Boolean methods and the use of the Utilities class

The use of Boolean methods

Produced by: Dr. Siobhán Drohan
Ms. Mairead Meagher

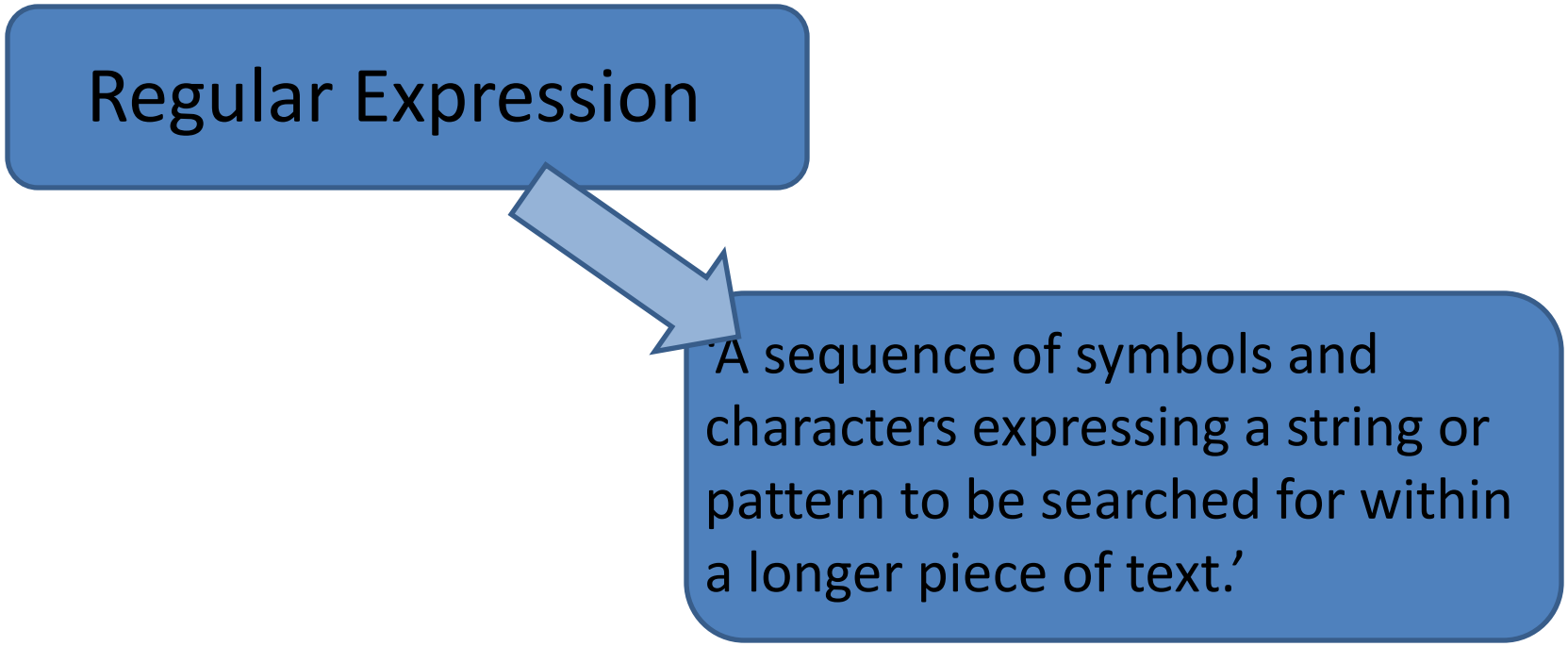
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topics list

1. Regular expressions
2. Methods with int return type
3. Boolean methods
4. **Utilities** class
5. **equals()** method

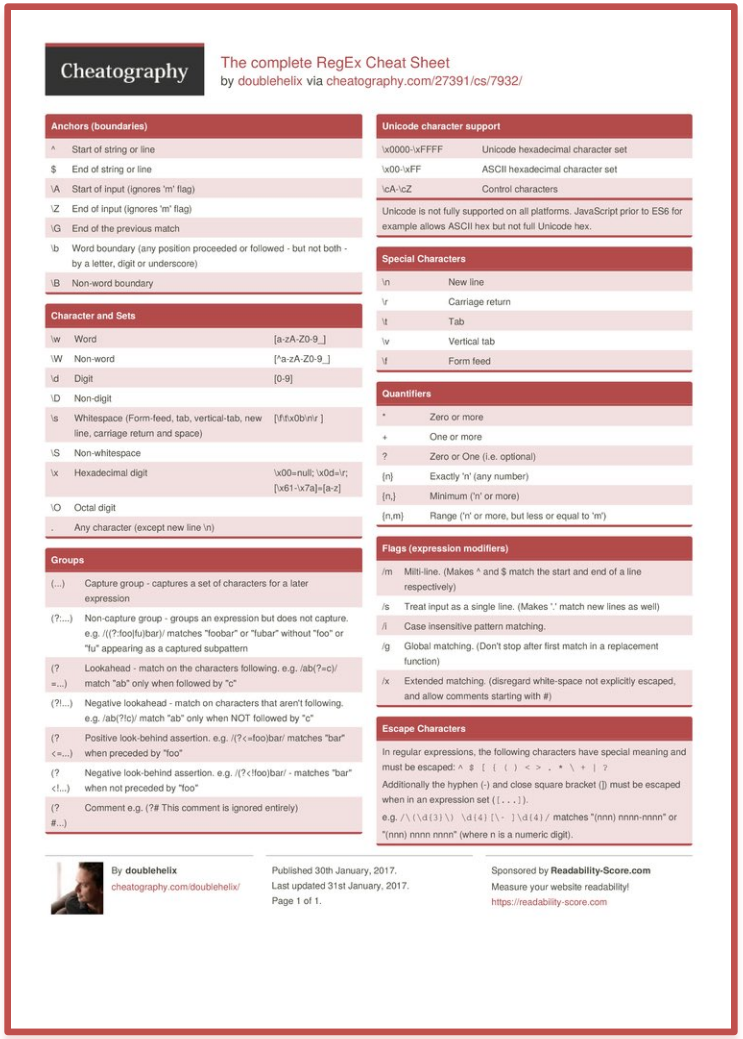
Regular Expressions



Resources

- https://regexone.com/ (good interactive site)
- This site has what you need but looks a bit busy:
  - https://www.regular-expressions.info/numericranges.html
- This site has a nice chart of what all the symbols need.
  - https://www.w3schools.com/jsref/jsref_obj_regexp.asp
- CHEATSHEET
  - https://cheatography.com/davechild/cheat-sheets/regular-expressions/

CheatSheet



Cheatography

The complete RegEx Cheat Sheet
by doublehelix via cheatography.com/27391/cs/7932/

Anchors (boundaries)

^	Start of string or line
$	End of string or line
\A	Start of input (ignores 'm' flag)
\Z	End of input (ignores 'm' flag)
\G	End of the previous match
\b	Word boundary (any position proceeded or followed - but not both - by a letter, digit or underscore)
\B	Non-word boundary

Character and Sets

\w	Word	[a-zA-Z0-9_]
\W	Non-word	[^a-zA-Z0-9_]
\d	Digit	[0-9]
\D	Non-digit	
\s	Whitespace (Form-feed, tab, vertical-tab, new line, carriage return and space)	[\f\t\x0b\n\r]
\S	Non-whitespace	
\x	Hexadecimal digit	\x00=null; \x0d=\r; [\x61-\x7a]=[a-z]
\O	Octal digit	
.	Any character (except new line \n)	

Groups

(...)	Capture group - captures a set of characters for a later expression
(?:...)	Non-capture group - groups an expression but does not capture. e.g. /((?:foo|fu)bar)/ matches "foobar" or "fubar" without "foo" or "fu" appearing as a captured subpattern
(?=...)	Lookahead - match on the characters following. e.g. /ab(?=c)/ match "ab" only when followed by "c"
(?!...)	Negative lookahead - match on characters that aren't following. e.g. /ab(?!c)/ match "ab" only when NOT followed by "c"
(?<=...)	Positive look-behind assertion. e.g. /(?<=foo)bar/ matches "bar" when preceded by "foo"
(?<!...)	Negative look-behind assertion. e.g. /(?<!foo)bar/ - matches "bar" when not preceded by "foo"
(?#...)	Comment e.g. (?# This comment is ignored entirely)

Unicode character support

\x0000-\xFFFF	Unicode hexadecimal character set
\x00-\xFF	ASCII hexadecimal character set
\cA-\cZ	Control characters

Unicode is not fully supported on all platforms. JavaScript prior to ES6 for example allows ASCII hex but not full Unicode hex.

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Quantifiers

*	Zero or more
+	One or more
?	Zero or One (i.e. optional)
{n}	Exactly 'n' (any number)
{n,}	Minimum ('n' or more)
{n,m}	Range ('n' or more, but less or equal to 'm')

Flags (expression modifiers)

/m	Milti-line. (Makes ^ and $ match the start and end of a line respectively)
/s	Treat input as a single line. (Makes '.' match new lines as well)
/i	Case insensitive pattern matching.
/g	Global matching. (Don't stop after first match in a replacement function)
/x	Extended matching. (disregard white-space not explicitly escaped, and allow comments starting with #)

Escape Characters

In regular expressions, the following characters have special meaning and must be escaped: `^ $ [ { ( ) < > . * \ + | ?`
Additionally the hyphen (-) and close square bracket (]) must be escaped when in an expression set (`[...]`).
e.g. `/\(\d{3}\) \d{4}[\- ]\d{4}/` matches "(nnn) nnnn-nnnn" or "(nnn) nnnn nnnn" (where n is a numeric digit).

By **doublehelix**
cheatography.com/doublehelix/

Published 30th January, 2017.
Last updated 31st January, 2017.
Page 1 of 1.

How to use Regular Expressions in Utilities

```
static boolean onlyContainsNumbers(String text)
{
    return (text.matches("[0-9]+"));

}
```

Regular expression pattern – this means ‘any number of digits’

Topics list

1. Regular expressions
2. Methods with int return type
3. Boolean methods
4. **Utilities** class
5. **equals()** method

Example 1. Methods with **int** return type

```
public int addTwoNumbers1(int num1, int num2) {
      int total;
      total = num1 + num2;
      return total;
}
```

Called as

```
int myVar = addTwoNumbers1(4,5);
System.out.println("The result is " + myVar);
```

And this is printed to the console

```
The result is 9
```

Example 2. Methods with **int** return type

```
public int addTwoNumbers1(int num1, int num2) {
    return (num1 + num2);
}
```

Returns the value directly without use of temporary variable

Called as

```
int myVar = addTwoNumbers1(4,5);
System.out.println("The result is " + myVar);
```

And this is printed to the console

The result is 9

Example 3. Methods with **int** return type – calling the function directly

```
public int addTwoNumbers1(int num1, int num2) {
    return (num1 + num2);
}
```

Call the method directly in the print statement

```
System.out.println("The result is " + addTwoNumbers1(4,5));
```

And this is printed to the console

The result is 9

Exercise 1 . Methods with **int** return type

Exercise 1. Write a method that takes in three integer parameters and returns the sum of those values.

```
public int addThreeNumbers(int num1, int num2 ,int num3) {
    return (num1 + num2 + num3);
}
```

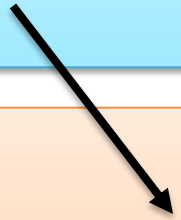
Topics list

1. Regular expressions
2. Methods with int return type
3. Boolean methods
4. **Utilities** class
5. **equals()** method

Example 4. Methods with **boolean** return type.

Write a method that returns true if t (integer) parameter value is greater than 10 and false otherwise.

```
public boolean overTen1(int i) {
      if (i>10) return true;
      else return false;
}
```



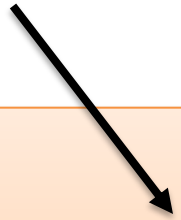
Called as

```
int x = 17;
if (overTen1(x) == true )
  System.out.println("This value : " + x + "is greater than 10" );
else
  System.out.println("This value : " + x + "is not greater than 10"
);
```

Example 5. Methods with **boolean** return type. (Direct use of condition)

Write a method that returns true if an (integer) parameter value is greater than 10 and false otherwise.

```
public static boolean overTen2(int i) {
  return (i>10);    // returns the value of the condition directly
}
```



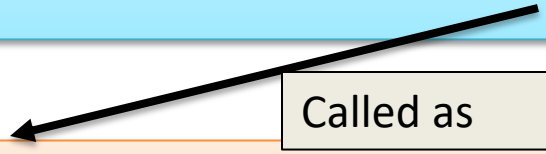
Called as

```
int x = 17;
if (overTen1(x) == true )
  System.out.println("This value : " + x + "is greater than 10" );
else
  System.out.println("This value : " + x + "is not greater than 10"
);
```

Example 6. Methods with **boolean** return type. (Direct use of condition in call of method)

Write a method that returns true if an (integer) parameter value is greater than 10 and false otherwise.

```
public static boolean overTen2(int i) {
  return (i>10);    // returns the value of the condition directly
}
```

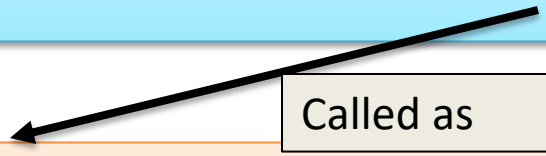



```
if (overTen1(x))   //calling method directly in condition
  System.out.println("This value : " + x + "is greater than 10" );
else
  System.out.println("This value : " + x + "is not greater than
10" );
}
```

Example 7. Methods with **boolean** return type. (Use of NOT in call)

Write a method that returns true if an (integer) parameter value is greater than 10 and false otherwise.

```
public static boolean overTen2(int i) {
  return (i>10);    // returns the value of the condition directly
}
```




```
if (!overTen1(x))   //calling method directly in condition
  System.out.println("This value : " + x + "is NOT greater than 10" );
else
  System.out.println("This value : " + x + "is greater than 10" );
}
```

If you are testing this code..

```
public static void main(String[] args) {
//   call the methods
}
```

Add the static modifier to all the methods

```
public static int addTwoNumbers2(int num1, int num2) {
   return (num1 + num2);
}
```

Source: Reas & Fry (2014)

Topics list

1. Regular expressions
2. Methods with int return type
3. Boolean methods
4. **Utilities** class
5. **equals()** method

Utilities Class

- Utilities Class will contain a collection of ‘useful’, reusable methods.
- These methods will not depend on any fields of the class, and solely on the parameters of the methods.
- Because there are no fields, there is no need to have objects of this class. We use the methods directly from the class
- To do this, we
  - Add the static modifier to the methods
  - Call the methods using the Class name, not the object name.

Utilities Class

- To do this, we
  - Add the static modifier to the methods

```
public static boolean overTen2(int i) {
   return (i>10);
}
```

  - Call the methods using the Class name, not the object name.

```
If (Utilities.overTen2(6) )  {
     System.out.println (“Over ten”);
else
     System.out.println (“Not over ten”);
```

Topics list

1. Regular expressions
2. Methods with int return type
3. Boolean methods
4. **Utilities** class
5. **equals()** method

equals() method

- What do we mean by two objects being equal?

- If we wish to check that two objects contain the same values (as opposed to being stored at the same address), we can use the equals() method.
- Similar to toString(), there is a high-level version of equals() written, but we can make this more specific by writing this more locally in the class (thus overloading the method).

- So you an decide exactly what it means for two objects to be equal.

equals() method

```
public class SimpleClass {
   int x;
   int y;
   SimpleClass(int x, int y) {   //constructor
      this.x = x;
      this.y = y;
   }
   int getX(){  return x;}
   int getY(){  return y;}

   public boolean equals(SimpleClass otherSimpleClassObject) {
      return (this.x == otherSimpleClassObject.getX() &&
                  this.y == otherSimpleClassObject.getY() );
   }
}
```

equals() method

- We define equals with one parameter
- - an object of the same class

- For two SimpleClass objects to be equal, each of the their integer fields of the first object must be equal to the other object's integer fields.

```
    public boolean equals(SimpleClass otherSimpleClassObject) {
        return (this.x == otherSimpleClassObject.getX() &&
                    this.y == otherSimpleClassObject.getY() );
    }
}
```

Using the equals() method

```
   public boolean equals(SimpleClass otherSimpleClassObject) {
      return (this.x == otherSimpleClassObject.getX() &&
                  this.y == otherSimpleClassObject.getY() );
   }
}
```

```
{  //code snippet
SimpleClass object1, object 2;
object1 = new SimpleClass(4,5);  //different objects but the same values
object2 =  new SimpleClass(4,5);

If (object1.equals(object2) )
  System.out.println("objects have equal values");
}
```

Questions?