CS 112 Introduction to Programming

Strings Processing: Boolean Methods
Strings vs Chars
Encryption/Decryption

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Admin.

- PS4 continuation
- PS5 posted on the Assignments page
  - Monte Carlo is fully doable
  - TaskMan needs some background on Arrays which we cover today
- Supporting materials of Exam 1 are on the Schedule page
Recap: nested if/else

- Nested if/else is a common structure
  - Mutually exclusive test conditions
    => Reduces # of comparisons
  - Non-mutual exclusive test conditions
    => Achieves mutual exclusion
Recap: Conditional Shortcuts

- **Java switch statement** is a shortcut allowing clear listing of multiple choices in some settings.

  ```java
  switch (expression) {
  case value1:
    statement-list1
  case value2:
    statement-list2
  case value3:
    statement-list3
  default:
    ...
  }
  ```

- **Java conditional operator** that evaluates a boolean expression to determine which one of the two expressions should be evaluated.

  ```java
  condition ? expression1 : expression2
  ```
Roadmap: String Processing

- Access methods, e.g.,
  - length, substring, charAt, indexOf

- String generator methods (string->string), e.g.
  - toLowerCase, toUpperCase, replace, split (string->strings)

- Conversion from String, e.g.,
  - Integer.parseInt, Double.parseDouble

- Input processing, e.g.,
  - Scanner.next, Scanner.nextLine

- Output formatting: format string
  - System.out.printf, String.format

- Boolean methods, e.g.,
  - equals, equalsIgnoreCase, startsWith, endsWith, contains, matches
Split a String

- Some common approaches
  - Use `indexOf` to find the location and then `substring` to extract a sub string
  - Use `scanner` to extract tokens
    ```java
    String s = "Years 2017 2018 will be great!";
    Scanner scan = new Scanner( s );
    String word = scan.next(); // skip Year
    int year1 = scan.nextInt(); // 2017
    word = scan.next();
    int year2 = Integer.parseInt( word ); // 2018
    ```
  - Use `split` to split a string into multiple strings
    (more later; part of ps5 alternative design)
Exercise

- Given the following string:

```java
// index 0123456789012345678901
String book = "Building Java Programs";
```

- How would you extract the word "Building" from `book`?
- More generally, how to extract the first word of a string?
Exercise

- Write a program to convert your “boring” name to a more exciting "gangsta name."
  - last initial
  - Diddy
  - first name (all caps)
  - -izzle

Example Output:
Type your name, playa: **Marge Simpson**
Your gangsta name is "S. Diddy MARGE-izzle"
// This program prints your "gangsta" name.
import java.util.*;

public class GangstaName {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type your name, playa: ");
        String name = console.nextLine();

        // split name into first/last name and initials
        String first = name.substring(0, name.indexOf(" "));
        first = first.toUpperCase();
        String last = name.substring(name.indexOf(" ") + 1);
        String lInitial = last.substring(0, 1);

        System.out.println("Your gangsta name is \\
                         " + lInitial + ", Diddy " + first + "-izzle");
    }
}
## String Boolean (Pattern-Matching) Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals(str)</code></td>
<td>whether two strings contain the same characters</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(str)</code></td>
<td>whether two strings contain the same characters, ignoring upper vs. lower case</td>
</tr>
<tr>
<td><code>startsWith(str)</code></td>
<td>whether one contains other's characters at start</td>
</tr>
<tr>
<td><code>endsWith(str)</code></td>
<td>whether one contains other's characters at end</td>
</tr>
<tr>
<td><code>contains(str)</code></td>
<td>whether the given string is found within this one</td>
</tr>
<tr>
<td><code>matches(regExp)</code></td>
<td>whether the string matches a regular expression</td>
</tr>
</tbody>
</table>

```java
Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String name = console.nextLine();

if (name.startsWith("Prof. Dr.")) {
    System.out.println("Are you from Germany?");
} else if (name.endsWith("Esq.")) {
    System.out.println("And I am Ted 'Theodore' Logan!");
}
```
Exercise: Word Rhyme

- Prompt the user for two words and report whether they:
  - "rhyme" (end with the same last two letters)
  - alliterate (begin with the same letter)

- Example output: (run #1)
  Type two words: car STAR
  They rhyme!

(run #2)
Type two words: bare bear
They alliterate!

(run #3)
Type two words: sell shell
They rhyme and alliterate!

(run #4)
Type two words: extra strawberry
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("Type two words: ");
    String word1 = console.next().toLowerCase();
    String word2 = console.next().toLowerCase();

    boolean isR = rhyme(word1, word2);
    boolean isA = alliterate(word1, word2);

    // output
}

// Returns true if s1 and s2 end with the same two letters.
public static boolean rhyme(String s1, String s2) {
    return s2.length() >= 2 && s1.endsWith(s2.substring(s2.length() - 2));
}

// Returns true if s1 and s2 start with the same letter.
public static boolean alliterate(String s1, String s2) {
    return s2.length() >= 1 && s1.startsWith(s2.substring(0, 1));
}
Offline Exercise/Read: Regular Expression Match (See Backup Slides)
Outline

- Admin and recap
- Strings
  - String access, modify, parse, format
  - String boolean methods
  - String vs char
String **charAt** method

- **The chars in a String can be accessed using the charAt method.**

```java
String food = "cookie";
char firstLetter = food.charAt(0); // 'c'
System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- **You can use a for loop to print or examine each character.**

```java
String major = "CS";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

**Output:**

```
C
S
```

StringExamples.java stringChars
Char internal stores the index of a char

Using `char` in arithmetic expressions will cause `char` to be converted to an `int`

To convert an `int` into the equivalent `char`, type-cast it.

```
(char) ('a' + 2) is 'c'
```

```java
for (char c = 'a'; c <= 'z'; c = (char)(c+1) ) {
    System.out.print(c);
}
```

```java
for (char c = 'a'; c <= 'z'; c ++ ) {
    System.out.print(c);
}
```
char vs. String

- "h" is a String
  'h' is a char   (the two behave differently)

- String is an object; it contains methods
  
  ```java
  String s = "h";
  s = s.toUpperCase();  // 'H'
  int len = s.length();  // 1
  char first = s.charAt(0);  // 'H'
  ```

- char is primitive; you can't call methods on it
  
  ```java
  char c = 'h';
  c = c.toUpperCase();  // ERROR
  ```
Consistent with primitive numerical types, you can compare char values using relational operators (ordering is alphabetical order):

'a' < 'b' and 'X' == 'X' and 'Q' != 'q'

```java
String word = console.next();
char ch = word.charAt(word.length() - 1);
if (ch == 's') {
    System.out.println(word + " is plural.");
}

ch = word.charAt(0);
if (‘a’ <= ch && ch <= ‘z’) {
    System.out.println(word + " starts w/ a letter");
}
```

StringExamples.java charCompare
**char vs. String (Comparison)**

- `<, <=, >, >=` are not defined on objects and hence you cannot write `str1 <= str2`.
- `!=` and `==` are defined on all objects, but they compare *references* (seen later), not values. So `==` often gives *false* even when two *Strings* have the same letters.

```java
Scanner console = new Scanner(System.in);
String name = console.next(); // user input Barney

if (name == "Barney") {
    System.out.println("I love you, you love me,'");
    System.out.println("We're a happy family!'");
}
```
**char vs. String (Comparison)**

- **String comparisons based on content use the method** `equals`.

```java
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- **String defines** `compareTo(anotherString)` **to compare the lexicographic order of two strings.**
# char vs. String

<table>
<thead>
<tr>
<th></th>
<th>String</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Store</strong></td>
<td>An object w/ a sequence of chars</td>
<td>A primitive value storing a unicode index</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td>.length, .toUpperCase, ...</td>
<td>No methods</td>
</tr>
<tr>
<td><code>==, !=</code></td>
<td>Compare reference, not value</td>
<td>Compare value</td>
</tr>
<tr>
<td><code>&lt;, &lt;=, &gt;, &gt;=</code></td>
<td>No defined. Use compareTo(anotherString) method</td>
<td>Lexicographic order</td>
</tr>
<tr>
<td>.equals method</td>
<td>Compare content</td>
<td>Not defined</td>
</tr>
</tbody>
</table>

Caesar Cipher

If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.

—Suetonius, Life of Julius Caesar 56
Caesar Cipher

Plaintext

attack said zeus
προσβάλλω

Ciphertext

dwwdfn vdlg chxv
στζφείξξό
Additional Caesar-Style Cipher

- **Augustus**
  - Right shift 1 without rotate, with AA for Z

- **Mezuzah**
  - Shift of one to encrypt the names of God

- **Vigenere (a variant) was used by the Confederacy during Civil War**

- **Mafia boss Provenzano (2006): A (4), ...**
Why Programming?

Dear Cleopatra,

I love you as big as the Aegean sea, as high as Mount Olympus, as many as the stars of Hyades, Orion, Sirius, and Ursa Major combined... My soul, heart, body and kingdom are yours for the taking. When I think of you, Cleo, my heart beats like that of a gladiator facing death at the claws of a rabid tiger...
## Caesar Cipher Encoding (Design 1)

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orig</strong></td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>...</td>
<td>...</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
</tr>
<tr>
<td><strong>Shift (+3)</strong></td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cipher</strong></td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>122</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
int cipher = ch + key;
if (cipher > 'z') cipher -= 26;
ch = (char)cipher;
```

Q: What if key is 100?
Caesar Cipher: Encoding (Design 2)

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orig</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td><strong>Shift (+3)</strong></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cipher (% 26)</strong></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{int cipherInt} = (\text{origInt} + \text{key}) \mod 26
\]

Assumption: a-z mapped to 0 to 25.
How to map a char in a-z to 0 to 25? char – ‘a’
# Caesar Cipher Encoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>Shift (+3)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>Cipher (% 26)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{int chRelativeIndex} = \text{ch} - 'a'; \\
\text{int cipherRelInd} = (\text{chRelativeIndex} + \text{key}) \mod 26; \\
\text{ch} = (\text{char})((\text{cipherRelInd} + 'a'));
\]
Offline Exercise/Think: How to Decode? (See Backup Slides)
Extension: Encrypt a Text File

- **Goal:** Instead of reading text from terminal, user specifies a file name, which gives the content to be encrypted.
Create a File object to get info about a file on your drive.

You can use File’s exists method to test existence:

```java
File file = new File("data.txt");
if (!file.exists()) {
    // try a second input file as a backup
    System.out.println("data file not found!");
    file = new File("data2.txt");
}
```

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canRead()</td>
<td>returns whether file is able to be read</td>
</tr>
<tr>
<td>delete()</td>
<td>removes file from disk</td>
</tr>
<tr>
<td>exists()</td>
<td>whether this file exists on disk</td>
</tr>
<tr>
<td>getName()</td>
<td>returns file's name</td>
</tr>
<tr>
<td>length()</td>
<td>returns number of bytes in file</td>
</tr>
</tbody>
</table>
public class ReadFile {
    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(new File("data.txt"));
            encode(input, 3);
        } catch (FileNotFoundException e) {
            // print an error message
            System.out.println("File not found exception");
        } // end of catch
    } // end of main
}

class Caesar {
    public static String encode(String text, int key) {
        String result = "";
        for (int i = 0; i < text.length(); i++) {
            int charIndex = text.charAt(i);
            if (charIndex >= 'A' && charIndex <= 'Z') {
                charIndex = (charIndex - 'A' + key) % 26 + 'A';
            } else if (charIndex >= 'a' && charIndex <= 'z') {
                charIndex = (charIndex - 'a' + key) % 26 + 'a';
            }
            result += (char) charIndex;
        }
        return result;
    }
}

public static void encode(Scanner scan, int key) {
    for (int line = 0; line < NLINES; line++) {
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
    }
}
Exercise: Breaking Caesar Encryption

- Caesar encryption (or permutation ciphers in general) is easy to break because it does not change the frequency of letters.
Backup Slides
A regular expression defines a search pattern for strings.

Regular expressions can be used to search, edit and manipulate text.

The pattern defined by the regular expression may match one or several times or not at all for a given string.

http://www.vogella.com/tutorials/JavaRegularExpressions/article.html
# Regular Expression

<table>
<thead>
<tr>
<th>RegEx</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Matches any character</td>
</tr>
<tr>
<td>^regex</td>
<td>Finds regex that must match at the beginning of the line.</td>
</tr>
<tr>
<td>regex$</td>
<td>Must match at the end of the line.</td>
</tr>
<tr>
<td>[abc]</td>
<td>Set definition, can match the letter a, b or c.</td>
</tr>
<tr>
<td>[abc][vz]</td>
<td>Set definition, can match a or b or c followed by either v or z.</td>
</tr>
<tr>
<td>[^abc]</td>
<td>When a caret appears as the first character inside square brackets, it negates the pattern. This can match any character except a or b or c.</td>
</tr>
<tr>
<td>[a-d1-7]</td>
<td>Ranges: matches a letter between a and d and figures from 1 to 7, but not d1.</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>XZ</td>
<td>Finds X directly followed by Z.</td>
</tr>
<tr>
<td>$</td>
<td>Checks if a line end follows.</td>
</tr>
<tr>
<td>(Yale)</td>
<td>(MIT)</td>
</tr>
</tbody>
</table>
Examples

Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String name = console.nextLine();

name.startsWith("Prof. Dr.")

name.matches("^Prof\. Dr\.")

name.endsWith("Esq."")

name.matches("Esq\.$")
# Meta Chars

<table>
<thead>
<tr>
<th>Regex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>Any digit, short for [0-9]</td>
</tr>
<tr>
<td>\D</td>
<td>A non-digit, short for [^0-9]</td>
</tr>
<tr>
<td>\s</td>
<td>A whitespace character, short for [\t\n\x0b\r\f]</td>
</tr>
<tr>
<td>\S</td>
<td>A non-whitespace character, short for [^\s]</td>
</tr>
<tr>
<td>\w</td>
<td>A word character, short for [a-zA-Z_0-9]</td>
</tr>
<tr>
<td>\W</td>
<td>A non-word character [^\w]</td>
</tr>
<tr>
<td>\b</td>
<td>Matches a word boundary where a word character is [a-zA-Z0-9_]</td>
</tr>
</tbody>
</table>
## Quantifiers

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Occurs zero or more times, is short for {0,}</td>
<td>X* finds no or several letter X, .* finds any character sequence</td>
</tr>
<tr>
<td>+</td>
<td>Occurs one or more times, is short for {1,}</td>
<td>X+ - Finds one or several letter X</td>
</tr>
<tr>
<td>?</td>
<td>Occurs no or one times, ? is short for {0,1}.</td>
<td>X? finds no or exactly one letter X</td>
</tr>
<tr>
<td>{X}</td>
<td>Occurs X number of times, {} describes the order of the preceding liberal</td>
<td>\d{3} searches for three digits, {10} for any character sequence of length 10.</td>
</tr>
<tr>
<td>{X,Y}</td>
<td>Occurs between X and Y times,</td>
<td>\d{1,4} means \d must occur at least once and at a maximum of four.</td>
</tr>
<tr>
<td>*?</td>
<td>? after a quantifier makes it a reluctant quantifier. It tries to find the smallest match.</td>
<td></td>
</tr>
</tbody>
</table>
Example

Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String name = console.nextLine();

.. 

name.contains("Yale")

    name.matches(".*Yale.*");

String ssn = console.nextLine();
// how to validate it is correct ssn: 123-45-6789

    ssn.matches("\\d{3}-\\d{2}-\\d{4}");

String ph = console.nextLine();
// how to validate right ph# format: 432-6400 or 432 6400

    ph.matches("\\d{3}[- ]\\d{4}");
Practice: Detect Email Address
# Caesar Cipher: Decoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orig</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td><strong>Shift (+3)</strong></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cipher (% 26)</strong></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Back (-3)</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Range (+26%26)</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
int cipherInt = (origInt + key) % 26

int origInt = (cipherInt - key + 26) % 26

- A single encode method with parameter key
  - To encode
    - encode(key)
  - To decode
    - encode(26 - key)