CS101

Java Programming

Fall 2012

Thursdays 10:00-12:00noon

RF Academy

Announcements

- Any Questions?!?!

`String/char` question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute.
Your secret key: 10
The encoded message: lbkn drsxuc kxqovsxk sc medo.
```

Strings answer 1

```
// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

import java.util.*;

public class SecretMessage {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Your secret message: ");
        String message = console.nextLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = console.nextInt();

        String hidden = encode(message, key);

        System.out.print("The encoded message: " + hidden);

    }

    ...
```

Strings answer 2

```
    // This method encodes the given text string using a Caesar
    // cipher, shifting each letter by the given number of places.
    // Assumes shift is between -26 and +26 exclusive
    public static String encode(String text, int shift) {
        String result = "";
        for (int i = 0; i < text.length(); i++) {
            char letter = text.charAt(i);

            // shift only letters (leave other characters alone)
            if (letter >= 'a' && letter <= 'z') {
                letter = (char) (letter + shift);

                // may need to wrap around
                if (letter > 'z') {
                    letter = (char) (letter - 26);
                } else if (letter < 'a') {
                    letter = (char) (letter + 26);
                }
            }
            result = result + letter;
        }
        return result;
    }
}
```

`Character` wrapper class

- Just as the `Math` class provided us with many useful mathematical functions, the `Character` class provides many character-related functions

- These are all static methods
  - Called on the `Character` class

- They take a single character as a parameter
  - Already saw earlier that we cannot call methods directly on data of type `char`, since `char` is a primitive type

- Example:

```
Character.isLetter(ch)   //ch is var of type char
```

Static Methods in Class `Character`

Name	Description	Type of Arguments	Type of Value Returned	Example	Value Returned
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')` `Character.toUpperCase('A')`	Both return `'A'`
`toLowerCase`	Convert to lowercase	`char`	`char`	`Character.toLowerCase('a')` `Character.toLowerCase('A')`	Both return `'a'`
`isUpperCase`	Test for uppercase	`char`	`boolean`	`Character.isUpperCase('A')` `Character.isUpperCase('a')`	`true` `false`
`isLowerCase`	Test for lowercase	`char`	`boolean`	`Character.isLowerCase('A')` `Character.isLowerCase('a')`	`false` `true`
`isWhitespace`	Test for whitespace	`char`	`boolean`	`Character.isWhitespace(' ')` `Character.isWhitespace('A')`	`true` `false`
Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line break character (`'\n'`).					
`isLetter`	Test for being a letter	`char`	`boolean`	`Character.isLetter('A')` `Character.isLetter('%')`	`true` `false`
`isDigit`	Test for being a digit	`char`	`boolean`	`Character.isDigit('5')` `Character.isDigit('A')`	`true` `false`

Precondition and Postcondition Comments

- The *precondition* for a method states the condition(s) that must be true before the method is invoked.
  - If the precondition is not met, the method should not be used and cannot be expected to perform correctly.

- The *postcondition* describes the result(s) of the method invocation, or what will be true after the method is done.

Precondition and Postcondition Comments

- If the precondition is satisfied and the method is executed, the postcondition will be true.

- Example

```
/**
   Precondition: The integer parameter is nonnegative (>=0).
   Postcondition: The factorial is returned.
*/
public static int factorial(int n) {
   ...
}
```

Precondition and Postcondition Comments

- If a returned value is the only postcondition, the postcondition often is not stated (as it is fairly obvious).

- The statement of the precondition and the postcondition typically precede the associated method in the form of a `javadoc` comment (`/**...*/`)

  - Let's see a quick demo of `javadoc` comments

- Important: We will use JavaDoc comments to document all our methods from this point forward.

Comment types

- `/**` comments are treated specially by the compiler
  - They will produce JavaDoc documentation that can be displayed by an IDE
  - You use these to comment method headers and class headers
  - You provide information on how to use the method or class
    - These are intended for the user, not other programmers
- `//` comments are ignored by the compiler
  - You use these to document the internal workings of your method or class
  - You provide information on how the method or class works
    - These are intended for other programmers, not the user

Verifying Preconditions

- A method should always verify any preconditions before it starts its real work.

- If a precondition is found not to be satisfied, the method can take one of several actions:
  - Return an "*error*" value
  - Stop the execution of the program
    - Use an assertion (we will visit these later)
    - Use `System.exit()`
  - Throw an exception
    - Read pp. 262-266

- Example: Let's verify the precondition of the Caesar cipher's encode method.

Exception example

```
/**
* encode --
* This method encodes the given text string using a Caesar
* cipher, shifting each letter by the given number of places.
* Only lower case letters are shifted.
* Precondition: shift amount is between -26 & +26 (exclusive)
* @param text -- string to encode
* @param shift -- shift amount
* @return  -- the encoded string
*/
public static String encode(String text, int shift) {
    if (shift<=-26 || shift>=26) {
        throw new IllegalArgumentException("Illegal shift amount");
    }
    String result = "";
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        //etc...
}
```