Sierpinski Valentine

http://xkcd.com/543/

Happy Valentine's Day.

-xKCD
CS 112  Introduction to Programming

Switch; Text Processing

Yang (Richard) Yang
Computer Science Department
Yale University
308A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Admin.

- PS4 questions
Recap: if/else and Boolean

- Our “journey” of introducing conditionals
  - simple if $(a \ <\ comp\ >\ b)$
  - nested if/else
  - Complexity of nested if/else: all path must return; mismatched else
  - `<condition>` combining multiple `<comp>`;
  - Boolean variables/expressions/methods
Assume Boolean method

- public static boolean isOdd(int n)

Students new to boolean often test if a result is true:

```java
if ( isOdd(57) == true ) {   // reads redundant
    ...
}
```

Simplify to:

```java
if ( isOdd(57) ) {   // concise
    ...
}
```

A similar pattern can be used for a false test:

```java
if ( isOdd(57) == false ) {   // does not read well
    ...
}
```
```java
if ( !isOdd(57) ) {   // concise
```
Students new to boolean often have an if/else to return true or false:

```java
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}

public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```
"Boolean Zen" template

- Replace

  public static boolean <name>(<parameters>) {
      if (<test>) {
          return true;
      } else {
          return false;
      }
  }

- with

  public static boolean <name>(<parameters>) {
      return <test>;
  }
Exercise: Apply Boolean Zen

public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) { // cumulative sum
            factors++;
        }
    }
    if (factors == 2) {
        return true;
    } else {
        return false;
    }
}
Switch Statement
Java `switch` statement allows clear listing of multiple choices

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

expression must result in an integral data type, like an integer or char

switch and case are keywords

If expression first matches value2, control jumps to here
The switch Statement: default

- A switch statement can have an optional default case as the last case in the statement.
- The default case has no associated value and simply uses the reserved word default.
- If the default case is present, control will transfer to it if no other case value matches.
- If there is no default case, and no other value matches, control falls through to the statement after the switch.
The **switch** Statement: **break**

- Often a **break** statement is used at the end of each case's statement list.

- A **break** statement causes control to transfer to the end of the **switch** statement.
  - Misunderstanding caused AT&T crash.

- If a **break** statement is not used, the flow of control will continue into the next case.
Exercise

- Implement a method to convert a number between 1 and 20 to its English word

Num2Word.java
Limitation of the `switch` Statement

- The result of the `expression` must be an integral type, e.g.,
  - `case x < 1: // ERROR`

- The value for each case must be a constant expression

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

- Admin and recap
- Text processing
Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is so common that Java introduces a short cut and you do not need to create `String` with `new`.

```java
String name = "text";

String name = expression;
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```
Indexes

- A string is a sequence of characters numbered with 0-based indexes:

  String name = "R. Kelly";

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>R</td>
<td>.</td>
<td>K</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

- First character's index : 0
- Last character's index : 1 less than the string's length

- The individual characters are values of type `char`
String Access/Modify Methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>number of characters in this string</td>
</tr>
<tr>
<td>charAt(index)</td>
<td>The character at index</td>
</tr>
<tr>
<td>substring(index1, index2) or substring(index1)</td>
<td>the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 is omitted, grabs till end of string</td>
</tr>
<tr>
<td>indexOf(str)</td>
<td>index where the start of the given string appears in this string (-1 if not found)</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>a new string with all lowercase letters</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>a new string with all uppercase letters</td>
</tr>
<tr>
<td>replace(str1, str2)</td>
<td>replaces occurrences of str1 with str2</td>
</tr>
</tbody>
</table>

- These methods are called using the dot notation:

```java
String name = "Dr. Dre";
System.out.println( name.length() ); // 7
```
String **Method Examples**

```java
// index       012345678901234
String s1 = “Yale University”;
String s2 = “CPSC112, Spring”;
System.out.println( s1.length() );         // 15
System.out.println( s1.indexOf("e") );     // 3
System.out.println( s1.substring(5, 15) );  // “University”
String s3 = s2.substring(9, 15);            // “spring”
```

- **Given the following string:**

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

- **How would you extract the word "Building" from book?**

- **Write code that can extract the first word from any string.**

`FirstWord.java`
Modifying Strings

- **Methods** like `substring` and `toLowerCase` **build** and return a new string, rather than **modifying** the current string.

  ```java
  String s = "lil bow wow";
  s.toUpperCase();
  System.out.println(s); // lil bow wow
  ```

- To modify a variable's value, you must reassign it:

  ```java
  String s = "lil bow wow";
  s = s.toUpperCase();
  System.out.println(s); // LIL BOW WOW
  ```
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 => Number

- You can use scanner to parse a string.
  ```java
  String s = "Year 2014 is great!";
  Scanner scan = new Scanner( s );
  String word = scan.next(); // skip Year
  int year = scan.nextInt();
  ```

- You can use the `Integer.parseInt()` method to convert a string to a number.
  ```java
  String s = "Year 2014 is great!";
  Scanner scan = new Scanner( s );
  String word = scan.next(); // skip Year
  String yearStr = scan.next();
  int year = Integer.parseInt( yearStr );
  ```
Number \Rightarrow String with Format Control

Before Java 1.5, Java provides formatting classes such as `NumberFormat` and `DecimalFormat` classes as part of the `java.text` package.

But many programmers like the more flexible `format string` control structure from the C programming language.

Hence, the format string is introduced in the `System.out.printf()` method and the `String.format()` method.
A format string can contain *placeholders* to insert parameters, e.g.,

- `%d`  integer
- `%f`  real number
- `%s`  string

  - these placeholders are used instead of `+` concatenation

**Example:**

```java
int x = 3;
int y = -17;
System.out.printf("x is \%d and y is \%d\n", x, y);
// x is 3 and y is -17!
```
Formatting Width

- `%Wd` integer, W characters wide, right-aligned
- `%Wd` integer, W characters wide, left-aligned
- `%Wf` real number, W characters wide, right-aligned
- ...

```java
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%4d", (i * j));
    }
    System.out.println(); // to end the line
}
```

Output:

```
1   2   3   4   5   6   7   8   9  10
2   4   6   8  10  12  14  16  18  20
3   6   9  12  15  18  21  24  27  30
```
### Formatting Precision

- `% .Df` real number, rounded to `D` digits after decimal
- `%W .Df` real number, `W` chars wide, `D` digits after decimal
- `%W .Df` real number, `W` wide (left-align), `D` after decimal

```java
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

**Output:**

```
your GPA is 3.3
more precisely: 3.254
```
Many more formatting control options supported format string, e.g., using the comma (,) to display numbers with thousands separator

```java
System.out.printf("%,d\n", 58625);
System.out.printf("%,.2f\n", 12345678.9);
```

Output:

58,625
12,345,678.90
## String Boolean Methods

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

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!");
}
Practice: 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));
}
Regular Expression

- 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
<table>
<thead>
<tr>
<th>Regular Expression</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\s. Dr\s.")

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>
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
Backup Slides
Comparing Strings

- Relational operators such as < and == fail on objects.

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

- This code will compile, but it will not print the song.

- == compares objects by references (seen later), so it often gives false even when two Strings have the same letters.