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
  - combining multiple <comp>

- Boolean variables/methods

"Boolean Zen", part 1

- Assume Boolean method
  - public static boolean isOdd(int n)

- Students new to boolean often test if a result is true:
  - if (isOdd(57) == true) { // reads redundant
    ...
  }

- Simplify to:
  - if (isOdd(57)) { // concise
    ...
  }

- A similar pattern can be used for a false test:
  - if (isOdd(57) == false) { // does not read well
  - if (!isOdd(57)) { // concise

"Boolean Zen", part 2

- Students new to boolean often have an if/else to return true or false:
  - 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
  ```java
  public static boolean <name>(<parameters>) {
    if (<test>) {
      return true;
    } else {
      return false;
    }
  }
  ```
- with
  ```java
  public static boolean <name>(<parameters>) {
    return <test>;
  }
  ```

Exercise: Apply Boolean Zen

```java
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

Alternative Testing using switch

- Java switch statement allows clear listing of multiple choices
  - expression must result in an integral data type, like an integer or char
  ```java
  switch (expression) {
    case value1 :
      statement-list1
    case value2 :
      statement-list2
    case value3 :
      statement-list3
    case ...
  }
  ```
- 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.
  ```java
  switch (expression) {
  case value1:
  statement-list1
  case value2:
  statement-list2
  case value3:
  statement-list3
  default: ...
  }
  ```

- The value for each case must be a constant expression.

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";
  ```

Indexes

- A string is a sequence of characters numbered with 0-based indexes:
  ```java
  String name = "R. Kelly";
  ```

<table>
<thead>
<tr>
<th>Character</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</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>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";
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);
System.out.println(s3.toLowerCase());
```

**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.toLowerCase();
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
  ```

**Strings Question**

- Given the following string:
  ```java
  // index       0123456789012345678901
  String book = "Building Java Programs";
  ```

  - How would you extract the word “University” from book?

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

**Strings Answer**

```java
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");
    }
}
```

**Number => 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.
Formatting Text

```
System.out.printf("format string", parameters);
String newStr = String.format("format string", parameters);
```

- 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!
", x, y);
```

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html

Formatting Width

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

```
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

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

Output:

```
your GPA is 3.3
more precisely: 3.254
```

Additional Formatting Control

- Many more formatting control options supported format string, e.g., using the comma (,) to display numbers with thousands separator

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

Output:

```
58,625
12,345,678.9
```

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);
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
```
```java
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);
}

// 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.

Examples

```java
Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String name = console.nextLine();
...
name.startsWith("Prof. Dr.");
name.matches("^Prof\sDr\.");
name.endsWith("Esq.");
name.matches("Esq\.$");
```

Meta Chars

<table>
<thead>
<tr>
<th>Regex</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>Any digit, short for [0-9]</td>
<td>$X$ finds no or several letter X</td>
</tr>
<tr>
<td>\D</td>
<td>A non-digit, short for [^0-9]</td>
<td></td>
</tr>
<tr>
<td>\s</td>
<td>A whitespace character, short for [\s\t\n\x0b\r\f]</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>A non-whitespace character, short for [^\s]</td>
<td></td>
</tr>
<tr>
<td>\w</td>
<td>A word character, short for [a-zA-Z_0-9]</td>
<td></td>
</tr>
<tr>
<td>\W</td>
<td>A non-word character[^w]</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Matches a word boundary where a word character is [a-zA-Z0-9-]</td>
<td></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</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 time, ? is short for ?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 number of preceding literal \d</td>
<td>finds X digits, (10) for any character sequence of length 10</td>
</tr>
<tr>
<td>{X,Y}</td>
<td>Occurs between X and Y times, {X,Y} means \d must occur at least once and a maximum of four</td>
<td></td>
</tr>
<tr>
<td>(?X)</td>
<td>Occurs between X and Y times, (10) for any character sequence of length 10</td>
<td></td>
</tr>
<tr>
<td>(?X)</td>
<td>Occurs between X and Y times, (10) for any character sequence of length 10</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>Occurs between X and Y times, (10) for any character sequence of length 10</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>Occurs between X and Y times, (10) for any character sequence of length 10</td>
<td></td>
</tr>
</tbody>
</table>
Example

Scanner console = new Scanner(System.in);
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

Comparing Strings

- Relational operators such as < and == fail on objects.
- Comparing objects by references (seen later), so it often gives false even when two Strings have the same letters.

Backup Slides