* Can use a string to match a pattern
  + String pattern = “fish”;
  + String wordToMatch = “fish”;
  + String wordToNotMatch = “dish”;
  + wordToMatch.matches(pattern) is true
  + wordToNotMatch.matches(pattern) is false
* But can also make things more complicated than this!
* Regex special characters:
  + [...]: match any particular character to what is in the brackets, creating a character class
    - [aeiou], [a-zA-Z], [0-9]
    - Only characters that are special characters in the brackets are ^,-,\
    - Can still use \ to escape these special characters
  + ^: inside brackets at the beginning, match anything but what is the class
    - [^AB]
  + +: match one or more of the character preceding it
    - A+B, AAB but not B
  + ?: match 0 or 1 of the character preceding it
    - A?B, AB and B but not AAB
  + \*: match 0 or more of the character preceding it
    - A\*B, B and AB but not ABB
    - (AB)\*, AB but not B
  + {n}: match exactly n copies of the character preceding it
    - [A-Z]{3}
  + {n, m}: match at least n but at most m copies of the character preceding it
    - [A-Z]{1,2}
  + .: match any character (except a line terminator)
  + (...): indicate a group to be “captured”, more information below
  + To match the character ?, Regex will think it’s a special character, so we need to escape it with \
  + Practice problems with Graph.java
* Character classes:
  + \d: match any digit
  + \D: match any non-digit
  + \n: match a newline
  + \s: match a whitespace
  + \w: match a word character (a-z, A-Z, 0-9)
  + \W: match a non-word character
* Pattern
  + Pattern.compile(String), creates a pattern of String
  + Pattern.compile(String, FLAG): creates a pattern of String obeying the provided flag
    - Potential flag: Pattern.CASE\_INSENSITIVE
* Matcher
  + Matcher m1 = (Pattern object).matcher(phrase to match);
  + m1.matches() will return if the phrase to match matches the Pattern object
  + m1.find() will return if there is a substring in phrase to match that matches the Pattern object
    - Will go through the phrase to match from right to left, once a section matches, matcher will stop at that point, next time find is called it will resume from that place and keep looking
* Capturing
  + Sometimes when we match things, but when we match things we want to do something with them!
    - What if we have a text file with a bunch of emails and we want to compile all the emails into a file
  + Use parentheses to indicate a group that you want to “capture” and return
  + (Matcher object).group(index)
    - Index 0: the whole phrase that matched
    - Index 1: the first parentheses-ed group
    - Index 2: the second parentheses-ed group
* Greedy vs. Reluctant
  + lo000ol hellooooo hahahahaha
  + What if we want to match lo000ol? We can match l.+l
  + . + and ? are all greedy operations (will match as much as possible!), can put ? after any of these operators to make them match as little as possible