Carleton University
Department of Systems and Computer Engineering
SYSC 1101 - Object-Oriented Software Development - Winter 2006

Lab 2

Background Reading

Objects First with Java, Chapters 1 through 3 (the first two chapters can be downloaded from the course Web site).
The BlueJ Tutorial. This tutorial is on the SYSC 1101 Software CD, or can be downloaded from `http://www.bluej.org`.

Attendance

To receive credit for attending this lab, you must email a jar file containing the project that you worked on during the lab session, by no later than 4:45 p.m., January 17, 2006. See Step 13 for instructions on how to do this. Even if you don't finish the lab exercise, make sure you email the jar file; i.e., submit the project even if it is incomplete.

Objective

The intent of this exercise is to provide you with further experience using the BlueJ environment. You will learn how to modify classes in a project, and store the project in a jar (Java archive) file. The project is version 2 of the Game of Nim. (This project is slightly different from the version that was presented in class and used in last week's lab).

Procedure

1. Download `Nim.java` and `NimUI.java` from the *nim-v2* folder on the course Web site. (Make sure that you get the Version 2 files, not the files from last week's lab).
2. Launch BlueJ (for example, by double-clicking on the BlueJ icon, or selecting Start > All Programs > BlueJ > BlueJ. Note that double-clicking on `Nim.java` will **not** launch BlueJ).
3. From the menu bar, select Project > New Project... A New Project dialogue box will appear. Browse to the folder where you want to store your project files, and in the File Name: field type `nim-v2`. This will create a folder called `nim-v2`. If you are working in one of the SCE labs, make sure you create this folder on your M: drive. Don't try to put your project in the BlueJ examples folder (`C:\BlueJ\examples`) as you cannot store files there.
4. From the menu bar, select Edit > Add Class From File... A dialogue box named Select Java Class to Add will appear. Browse until you find `Nim.java`, select it, and click the Add button. A copy of `Nim.java` will be placed in your `nim` folder (the original file will not be moved or deleted). A box labelled `Nim` should appear in the BlueJ main window. This box is an icon that represents the `Nim` class.
5. Add `NimUI.java` to the project (see step 4).
6. Double-click on the `NimUI` class icon. The Java source for the class will appear in an editor window. Notice that the `play()` method has changed from Version 1.00 of this class. If the name of the current player is "Computer", a dialogue box is displayed, telling the player that it is the computer's turn. After the player clicks OK, `takeTurn()` is invoked on the `nim` object. This causes the computer to remove 1, 2, or 3 sticks. After this method returns, a second dialog box is displayed, telling the player how many sticks the computer removed.
7. Double-click on the `Nim` class icon. The Java source for the class will appear in an editor window. Notice that the constructor is now overloaded. The new constructor has two parameters (the number of

sticks and the name of the first player). This constructor initializes the second player's name to "Computer". It provides a convenient way to create the `Nim` object that is used when a player wants to play against the computer. Notice that the `takeTurn()` method is incredibly naive - it always removes 1 stick from the pile. We'll change this later in this exercise.

8. Error messages will be displayed if you compile the project, because the `sticksRemoved()` method in `Nim` has not been completed. Edit the `Nim` class, adding a private field called `removed`. This field will keep track of the number of sticks removed during the last turn. Edit `sticksRemoved()` to return the value stored in this field. Edit the two constructors and `newGame()` to initialize this field to 0. Modify `removeSticks()` to change the value stored in this field.
9. Compile the project. Interactively create a `Nim` object and a `NimUI` object, and play a round of Nim against the computer. (You may have to close some windows on your desktop to find the game's user interface.) It shouldn't be difficult to beat the computer, because every time it takes a turn, it removes exactly one stick.
10. There is an easy way to determine a perfect strategy for winning Nim. If the number of sticks is known, one of the players has a winning strategy. Consider the situation from the point of view of the player whose turn it is. One stick left is clearly a losing situation, but two, three, or fours sticks are winning situations, because by removing the appropriate number of sticks, the player can leave the other player with the last stick. If there are five sticks left, this is a losing situation, because no matter what the player does, the other player will be left with a winning situation; that is, the other player will be left with two, three, or four sticks. Continuing in this manner, we see that if there are six, seven, or eight sticks left, the player can leave the other player with five sticks; that is, leave the other player in a losing situation. As such, six, seven or eight sticks left is a winning situation for the player. To summarize:

    ```
    No. of sticks: 1 2 3 4 5 6 7 8 9 10 11 12 13 ...
    Win/lose:      L W W W L W W W L W  W  W  L  ...
    ```

    You are now going to change your `Nim` class so that the computer does a better job of deciding how many sticks to remove. Modify `takeTurn()` so that it decides to remove 1, 2, or 3 sticks according to the perfect strategy described in the previous paragraph. (Don't duplicate any of the code in `removeSticks()`: after determining how many sticks to remove, `takeTurn()` must call `removeSticks()` to remove the sticks from the pile. Hint: you will find the remainder operator, `%`, useful.

    (This part was based on an exercise from *An Introduction to Programming and Object-Oriented Design Using Java, Second Edition*, Nino and Hosch.)

11. Compile the project. Interactively create a `Nim` object. Right-click on the `Nim` object, and select *Inspect* from the pop-up menu. An Object Inspector window will appear. The object inspector will help you keep track of the state of the `Nim` object as you test `takeTurn()`. Verify the behaviour of your `takeTurn()` method by interactively invoking this method several times. (Right click on the `Nim` object, and select `takeTurn()` from the pop-up menu).
12. Create a `NimUI` object, and play Nim against the computer. Play the game several times. Does the computer always win if you follow a naive strategy (e.g., you always take 2 sticks, unless there is only one stick remaining)?
13. A document describing how to submit your assignments and lab exercises is posted on the course Web site. Package your Nim project in a jar file called `lab2_`*`abcdefghi`*`.jar` (where *`abcdefghi`* is your 9 digit Carleton student number). Use your Carleton Connect account or your Engsoc account to email this file to `bailey@sce.carleton.ca`. In the subject line, type:

    `SYSC 1101 Lab 2` *`yourfirstname yourlastname`*

When typing this line, make sure you follow the format described in the assignment and lab submission instructions.

14. When you've finished this exercise, close the project by selecting Project > Close from the BlueJ menu, then quit BlueJ.

Version 1.01, January 17, 2006