

Lab 5: An example of programming in the real world.

The results of the 2000 presidential election were called into question because of the use of manual voting means in several states, notably Florida. Voters were asked to punch holes in cardboard to reflect their selection, and the cardboard sheets were counted by machines. The system was 50 years old. The voters' choices were sometimes incompletely punched-through, leaving a hanging "chad", and resulting in miscounts or disqualified entries. Consequently, some votes were never counted in the election. In all respects, the election was close, and the final decision was very much dependent on the electoral votes from the state of Florida. In the end, the miscast votes might have been a deciding factor.

To prevent such problems in the future, several states have moved to computerize their voting systems. What does it mean to "computerize" a manual process? In the case of voting systems, it suggests removing the inconsistencies and unavailability of manual, mechanical hole punching and counting. Cardboard sheets were replaced with computer screens that display clear, readable choices. Users select candidates by touching the candidates' names on-screen. A computer program, written by programmers just like you, ensures that only one choice is made and counts all of the votes for all of the candidates. At the end of the voting day, all of the separate voting locations electronically email their totals to a central computer, which tallies the results.

The voting system illustrates many of the things that computers, software, and programmers do best: presenting colorful, easy-to-read, unambiguous information on a computer screen; allowing users to make selections between many alternatives; checking a user's selection to guarantee that it is a proper one; and counting up literally hundreds of thousands of entries in a dependable, tireless, repeatable process.

Many of the counties and states that have computerized their voting systems use software produced by University of Buffalo graduates who work as programmers at NTS-Compumail in Niagara Falls.

For an interesting take on this issue, see:

<http://www.youtube.com/watch?v=5WVG34cv0zM>
<http://www.youtube.com/watch?v=cy1IIAXeV30>

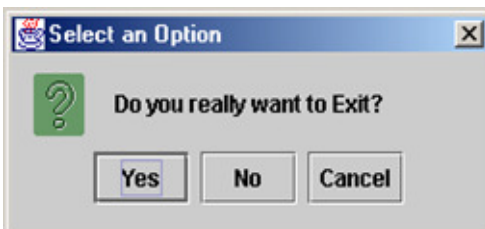
New programming concepts covered in this lab:

There are three new concepts in this lab – using a new method in `JOptionPane` called `showConfirmDialog`, the use of a *while(true)* statement to repeat a program forever, and finally, the use of *objects to store information*. In this case, we will be creating Obama and McCain objects to each keep track of their own votes.

1. `JOptionPane.showConfirmDialog` is a simple "yes/no/cancel" pop-up menu. It is called like this:

```
int whichButtonGotPressed = 0; // an int variable to hold a value returned from the method  
whichButtonGotPressed = JOptionPane.showConfirmDialog (null, "Do you really want to Exit?");
```

When called, it shows this:



It returns an int variable (in this case, placed into *whichButtonGotPressed*) –
`whichButtonGotPressed` = 0 means Yes
`whichButtonGotPressed` = 1 means No

whichButtonGotPressed = 2 means Cancel

so you can test the *whichButtonGotPressed* variable to see what the user wants, like this

```
if (whichButtonGotPressed == 0 )
{
    System.exit( 0 ); //exit if the user pressed YES
}
```

The call to `showConfirmDialog` contains two parameters in its parenthesis... the first one is “null”, which essentially is no value at all. The author of the `showConfirmDialog` method decided to allow the user to place the pop-up window on any networked computer. That first parameter tells where, and null selects the computer running the program (yours).

2. Running a program repeatedly... any code placed within the following brackets, will repeat forever:

```
int x = 1;
while (x == 1)
{

}
```

It's easy to see why... because x is always 1 (that is, `x == 1` is always “true”), the while loop is always activated. Programmers often use the following, to the same effect:

```
while ( true )
{

}
```

In Lab 5, we will combine these two new concepts.

3. Using an object to store its own information – we are going to define a class to model the behavior of a candidate (essentially, collect votes). We will instantiate two copies... a Barack object, and a McCain object. Each object will separately contain the number of votes cast for that candidate. **Read Chapter 2 in Head First Into Java.**

Programming Lab 5:

In this lab, we will construct a simple voting system, using two classes in two files: a `CandidateClass`, which contains one variable and three methods, and a `VotingClass` (with main), which will instantiate a separate object for each of two presidential candidates, allows the user to vote, and counts the number of votes.

Programming Task 1: construct the `CandidateClass` file

1. Create the class framework by beginning the class with the usual `public class CandidateClass`, followed by an opening and closing bracket. Comment the closing bracket with `// end class CandidateClass`.
2. Between the brackets of the class, define an int variable named `numberOfVotes`. Initialize it to zero. Let's make the variable public, like this:

```
public int numberOfVotes = 0;
```

3. The first method we should write will be the method that gets run when the VotingClass is instantiated into an object, in some far distant *main*. It's called the constructor, and it has a very simple format. It can receive parameters, but not return any, and it is always defined like this:

```
public className( and parameters to start the object )  
{  
  
}
```

In our case, we want a simple constructor to merely announce the creation of this class as an object:

```
public CandidateClass( String name )  
{  
    System.out.println( "Creating: " + name);  
}
```

4. After the constructor, create a **public void** method that accepts no parameters, named `incrementVotes`, that simply adds one, to the `numberOfVotes` variable. (in programming, this is called a *mutator* method, because it changes – or mutates – a single variable).
 - a. You must declare the method **public void incrementVotes()**, followed by an opening and closing bracket. Comment the closing bracket with **// end method incrementVotes**.
 - b. The only code between the brackets of method `incrementVotes()`, should be to assign to `numberOfVotes`, its current value plus 1.
5. After the closing bracket of `incrementVotes()`, create a new method called `getVotes()`, that accepts no parameters but returns the `numberOfVotes`. (in programming, this is called an *accessor* method).
 - a. You must declare the method **public int getVotes()**, followed by an opening and closing bracket. Comment the closing bracket with **// end method getVotes**.
 - b. The only code between the brackets of method `getVotes()`, should be to return `numberOfVotes`, so that when this method is called from `main`, it will return the value of the `numberOfVotes`. Therefore, it should have only one line of code:
return(numberOfVotes);

There! You have a class that models the behavior of a presidential candidate. They get votes. That's pretty much it.

Programming Task 2: Create a VotingClass to count user votes.

1. Create the class framework by beginning the class with the usual **public class VotingClass**, followed by an opening and closing bracket. Comment the closing bracket with **// end class VotingClass**.
2. make sure you **import javax.swing.JOptionPane**; at the top of the class.
3. The Voting class must contain the main method:
public static void main(String args [])

4. Within main, instantiate two objects of the Candidate class:
 - a. The first object will be to track votes for Barack Obama. Call the object Obama, and instantiate the class using `CandidateClass Obama = new CandidateClass("Barack Obama");`
 - b. The second object will be used to track votes for John McCain. Call the object McCain, and instantiate the class using `CandidateClass McCain = new CandidateClass("John McCain");`

Programming Task 3: use `javax.swing.JOptionPane.showConfirmDialog` to ask the user for votes.

1. Within the main method, create an int variable called `buttonPressed`, which will receive the return value of calls to `JOptionPane.showConfirmDialog`
2. Within the main method, create an int variable called `votesForObama`, which will receive the return value of calls to `Obama.getVotes()`, reflecting the number of votes cast for Barack Obama.
3. Within the main method, create an int variable called `votesForMcCain`, which will receive the return value of calls to `McCain.getVotes()`, reflecting the number of votes cast for John McCain.
4. You will make two calls to `JOptionPane.showConfirmDialog`. One to cast votes for the appropriate candidate, and one to query the user to exit or vote again.
 - a. The first call will ask the voter, if he/she wants to vote for Obama:
`buttonPressed = JOptionPane.showConfirmDialog(null, "Do you want to vote for Barack Obama?");`
 - b. Immediately after the call above, test the value of `buttonPressed` in an "if" statement. If the user selected "yes" then the value of `buttonPressed` will be zero. If the value of `buttonPressed` is zero, increment the number of votes for Barack Obama by calling the `incrementVotes` method of the Obama object: `Obama.incrementVotes();` Note that the method is sent no parameters, nor does it return any values. It just increments the *class instance* variable `numberOfVotes`, within itself.
 - c. Immediately after the above check, test the value of `buttonPressed` in an "if" statement. If the user selected "no" then the value of `buttonPressed` will be one. If the value of `buttonPressed` is one, it means s/he didn't want to vote for Obama. Since s/he is voting and s/he didn't want to vote for Obama and also since the voter knows there are only two candidates, s/he wants to vote for John McCain. So increment the number of votes for John McCain by calling the `incrementVotes` method of the McCain object: `McCain.incrementVotes();` (If the `buttonPressed` is 2, then the user selected "Cancel" which means he/she doesn't want to vote for both candidates, so we do not do anything. So there will be no "if" statement to check if `buttonPressed` was two).
 - d. The last call will ask the voter, if he/she wants to exit the program:
`buttonPressed = JOptionPane.showConfirmDialog(null, "Do you want to Exit?");`
 - e. Immediately after the call above, test the value of `buttonPressed` in an "if" statement. If the user selected "yes" then the value of `buttonPressed` will be zero. If the value of `buttonPressed` is zero, then the user wants to exit. Write the "if" statement like this:

```

    if (buttonPressed == 0) // user wants to exit
    {
        // perform the tasks in #5 below.
    }

```

5. Handling the exit – we need to display the votes for each candidate.
 - a. Before exiting, call `votesForObama = Obama.getVotes();` and print the result.
 - b. Also before exiting, call `votesForMcCain = McCain.getVotes();` and print the result.
 - c. After printing the vote totals above, exit using `System.Exit(0);`
6. Handling the NO exit – if the voter did not answer yes to the last question, the “if” statement above will fail, and the program will simple continue. If you surround everything in the main method AFTER THE OBJECT INSTANTIATIONS AND AFTER THE VARIABLE DECLARATIONS (but before the calls to JOptionPane) with a while(true) loop, the program will simply start asking its questions again.

note: you want to use the while(true) loop to surround only the code that you want repeated. The general structure of the VotingClass must be:

```

public class VotingClass
{
    public static void main (String [ ] args)
    {
        // variable definitions here
        // object instantiations here
        while(true) {
            // calls to JOptionPane, object.incrementVotes( ), and exit processing here

        }
    } // end main method
} // end class VotingClass

```

Grading:

in CandidateClass:

- Constructor CandidateClass(): [10 pts.]
- Correct incrementVotes() method: [10 pts.]
- Correct getVotes() method: [10 pts.]

in VotingClass:

- Correctly instantiating two objects: [10 pts.]
- Asking for the votes: [10 pts.]
- Calling the correct .incrementVotes(): [10 pts.]
- Repeating until exit: [10 pts.]
- Handling the exit, printing the candidate vote info: [10 pts.]
- Handling the exit, exiting when finished: [10 pts.]

As always, proper formatting, layout, and comments: [10 pts.]