6.1 Language Support

6.1.1 The String Class
6.1.2 The Masquerade and the + Operator

6.2 String Handling

6.2.1 Overview of String Methods
6.2.2 Accessors
6.2.3 Transformers
6.2.4 Comparators
6.2.5 Numeric Strings

6.3 Applications

6.3.1 Character Frequency
6.3.2 Character Substitution
6.3.3 Fixed-Size Codes
6.3.4 Variable-Size Codes

6.4 Advanced String Handling

6.4.1 The StringBuffer Class
6.4.2 Pattern Matching & Regular Expressions
6.4.3 Command-Line Arguments

Write a fragment that creates a Stock that encapsulates the symbol "ab"
Write a fragment that creates a Stock that encapsulates the symbol "ab"

```java
Stock stk = new Stock("ab");
```

Write a fragment that creates a Fraction that encapsulates 3/5

```java
Fraction f = new Fraction(3, 5);
```
Write a fragment that creates an Equation that encapsulates $3x^2 - 2x + 7 = 0$

```java
Equation e = new Equation(3, -2, 7);
```

Write a fragment that creates a String that encapsulates “York”
Are Strings Special?

Write a fragment that creates a String that encapsulates "York"

```java
String s = new String("York");
```

Creating strings is not different from creating any other object.

---

The Masquerade

Can create a String without using `new`:

```java
String s = "York";
```

The compiler replaces the above with:

```java
String s = new String("York");
```

---

The `+` Operator

Can concatenate two strings using a "fake" operator:

```java
String s = "York" + "Lane";
```

The compiler replaces the above with:

```java
String s = new String("YorkLane");
```

These (convenience) shortcuts make strings "look like" mutable primitive types.
More on the + Operator

How does the compiler handle \( x + y \) ?

- If \( x \) and \( y \) are both numeric, this is the addition operator.
- If either \( x \) or \( y \) is a string, this is the concatenation operator. In this case, the other operand is coerced to a string.
- Otherwise, there is a syntax error in this expression.

6.2 String Handling

Given a String, invoke:

- Accessors
- Transformers
- Comparators
- Numeric/String Converters

**Note the absence of mutators**

String Methods

- `length()`
- `charAt(int)`
- `substring(int, int)` (int)
- `indexOf(String)` (String, int)
- `toString()` and `equals()`
- `compareTo()`
- `toUpperCase()` and `toLowerCase()`
Notes on String Methods
• The "# of character" language versus the position language. They differ by 1.
• Can you live w/o substring(int) given the overloaded (int,int)?
• How would use use indexOf to detect all occurrences of a substring?
Notes on String Methods

- The # of character language versus the position language. They differ by 1.
- Can you live w/o substring(int) given the overloaded (int, int)?
- How would you use indexOf to detect all occurrences of a substring?
- Do not underestimate what equals does!
  Given two very long strings, when does equals deem them equal?

Notes on String Methods

- The power of compareTo.
- The notion of lexicographic ordering.

Notes on String Methods

- Substring and toUpperCase() must return a brand new string.
Numeric Strings

The Wrapper Classes

String s = "1020";
int number = Integer.parseInt(s);

The other way (from number to string) is best handled thru the + operator (see next)

6.3 Applications

Read the four applications in sections 6.3.1-4 and note, in particular, how
indexOf and substring can be used to perform pattern lookup/substitution.

Here, we will discuss three different applications but they employ the same techniques:

Applications:
- SpaceCounter
  Prompt for, and read, a string from the user. Output the number of spaces in it.
- FileSpaceCounter
  Similar to the previous one but it gets its input from the file. The user is prompted to enter the filename.
- DigitSpeller
  Read a string from the user and spell out the names of the digits in it, e.g. input "this6is a5 test4" leads to output: "SIX", "FIVE", and "FOUR".
6.4 Advanced String Handling

- Efficiency calls for immutability
- A separate class, `StringBuffer`, was added to handle mutation.
- The new class has three mutators:
  
  ```
  StringBuffer append(anything)
  StringBuffer insert(int, anything)
  StringBuffer delete(int, int)
  ```

The `+` Operator & StringBuffer

Given two strings `x` and `y`, the compiler replaces:

```
String s = x + y;
```

with:

```
String s = new StringBuffer().append(x).append(y).toString();
```
Command-Line Arguments

Run this app with AABCBA B as arguments:

```java
PrintStream output = System.out;
String s = args[0];
char c = args[1].charAt(0);
int count = 0;
for (int index = 0; index < s.length(); index++)
{
    String token = s.substring(index, index+1);
    if (token.equals("" + c))
    {
        count++;
    }
}
output.println(count);
```

The output is 2.