can query the location of the sequence by using the start and end methods.

- See Ch9PMCountJava2

Problem Statement

Write an application that will build a word concordance of a document. The output from the application is an alphabetical list of all words in the given document and the number of times they occur in the document. The documents are a text file (contents of the file are an ASCII characters) and the output of the program is saved as an ASCII file also.

Overall Plan

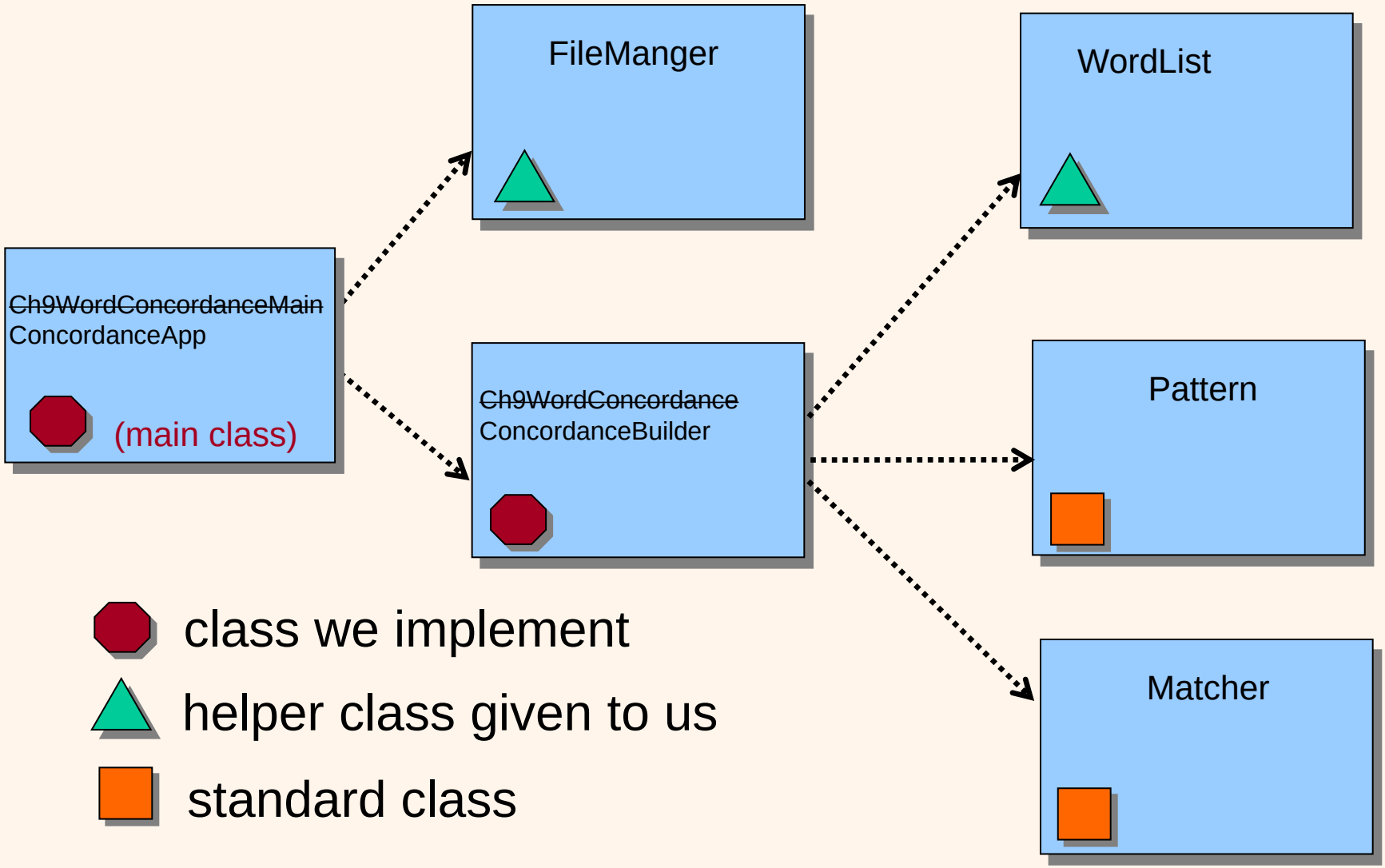
- Tasks expressed in pseudocode:

```
while ( the user wants to process
        another file    ) {

   Task 1: read the file;

   Task 2: build the word list;

   Task 3: save the word list to a file;
}
```

Design Document

Class	Purpose
Ch9WordConcordanceMain	The instantiable main class of the program that implements the top-level program control.
Ch9WordConcordance	The key class of the program. An instance of this class managers other objects to build the word list.
FileManager	A helper class for opening a file and saving the result to a file. Details of this class can be found in Chapter 12.
WordList	Another helper class for maintaining a word list. Details of this class can be found in Chapter 10.
Pattern/Matcher	Classes for pattern matching operations.

Class Relationships



Development Steps

- We will develop this program in four steps:

1. Start with a program skeleton. Define the main class with data members. Begin with a rudimentary Ch9WordConcordance class.
2. Add code to open a file and save the result. Extend the existing classes as necessary.
3. Complete the implemention of the Ch9WordConcordance class.
4. Finalize the code by removing temporary statements and tying up loose ends.

Step 1 Design

- Define the skeleton main class
- Define the skeleton Ch9WordConcordance class that has only an empty zero-argument constructor

Step 1 Code

Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

Directory: Chapter9/Step1

Source Files: Ch9WordConcordanceMain.java
Ch9WordConcordance.java

Step 1 Test

- The purpose of Step 1 testing is to verify that the constructor is executed correctly and the repetition control in the start method works as expected.

Step 2 Design

- Design and implement the code to open and save a file
- The actual tasks are done by the FileManager class, so our objective in this step is to find out the correct usage of the FileManager helper class.
- The FileManager class has two key methods: openFile and saveFile.

Step 2 Code

Directory: Chapter9/Step2

Source Files: Ch9WordConcordanceMain.java
Ch9WordConcordance.java

Step 2 Test

- The Step2 directory contains several sample input files. We will open them and verify the file contents are read correctly by checking the temporary echo print output to System.out.
- To verify the output routine, we save to the output (the temporary output created by the build method of Ch9WordConcordance) and verify its content.
- Since the output is a textfile, we can use any word processor or text editor to view its contents.

Step 3 Design

- Complete the build method of Ch9WordConcordance class.
- We will use the second helper class WordList here, so we need to find out the details of this helper class.
- The key method of the WordList class is the add method that inserts a given word into a word list.

Step 3 Code

Directory: Chapter9/Step3

Source Files: Ch9WordConcordanceMain.java
Ch9WordConcordance.java

Step 3 Test

- We run the program against varying types of input textfiles.
  - We can use a long document such as the term paper for the last term's economy class (don't forget to save it as a textfile before testing).
  - We should also use some specially created files for testing purposes. One file may contain one word repeated 7 times, for example. Another file may contain no words at all.

Step 4: Finalize

- Possible Extensions
  - One is an integrated user interface where the end user can view both the input document files and the output word list files.
  - Another is the generation of different types of list. In the sample development, we count the number of occurences of each word. Instead, we can generate a list of positions where each word appears in the document.