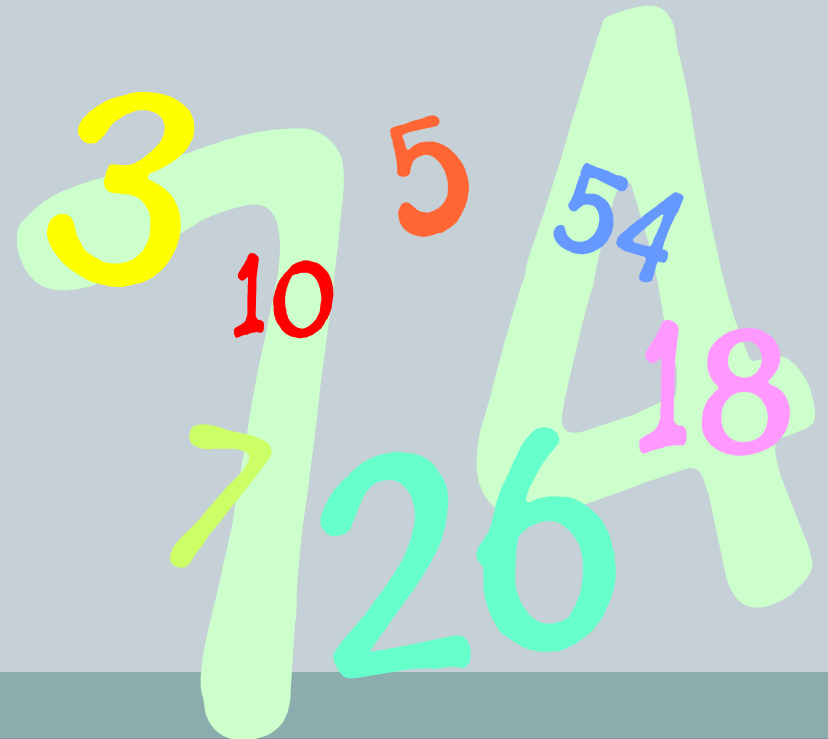


Enumerations



Outline

- **Avoiding magic numbers**
 - Variables takes on a small set of values
 - Use descriptive names instead of literal values
 - Java enumerations
 - Using in a switch statement



Variables from a Set of Values

- Magic numbers
 - Where did the value come from?
 - What does it mean?
 - What if you mistype the number?
 - What if you want to keep value in specific range?



```
int direction = 0;

...

if ((direction == 1) || (direction == 3) ||
    (direction == 5) || (direction == 7))
{ /* TBD */ }

direction = 0;           // Valid???
direction = 8;           // Valid???
direction = -2729;       // Valid???
```

Variables from a Set of Values

- **Solution 1: Create final constants**

- Descriptive names means everybody can read
- Bugs less likely, typo in name = compile error
- Keyword `final` ensures nobody can change value

```
final int NORTH      = 0;  
final int NORTHEAST  = 1;  
final int EAST       = 2;  
final int SOUTHEAST  = 3;  
final int SOUTH      = 4;  
final int SOUTHWEST  = 5;  
final int WEST       = 6;  
final int NORTHWEST  = 7;
```

```
int direction = NORTH;
```

```
...
```

```
if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||  
    (direction == SOUTHWEST) || (direction == NORTHWEST))  
{ // TBD }
```



Constants not Always Ideal

```
final int NORTH      = 0;  
final int NORTHEAST  = 1;  
final int EAST       = 2;  
final int SOUTHEAST  = 3;  
final int SOUTH      = 4;  
final int SOUTHWEST  = 5;  
final int WEST       = 6;  
final int NORTHWEST  = 7;
```

Problem 1: Tedious to type. Also easy to mess up, e.g. setting two constants to same value.

```
int direction = 0;
```

Problem 2: Not forced to use the friendly names.

```
...
```

```
if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||  
    (direction == SOUTHWEST) || (direction == NORTHWEST))  
{/* TBD */}
```

```
direction = 0;           // Valid???  
direction = 8;           // Valid???  
direction = -2729;       // Valid???
```

Problem 3: Not forced to stay in range. What does it mean to be 8 or -2729 if you are a compass direction?

Enumerations

- A better solution: **enumerations**
 - Specifies exact set of friendly names
 - Compiler ensures we stay in range

Easiest to declare outside class. Semicolon is optional.

```
public enum Compass {NORTH, NORTHEAST, EAST, SOUTHEAST,
                    SOUTH, SOUTHWEST, WEST, NORTHWEST}
```

```
public class CompassTest
```

```
{
```

```
    public static void main(String [] args)
```

```
    {
```

```
        Compass direction = Compass.NORTH;
```

```
        if ((direction == Compass.NORTHEAST) ||
```

```
            (direction == Compass.SOUTHEAST) ||
```

```
            (direction == Compass.SOUTHWEST) ||
```

```
            (direction == Compass.NORTHWEST))
```

```
        { /* TBD */ }
```

```
        direction = 0;
```

```
    }
```

```
}
```

Now a compile error.
Way to watch our back compiler!

Enumeration Tricks

- Enumerations

- Actually objects with a few handy methods:

<code>toString()</code>	Print out friendly name corresponding to value of variable
<code>values()</code>	Returns array of all the possible values type can take on

```
public enum Compass {NORTH, NORTHEAST, EAST, SOUTHEAST,
                    SOUTH, SOUTHWEST, WEST, NORTHWEST}
```

```
...
```

```
for (Compass d : direction.values())
```

```
{
```

```
    if (checkMonster(hero, d))
```

```
        System.out.println("You see a monster to the " +
                           d.toString());
```

```
}
```

for-each loop, goes over
all values of the
enumeration

switch Statement

```
Compass direction = Compass.NORTH;
```

```
switch (direction)
{
    case NORTH:
        hero.move(0, 1);
        System.out.println("Walking north");
        break;
    case SOUTH:
        hero.move(0, -1);
        System.out.println("Walking south");
        break;
    case EAST:
        hero.move(1, 0);
        System.out.println("Walking east");
        break;
    case WEST:
        hero.move(-1, 0);
        System.out.println("Walking west");
        break;
}
```

Note: normally you need "Compass.", but not in switch case since Java knows type

You can have as many statements as you want between case and break.

Summary

- **Avoiding magic numbers**
 - Variables takes on a small set of values
 - Use descriptive names instead of literal values
 - Java enumerations
 - Using in a switch statement



Regular Expressions

having nothing driving regulating growing pausing bringing stepping knocking not
hing surprising leaning looking striving pacing Nothing loitering falling enchan
ting reaching overlapping receiving meaning going something something taking goi
ng being broiling thing putting lording making anything knowing paying paying be
ing paying being considering having whaling going wh
ling being performing cajoling resulting discriminat

earning reachi
owing shoulder
thing flying h
ng creaking lo
ting chatterin
ng straggling
ing painting s
sweeping deat
ting dark-look
uminating ador
oking nothing
leeping making
ying dusting p
g standing loc



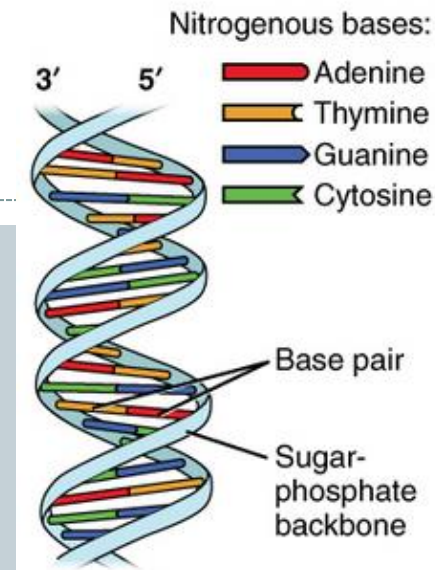
TTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Outline

- Regular expressions

- Convenient notation to detect if a string is in a set
 - ✦ **Built-in** to many modern programming languages
 - ✦ Usually **easier** than writing custom string parsing code
- Very powerful
 - ✦ But still some things it can't do:
 - e.g. Recognize all bit strings with equal number of 0's and 1's
- Well-supported in Java String class:
 - ✦ Test if a String **matches** an RE
 - ✦ **Split** a String based on an RE
 - ✦ **Find-and-replace** based on an RE

Pattern Matching



- Is a given string in a set of strings?
 - Example from genomics:
 - ✦ DNA: sequence of **nucleotides: C, G, A or T**
 - ✦ Fragile X syndrome:
 - Common cause of mental disability
 - Human genome contains **triplet repeats of CGG or AGG**, bracketed by **GCG at the beginning** and **CTG at the end**
 - Number of **repeats is variable**, correlated with syndrome

Set of strings: "all strings of G, C, T, A having some occurrence of GCG followed by any number of CGG or AGG triplets, followed by CTG"

Question: Is the following string in this set of strings?

GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Pattern Matching

- Is a given string in a set of strings?

- Example from genomics:

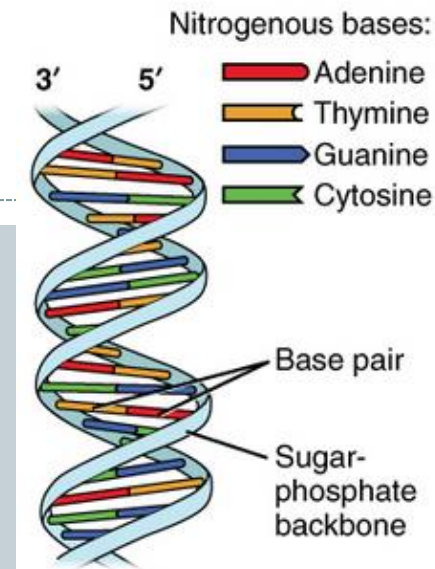
- ✦ DNA: sequence of **nucleotides: C, G, A or T**

- ✦ Fragile X syndrome:

- Common cause of mental disability

- Human genome contains **triplet repeats of CGG or AGG**, bracketed by **GCG at the beginning** and **CTG at the end**

- Number of **repeats is variable**, correlated with syndrome



Set of strings: "all strings of G, C, T, A having some occurrence of GCG followed by any number of CGG or AGG triplets, followed by CTG"

Question: Is the following string in this set of strings?

GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTG**GCGCGGAGGCGGCTG**GCGCGGAGGCTG

Answer: Yes

A Pattern Matching Application

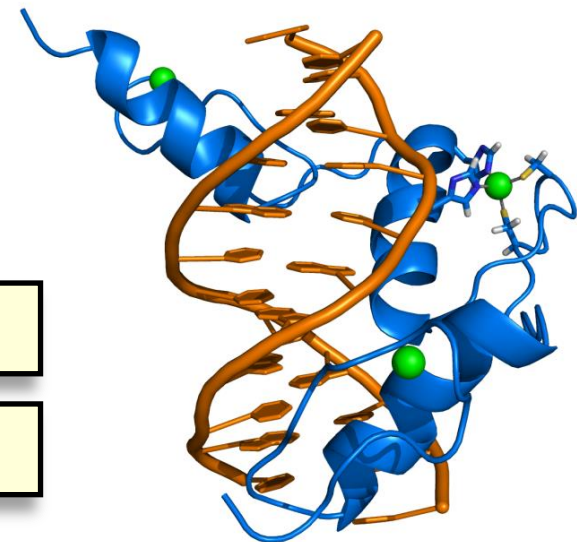
- PROSITE

- Huge database of protein families and domains
- How to identify the C₂H₂-type zinc finger domain?

1. C
2. Between 2 and 4 amino acids
3. C
4. 3 amino acids
5. One of the following amino acids: LIVMFYWCX
6. 8 amino acids
7. H
8. Between 3 and 5 amino acids
9. H

CAASC^CGGPYACGGWAGYHAGWH

CAASC^CGGP^YACGGWAGY^HAGW^H



Another Pattern Matching Application

- What are people saying about Keith on twitter?
 - Collecting ~1% of tweets since 2010
 - ✦ Currently ~~737 GB~~ 1.6 TB compressed!
 - Find all tweets starting with "keith is"
 - How many?
 - ✦ Out of 54 M "sensible" English tweets: 91



```
keith is so awesome
keith is fun
keith is beautiful
keith is sweet
keith is the king of this here compound
keith is great
keith is always there when i need to laugh
keith is the bestest
keith is awesome
keith is so sweet
keith is hilarious
keith is such a kind soul and life saver
...
```

Even More Applications

- Test if a string matches some pattern
 - Process natural language
 - Scan for virus signatures
 - Access information in digital libraries
 - Find-and-replace in word processors
 - Filter text (spam, NetNanny, ads, Carnivore, malware)
 - Validate text fields (dates, email, URL, credit card)
- Parse text files
 - Compile a Java program
 - Crawl and index the web
 - Create Java documentation from Javadoc comments

Regular Expressions

- Regular expressions (REs)
 - Notation that specifies a **set of strings**

operation	regular expression	matches	does not match
<i>concatenation</i>	aabaab	aabaab	<i>every other string</i>
<i>wildcard</i> .	.u.u.u.	cumulus jugulum	succubus tumultuous
<i>union</i> 	aa baab	aa baab	<i>every other string</i>
<i>closure / star</i> (0 or more) *	ab*a	aa abbba	ab ababa
<i>parentheses</i> ()	a(a b)aab	aaaab abaab	<i>every other string</i>
	(ab)*a	a ababababa	aa abbba

Regular Expressions

- Regular expressions (REs)
 - Notation is **surprisingly expressive**

regular expression	matches	does not match
<code>.*spb.*</code> <i>contains the trigraph spb</i>	raspberry crispbread	subspace subspecies
<code>a* (a*ba*ba*ba*)*</code> <i>multiple of three b's</i>	bbb aaa bbbaababbaa	b bb baabbbbaa
<code>.*0....</code> <i>fifth to last digit is 0</i>	1000234 98701234	111111111 403982772
<code>gcg(cgg agg)*ctg</code> <i>fragile X syndrome indicator</i>	gcgctg gcgcggctg gcgcggaggctg	gcgcgg cggcggcggctg gcgcaggctg

Regular Expressions

- Regular expressions (REs)
 - A standard programmer's tool
 - ✦ Built into many languages: Java, Perl, Unix, Python, ...
 - Additional convenience operations:
 - ✦ e.g. `[a-e]+` shorthand for `(a|b|c|d|e)(a|b|c|d|e)*`
 - ✦ e.g. `\s` is shorthand for any whitespace character

operation	regular expression	matches	does not match
<i>one or more</i> +	<code>a(bc)+de</code>	abcde abcbcbde	ade bcde
<i>character class</i> []	<code>[A-Za-z][a-z]*</code>	lowercase Capitalized	camelCase 4illegal
<i>exactly k, between k and j</i> {k}, {k,j}	<code>[0-9]{5}-[0-9]{4}</code>	08540-1321 19072-5541	111111111 166-54-1111
<i>negation</i> ^	<code>[^aeiou]{5,6}</code>	rhythm synch	decade rhythms

Pattern Matching Application

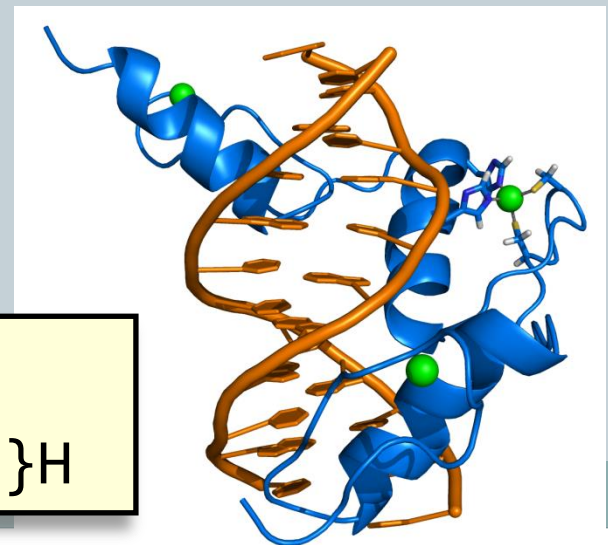
- PROSITE

- Huge database of protein families and domains
- Identify the C₂H₂-type zinc finger domain, **how???**

1. **C**
2. Between 2 and 4 amino acids
3. **C**
4. 3 more amino acids
5. One of the following amino acids: LIVMFYWCX
6. 8 more amino acids
7. **H**
8. Between 3 and 5 more amino acids
9. **H**

Use a regular expression!

`C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H`



Regular Expressions in Java

- Helps match and split up strings
 - Built-in to Java String class methods
 - Note: escape \ in regular expression with \\

```
public class String
```

```
boolean matches(String re)           // Does this String match the given re?  
  
String replaceAll(String re, String str) // Replace all occurrences of re with str  
  
String replaceFirst(String re, String str) // Replace first occurrence of re with str  
  
String [] split(String re)           // Split string around matches of re
```

```
String [] cols = line.split("\\s+");
```

Regular expression that matches 1 or more whitespace characters.
NOTE the escaped backslash!

Parsing Data into Columns

- Goal: Compute average of a line of numbers
- Problem: Numbers per line is unknown

```
10 20 30
40.0

50 60.12
70 80 90 100 110 120 130 140
1.2 2.3 3.4
```

avgnums.txt

```
% java AvgPerLine avgnums.txt
20.0
40.0
55.06
105.0
2.3000000000000003
```

AvgPerLine Implementation

```
public class AvgPerLine
{
    public static void main(String [] args)
    {
        Scanner scan = new Scanner(System.in);
        while (scan.hasNext())
        {
            String line = scan.nextLine();
            String [] cols = line.split("\\s+");
            if ((cols.length > 0) && (cols[0].length() > 0))
            {
                double total = 0.0;
                for (String col : cols)
                    total += Double.parseDouble(col);
                System.out.println(total / cols.length);
            }
        }
    }
}
```

Read in entire line of
text

Split on whitespace



Regular Expression Example

- **Goal:** Display all words in a file ending -ing


```
% java GerundFinder mobydict.txt
```

```
having nothing driving regulating growing pausing bringing stepping knocking not  
hing surprising leaning looking striving pacing Nothing loitering falling enchan  
ting reaching overlapping receiving meaning going something something taking goi  
ng being broiling thing putting lording making anything knowing paying paying be  
ing paying being considering having whaling going whaling something "Whaling wha  
ling being performing cajoling resulting discriminating overwhelming attending e  
verlasting ignoring whaling Quitting learning reaching following whaling somethi  
ng everything monopolizing having following shouldering comparing halting pausin  
g tinkling stopping moving proceeding thing flying hearing sitting beating weepi  
ng wailing teeth-gnashing backing Moving creaking looking swinging painting repr  
esenting swinging leaning howling toasting chattering shaking everlasting making  
holding being blubbering going Entering straggling reminding painting understan  
ding throwing something hovering floating painting something weltering purposing  
spring impaling glittering resembling sweeping death-harvesting horrifying whal  
ing sojourning Crossing howling Projecting dark-looking goggling cheating enteri  
ng examining telling tapping sharing ruminating adorning stooping working trying  
adjoining Nothing winding scalding looking nothing knowing evening rioting Star  
ting offing tramping capering making sleeping making dazzling seeming sleeping s  
leeping being getting going feeling saying dusting planing grinning spraining pl  
aning gathering throwing yoking leaving standing looking seeing spending cherish
```



GerundFinder

```
public class GerundFinder
{
    public static void main(String [] args)
    {
        Scanner scan = new Scanner(System.in);
        while scan.hasNext()
        {
            String word = scan.next();
            if (word.matches("."+ing"))
                System.out.print(word + " ");
        }
        System.out.println();
    }
}
```

Read in next whitespace
separated chunk of text



1 or more characters
followed by "ing"



Regular Expression Quick Reference

Construct	Matches
.	Any character
\d	A digit: 0-9
\s	A whitespace character
\w	A word character: a-z A-Z 0-9 _
\D	A non-digit (anything except 0-9)
\S	A non-whitespace character
\W	A non-word character

Classes	Matches
[abc]	Character a, b or c
[^abc]	Any character except a, b, or c
[a-z]	Characters a, b, c, ..., z
[A-Z]	Characters A, B, C, ..., Z
[a-zA-Z]	Characters a, A, b, B, ..., z, Z

Expression	Example matches
...	cat, sat, mat, ...
c..	cat, cow, cut, ...
[abc]at	aat, bat, cat
[abc]+z	az, bz, cz, aaz, abz, bcz, bbacz, ...
[0-9]{5}	12345, 59701, 01234, ...
\d\d\d\d	1980, 2005, 9999, ...

Quantifier	Matches
*	Zero or more occurrences
+	One or more occurrences
?	Zero or one occurrences
{n}	Exactly n occurrences
{n,}	At least n occurrences
{n,m}	Between n and m occurrences inclusive

Summary



- Regular expressions
 - Convenient notation to detect if a string is in a set
 - ✦ **Built-in** to many modern programming languages
 - ✦ Usually **easier** than writing custom string parsing code
 - Very powerful
 - ✦ But still some things it can't do:
 - e.g. Recognize all bit strings with equal number of 0's and 1's
 - Well-supported in Java String class:
 - ✦ Test if a String **matches** an RE
 - ✦ **Split** a String based on an RE
 - ✦ **Find-and-replace** based on an RE