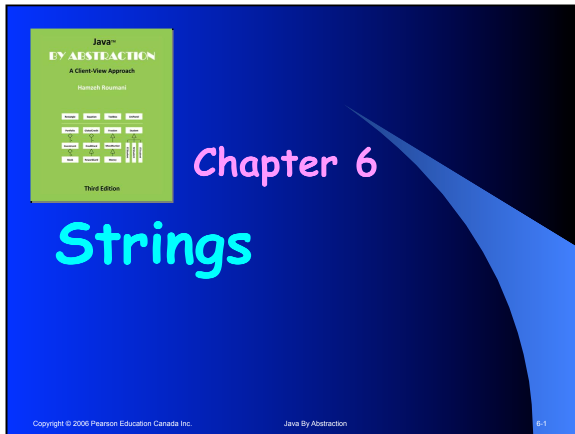

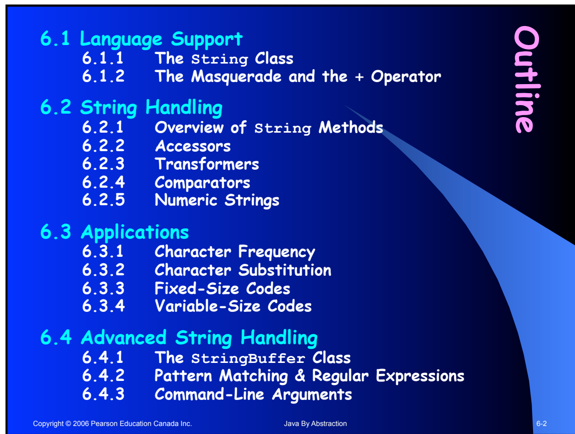

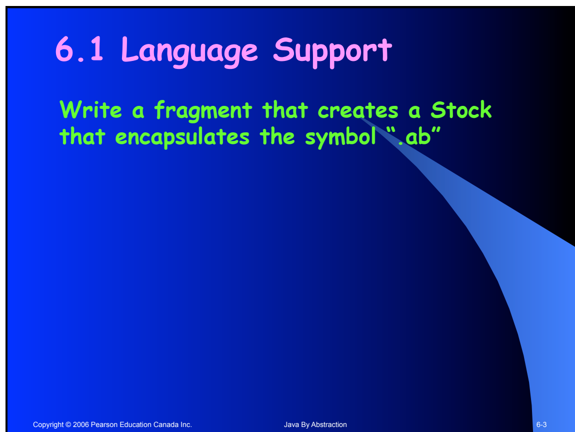

Copyright © 2006 Pearson Education Canada Inc.

Are Strings Special?

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

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

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-4

Are Strings Special?

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

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-5

Are Strings Special?

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

```
Fraction f = new Fraction(3, 5);
```

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-6

Are Strings Special?

Write a fragment that creates an Equation that encapsulates $3x^2 -2x + 7 = 0$

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-7

Are Strings Special?

Write a fragment that creates an Equation that encapsulates $3x^2 -2x + 7 = 0$

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

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-8

Are Strings Special?

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

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-9

Are Strings Special?

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

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

Creating strings is not different from creating any other object.

The Masquerade

Can create a String without using new:

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

The compiler replaces the above with:

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


The + Operator

Can concatenate two strings using a "fake" operator:

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

The compiler replaces the above with:

```
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.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-13

6.2 String Handling

Given a String, invoke:

- Accessors
- Transformers
- Comparators
- Numeric/String Converters

Note the absence of mutators

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-14

String Methods

- length()
- charAt(int)
- substring(int,int) (int)
- indexOf(String) (String,int)
- toString() and equals()
- compareTo()
- toUpperCase() and toLowerCase()

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-15

Copyright © 2006 Pearson Education Canada Inc.

Notes on String Methods

- The "#of character" language versus the position language. They differ by 1.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-16

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)?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-17

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?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-18

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?
- **Do not underestimate what equals does!**
  Given two very long strings, when does equals deem them equal?

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-19

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?
- Do not underestimate what equals does
- **The power of compareTo**
  The notion of lexicographic ordering

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-20

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?
- Do not underestimate what equals does
- The power of compareTo.
- **Substring and toUpper/LowerCase() must return a brand new string**

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-21

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)

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-22

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:

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-23

Applications:

- SpaceCounter
  Prompt for, and read, a string from the user. Output the number of spaces in it.
- FileSpaceCounte
  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".

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-24

Copyright © 2006 Pearson Education Canada Inc.

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)
```

Copyright © 2006 Pearson Education Canada Inc.

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();
```

Copyright © 2006 Pearson Education Canada Inc.

Regular Expressions

CHARACTER SPECIFICATIONS	
[a-m]	*Range. A characters between a and m, inclusive*
[a-m[A-M]]	*Union. a through m or A through M*
[abc]	*Set. The character a, b, or c*
[^abc]	*Negation. Any character except a, b, or c*
[a-m&&[^ck]]	*Intersection. a though m but neither c nor k*
PREDEFINED SPECIFICATIONS	
.	*Any character*
\d	*A digit, [0-9]*
\s	*A whitespace character,* [\t\n\x0B\f\r]
\w	*A word character, [a-zA-Z_0-9]*
\p{Punct}	*A punctuation,* [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]
QUANTIFIERS	
x?	*x, once or not at all*
x*	*x, zero or more times*
x+	*x, one or more times*
x{n,m}	*x, at least n but no more than m times*

Copyright © 2006 Pearson Education Canada Inc.

Copyright © 2006 Pearson Education Canada Inc.

Command-Line Arguments

Run this app with AABCBA B as arguments:

```
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.

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 6-28