Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Regular expressions

Regular Expression

- Assume you need to check if a string is "acceptable" for a given circumstance.
- Brute Force Solution
  - Compile a long list of every possible acceptable string.
  - Look for the string you are checking for in the list.
- A better solution is to use a regular expression.
- A regular expression defines a set of strings that follow a pattern.
- Regular Expression Solution
  - Create a regular expression that defines which strings are acceptable.
  - Check if the string follows the pattern defined by the regular expression.
- For example, when a user defines a password for a system it must have certain characteristics.
- A regular expression can be used to define the acceptable pattern for passwords.

Regular Expression

Regular Expression

- Regular expressions are useful for validating input data and checking to make sure that data is in a certain format.
- The String matches method checks if a string matches a regular expression.
- **matches** checks the string against a pattern.
- For example:

```
String s = "a";
if ( s.matches("[a-z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}
```

"[a-z]" is a regular expression. The [ ] are used to define a set of characters to search for. matches will return true because 'a' is a character in the given range a-z. Matches one character in the given range.

Regular Expression

- More examples:

False. Capital A does NOT fall within the range

```
String s = "A";
if ( s.matches("[a-z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}
```

True. A-Z will match 'A'

```
s = "A";
if ( s.matches("[a-zA-Z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}
```

Regular Expression

- Examples with multiple characters:

```
String s = "ab";
if ( s.matches("[a-z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}

if ( s.matches("[a-z][a-z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}

if ( s.matches("[a-z][a-z][a-z]") ) {
   System.out.println("Matches");
} else {
   System.out.println("Does NOT match");
}
```

False. The regular expression expects only 1 character.

True. The regular expression expects 2 characters and the characters are within the range.

False. The regular expression expects 3 characters.

- Here are some predefined symbols that search for a range of characters.

Character	Description
.	Matches any character.
\w	Any word character. A word character includes upper and lowercase letters, any digit, or the underscore character.
\W	Any nonword character.
\d	Any digit.
\D	Any nondigit.
\s	Any whitespace.
\S	Any nonwhitespace.

Pattern Matching

- Examples using predefined symbols:

Note

\\ stands for \ in a string (the first \ is the escape char)

```
String s = "a";
System.out.printf("%b\n", s.matches("\\w") ); // true
System.out.printf("%b\n", s.matches("\\W") ); // false
System.out.printf("%b\n", s.matches("\\d") );  // false
System.out.printf("%b\n", s.matches("\\D") );  // true

s = "a1";
System.out.printf("%b\n", s.matches("\\w\\d") ); // true
```

Regular Expression

Quantifiers

- Quantifiers allow you to match multiple occurrences of a subexpression.
- An * matches 0 or more instances of the preceding subexpression.
- A + matches 1 or more instances of the preceding subexpression.
- For example:

```
String s = "aaa";
System.out.printf("%b\n", s.matches("a*") );
System.out.printf("%b\n", s.matches("a+") );

s = "";
System.out.printf("%b\n", s.matches("a*") );
System.out.printf("%b\n", s.matches("a+") );
```

Match 0 or more occurrences of "a"

Both return true

Match 1 or more occurrences of "a"

true

false (requires at least 1)

Quantifiers

- More examples with quantifiers:

Note

\\ stands for \ in a string (the first \ is the escape char)

```
s = "12589";
System.out.printf("%b\n", s.matches("\\d*") ); // true

s = "12589a";
System.out.printf("%b\n", s.matches("\\d*") ); // false
System.out.printf("%b\n", s.matches("\\d*\\w") ); // true
```

Quantifiers

- Quantifiers can be used with [ ].
- For example:

Match any number of a, b, c characters

```
String s = "acb";
System.out.printf("%b\n", s.matches("[a-c]*") );
```

true

```
s = "acx";
System.out.printf("%b\n", s.matches("[a-c]*") );
System.out.printf("%b\n", s.matches("[a-c]*x") );
```

false (the 'x' is not in range)

true (the first two characters match [a-c]* and 'x' matches 'x')

- Table of quantifiers.

Character	Description
*	0 or more occurrences
+	1 or more occurrences
?	0 or 1 occurrences
{n}	Exactly n occurrences. For example, {1} means one occurrence, {3} means 3 occurrences and so on…
{n,}	At least n occurrences. For example, {3,} means 3 or more occurrences.
{n,m}	Between n and m occurrences. For example, {3,5} means 3, 4, or 5 occurrences.

Quantifiers

Quantifiers

- More quantifier examples:
- For example:

```
String s = "a";
System.out.printf("%b\n", s.matches("[a-c]?") );

s = "ac";
System.out.printf("%b\n", s.matches("[a-c]?") );
System.out.printf("%b\n", s.matches("[a-c]?[a-c]?") );
```

Match 0 or 1 a, b, c characters

True

False. Expects 0 or 1 a-c

Matches 'a'

Matches 'c'

True. Expects 0 or 1 a-c followed by another 0 or 1 a-c

- What do the following cause to happen?

```
String s = "ac";
System.out.printf("1 %b\n", s.matches("ac") );
System.out.printf("2 %b\n", s.matches("ac{2}") );
System.out.printf("3 %b\n", s.matches("[a-c][a-c][a-c]") );
System.out.printf("4 %b\n", s.matches("[a-c]?[a-c]?[a-c]?") );
```

Quantifiers

- What do the following cause to happen?

```
String s = "ac";
System.out.printf("1 %b\n", s.matches("ac") );
System.out.printf("2 %b\n", s.matches("ac{2}") );
System.out.printf("3 %b\n", s.matches("[a-c][a-c][a-c]") );
System.out.printf("4 %b\n", s.matches("[a-c]?[a-c]?[a-c]?") );
```

Answer

1 true ← **Exact match**

2 false ← **Requires exactly 2 occurrences of "c" (like "acc)"**

3 false ← **Requires 3 characters that are a-c**

4 true ← **? allows 0 or 1 occurrences. The last [a-c]? can be empty and match.**

Quantifiers

Capturing Group

- You can check for groups of characters.
- Use **( )** to check for a group of characters.
- This is called a capturing group.
- [ac] is different than (ac).
  - [ac] means one character that is either an 'a' or 'c'.
  - (ac) means an 'a' followed by a 'c'
- For example:

```
String s = "acac";
System.out.printf("%b\n", s.matches("(ac)*") );
```

The result is true since ac is followed by ac

Searches for 0 or more occurrences of "ac"

Substitution

- You can replace parts of a string with different characters if you want.
- replaceAll - Replaces all occurrences of one substring with another.
- replaceFirst - Replaces the first occurrence of one substring with another.
- For example:

```
String s = "a,b,c,";

String rs = s.replaceAll( "," , "_");

System.out.printf("%s\n", rs );
```

Replaces all occurrences of ',' with '_'

Prints a_b_c_

Substitution

- More substitution examples:

```
String s, rs;

s = "abcabc";
rs = s.replaceAll("(bc)", "xy");
System.out.printf("%s -> %s\n", s, rs);

s = "abcabc";
rs = s.replaceFirst("(bc)", "xy");
System.out.printf("%s -> %s\n", s, rs);
```

Resulting string is: axyaxy

Resulting string is: axyabc

- End of slides

End of Slides