CS 2334
Project 1: Elemental Finch Actions

Object Oriented Design

Class Hierarchies and Abstraction

Generating Proper Documentation

String Manipulation

Plan for Today

- Using javadoc effectively
- Project 1 overview:
  - What to do
  - When is it due
  - What to turn in
- String manipulation
- Begin design of your project

Documentation

Javadoc will include some of your in-line documentation into the html files that are generated

- See “Documentation Requirements.pdf” for details of use (in projects/general)

An Example: Top of Your Java Source File

```
/**
 * @author DO NOT INCLUDE AUTHOR NAMES
 * @version 1.0
 *
 *<P>
 *
 *  Class represents the action of
 *    producing a tone.
 */

public class FinchTone extends FinchAction {
:
```

An Example: Method Documentation

```
/**
 * Primary constructor
 *
 * @param name     String describing
 *                 the name of the action.
 * @param durationDuration of the action
 * @param frequency    Frequency of the
 *                 tone to be generator
 */
FinchTone(String name, int duration, int
frequency){
```

An Example: Method Documentation

```
/**
 * Accessor: Frequency of the
 *tone to be generated
 *
 * @return The tone frequency in Hz
 */
double getFrequency() {
        :
        :
```

More Complete Example

```
/**
 * A descriptive comment goes here. This comment may be several lines long.
 * <P>
 * Algorithm:<br>
 * 1. Each step of the algorithm is listed here.<br>
 * 2. Be sure to put an html <br> tag after each step so that
 *    each step shows up on a separate line.<br>
 * </P>
 * @param             input      Each paramter has a separate listing like
 *                                          this one.
 * @return            Include a descriptive comment describing the return
 *                    type.
 * @exception         IllegalArgumentException    Explain when this
 *                                                exception will be thrown.
 * <dt><b>Conditions:</b>
 * <dd>PRE  -         List the precondition here. Each precondition has a
 *                    separate listing.
 * <dd>POST -         Postconditions are listed the same way as
 *                    preconditions but with <dd>POST instead of <dd>PRE.
 */
```

Project 1 Components

- FinchAction: superclass for general actions
- FinchActionTimed: a superclass for classes with a notion of time
- Subclasses: FinchMove, FinchBuzz, and FinchNose
  - For each: execute() method performs the action with the Finch
- FinchActionList: creating and manipulating a list of actions
- Reading an action list from a file

Project 1 Components (cont)

- Allow a user to specify which actions are displayed and executed
- milestoneX.java files: for each milestone, create a driver class (milestoneX) that tests the components that you wrote for that milestone
  - Note: some milestones involve creation of abstract classes only. In these cases, you will need to make these classes concrete for testing purposes

Example File

MOVE	forward_short	2000	10.0	10.0
MOVE	forward_long	5000	20.0	20.0
MOVE	forward_left	2000	10.0	20.0
MOVE	forward_right	2000	20.0	10.0
NOSE	dance	255	30	0
MOVE	dance	2000	15.0	15.0
NOSE	dance	0	30	255
BUZZ	dance	1000	200	
MOVE	dance	4000	-20.0	20.0
BUZZ	dance	500	400	
NOSE	dance	0	255	20

User Interaction

- User can specify (by typing) a certain name or “All”
- Actions with matching name will be displayed and executed in order

Design: On Paper

- Cover page: Group members, work contributed, and outside citations
- The UML on engineering paper: should be neatly arranged and easily readable

Hardcopy Due: Sept 16th @ 5:00pm

Design: On Computer

Submit **<u>project1_design</u>**.zip

- Documentation and stubs only (no "working" code)
  - Class variables and method prototypes
  - Points will be subtracted for code (except the necessary "return" keywords)
- Javadocs-produced html made by using proper documentation in the source files

Electronic Copy Due: Sept 16th @ 5:00pm

Design

If a design is submitted early in its entirety, we will attempt to evaluate it early

- You must inform your TA when your submission is complete

Final Project: On Paper

Update from stage 1

- Cover page: Group members, work contributed, and outside citations
- The UML on engineering paper: should be neatly arranged and easily readable

Hardcopy Due: Sept 23th @ 5:00pm

Final Project: On Computer

Submit project1.zip:

- Complete working code
- Proper documentation (internal and Javadoc)

Electronic Copy Due: Sept 23th @ 5:00pm

Final Project: Demonstration

Demonstrate to the instructor or one of the TAs by Sept 23th @ 5:00pm

- Be prepared for new test files

Final Project: Bonus

If **all** elements of the project are complete by 5:00pm on Monday the 20th, your group grade will be multiplied by 1.05

- You must inform your TA when you have completed an early submission

Overriding toString() in your Subclasses

- You can implement toString() which provides a string describing your instance
- A common use case will be to display important pieces of information when using:

```
System.out.println(MyClass);
```

Example of overriding toString()

```java
public class StringsExample {
    private int valOne;
    private int valTwo;

    public StringsExample(){
        valOne = 0;
        valTwo = 0;
    }

    public StringsExample(int valOne, int valTwo){
        this.valOne = valOne;
        this.valTwo = valTwo;
    }

    public String toString(){
        return "StringsExample("+valOne + ", " + valTwo + ")";
    }
```

Output from our example

Calling StringsExample.toString() in two different ways:

```
StringsExample se1 = new StringsExample(2,3);
System.out.println("System.out.println(se1.toString()) : " + se1.toString());
System.out.println("System.out.println(se1) : " + se1);
```

Output:

```
Console
<terminated> StringsExample [Java Application] /opt/jdk1.6.0_20/bin/java (Sep 8, 2010 9:56:24 AM)
System.out.println(se1.toString()) : StringsExample(2, 3)
System.out.println(se1) : StringsExample(2, 3)
```

Equals

- String.equals(String s)
  - Useful for checking if “MOVE” == “blink”
- String.equalsIgnoreCase(String s)
  - Useful for checking if “MovE” == “moVe”
- String.contains(String s)
  - Can tell you that “The finch should move” includes “move”

Examples using equals,contains

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

    String abc = "abc";
    String aBc = "aBc";
    String longString = "12345----abc-----6789";

    boolean equalsResult1 = abc.equals(aBc);
    boolean equalsResult2 = abc.equalsIgnoreCase(aBc);
    boolean containsResult = longString.contains(abc);

    System.out.println("abc and aBc with equals(): " + equalsResult1);
    System.out.println("abc and aBc with equalsIgnoreCase(): " + equalsResult2);
    System.out.println("12345----abc-----6789 contains abc? : " + containsResult);
    return;
}
```

Console

```
<terminated> StringsExample [Java Application] /opt/jdk1.6.0_20/bin/java (Sep 8, 2010 10:01:08 AM)
abc and aBc with equals(): false
abc and aBc with equalsIgnoreCase(): true
12345----abc-----6789 contains abc? : true
```

Referencing the Finch Support

At the top of your source java file(s) – may need to be included in several files:

```
import finch.*;
```

Opening the connection to the Finch:

```
Finch myFinch = new Finch();
```

Used only at the end of the main method:

```
myFinch.quit();
```

And for the rest of lab

Finalize groups and start your UML…

Groups:

- Pairs only
- May not cross lab sections
- Can only be assigned if members are present (unless prior arrangements have been made with the instructor)