**6. Textareas**

The BankView program has a textfield where the user can enter a deposit in a bank account or a withdrawal. Positive numbers indicate deposits and negative numbers indicate checks. It also has a button that the user clicks to see the last five transactions made. Each user input produces a remark in the large rectangular output area.

The BankView frame looks as shown in the accompanying figure after inputs of 100, -30, and 42, and a button click, except that the bottom part of the output area has been cut off. Scroll bars will appear when the textarea has too much material to display at one time. If you would like to see this BankView program in action, click this [execution](http://docs.google.com/EEdemos_CD.html) link.



The BankView class in Listing 6 has two variables, modelAccount and itsOutput, that contain objects that have to be accessed in several methods of the class. This means you have to define them as instance variables, outside of all methods, as shown in lines 1 and 2. The modelAccount object keeps track of checks and deposits for a bank account. The itsOutput object keeps track of a large area where information written by the program is to appear. We construct it in line 2, we add it to the frame in line 8, and we write information to it in lines 11 and 18.

An **ETextArea** object displays a large amount of textual information. The ETextArea construction in line 2 creates a rectangular area suitable for writing up to 10 rows of words, each row having room for at least 42 characters. This kind of component is a **textarea**. You will also have scroll bars on this area, so the user can scroll up-and-down or left-and-right to read all the material printed there. You add one line to the information displayed in an ETextArea by using **someETextArea.say(messageString)**.

An ETextArea also has the same setText and getText methods that ELabels, EButtons, and EFields have. In particular, **someETextArea.setText("")** clears the area.

Line 7 of this class constructs a new UpdateField (a subclass of EField), then sets its width to be 5 characters. Since the width method returns the EField object itself, you can apply the text method to it, setting its text to be a zero. Since the text method also returns the EField object itself, you can use it as the parameter for the add method. All you need for an applet that contains the BankView program is the following:

**public class BankViewApplet extends javax.swing.JApplet**

{

**�� public void start()**

�� {� setContentPane (new BankView());

�� }� //======================

}

The constructor lays out a button and a textfield for input and a textarea for output (lines 3-8). Then the runtime system pauses until some event takes place: When the button is clicked, a record of the last five transactions in the bank account appears in the textarea (line 11; see the figure above). When the user enters a new transaction in the textfield (line 15), it is checked to make sure it is not blank (line 16). If so, the bank account is updated (line 17) and then the current balance is printed in the textarea (lines 18-19). Line 21 erases the textfield for the user's convenience.

Listing 6 The BankView program (uses the BankAccount class)

**public class BankView extends EPanel**

{

�� private BankAccount modelAccount = new BankAccount();������ // 1

�� private ETextArea itsOutput = new ETextArea (10, 42);������ // 2

**� /\*\* The View:� Lay out the GUI components. \*/**

**�� public BankView()����������������������������������������** ��// 3

�� {� add (new LastFiveButton().text�������������������������� // 4

����������� ("show last five transactions"));����������������� // 5

����� add (new ELabel ("Enter deposit(+) or check(-)"),������� // 6

����������� new UpdateField().width (5).text ("0"));���������� // 7

����� add (itsOutput);���������������������������������������� // 8

�� }� //======================

**� /\*\* Controllers: React to click of button or ENTER key. \*/**

**�� private class LastFiveButton extends EButton�������������** ��// 9

�� {� public void onClick()����������������������������������� //10

����� {� itsOutput.say (modelAccount.lastFiveTransactions());� //11

����� }������������������������������������������������������� //12

�� }� //======================

**�� private class UpdateField extends EField**������������������� //13

�� {� public void onEnter()����������������������������������� //14

����� {� String input = this.getText();����������������������� //15

�������� **if** (input.length() > 0)������������������������������ //16

�������� {� modelAccount.makeChange (input);������������������ //17

����������� itsOutput.say (modelAccount.currentBalance() +���� //18

�������������� " = balance.");�������������������������������� //19

�������� }���������������������������������������������������� //20

�������� this.setText ("");� // clear the entry field��������� //21

����� }������������������������������������������������������� //22

�� }� //======================

}

This program follows the standard **MVC (model-view-controller)** **design**:

�€€€€€€€€ The constructor creates the **view** (what the user sees on the screen and interacts with) and then suspends the action of the program until some input event occurs.

�€€€€€€€€ The last two methods are the **controllers** of what goes on in the program, reacting to events.

�€€€€€€€€ The BankAccount object stores information that is accessed at several points in the execution of the program. So it is a **model** of a part of the real world (a bank account in this case). The controllers act as intermediaries between the model and the view.

The BankAccount makeChange method, called in line 17, accepts any string of one or more characters and makes the change for a check (if less than zero) or a deposit. The methods in this class have nothing to do with GUIs, so we do not discuss them here. You should study the discussion of the BankAccount class at this [language](http://docs.google.com/EElanguage.html#6) link if you are not fully familiar with how to make this kind of class.

[**Exercise**](http://docs.google.com/EEexercises.html#6) **6.1** Change BankView so that it says "The last 5 are " before it lists the five.

**Exercise 6.2** Change BankView as follows, on the assumption that BankAccount has another instance method, numChecks(), to tell the number of checks so far:� Clicking a button causes the number of checks to appear in the textarea.

**Exercise 6.3\*** Revise BankView to have two textfields, one for checks only and one for deposits only. Both will be entered as positive numbers. Delete any leading '$' character from input sent to the makeChange method. Hint: Use the substring method.

**Exercise 6.4\*\*** Write a program that has one textfield and one textarea. The user is to enter numbers in the textfield. After each number is entered, write a line that says "INPUT: " followed by the user's input, then a line that gives the average of all numbers entered so far. Hint: The statement n = Integer.parseInt(this.getText()) produces the numeric equivalent of the string of characters that the user enters. Have a variable that tracks the total entered and another variable that counts how many entries are made.

**7. Timers**

A subclass of **ETimer** is generally defined inside the EPanel subclass. It must have an **onBeep()** instance method. You use the instance method **delay(m)** to tell the timer to wait for a period of m **milliseconds** before executing its onBeep method. 1000 milliseconds is 1 second.� The delay method returns the ETimer object itself, so it can be used in e.g. add (new XTimer().delay(1200)). You can stop the beeping with the ETimer **stop()** method. ETimer is a subclass of ELabel, since an ETimer is often used to display a picture that changes frequently, once each time onBeep is called. So you have available the setPicture, getPicture, setText, and getText methods.

Example: You could add the following three statements in the constructor of any of the example classes we have had. They will display a picture of Rome for just 1.2 seconds, then replace the picture of Rome by the picture of Ireland, if you have defined the RomeTimer class shown:

������� ETimer flash = new RomeTimer();

������� flash.setPicture ("Rome.jpg");

������� add (flash);

**�� private class RomeTimer extends ETimer**

�� {� public void onBeep()

����� {� this.setPicture ("Ireland.jpg");

����� }

�� }

Each EButton and EField has a setEnabled method: **x.setEnabled(false)** keeps the user from clicking the button or typing in the textfield. **x.setEnabled(true)** re-activates the component. When a component is first created, it is of course already enabled.

The GuessNumberGame is a subclass of EPanel you can put in an EFrame (for an application) or a JApplet (for an applet in a webpage). The design is to have a button the user clicks to start a new game. When that happens, the program chooses a secret number from 1 to 100 and invites the user to guess it. Each time the user types a guess into the textfield, the response is "Too high" or "Too low", whichever is appropriate. When the user gets it right, the response is "Right. Congratulations!" and the game stops; the user clicks the button to play again. The GuessNumberGame frame looks as shown in the accompanying figure, after a guess of 40 which was too high. If you would like to see this GuessNumberGame program in action, click this [execution](http://docs.google.com/EEdemos_CD.html#7) link.



The GuessNumberGame class in Listing 7 has four instance variables, since they have to be accessed in several methods of the class:

�€€€€€€€€ The textfield named itsInput is declared in line 1 to allow room for 3 or more characters. It is added to the panel in line 8. It is disabled initially in lines 10, and again disabled at the end of each game in line 35. It is enabled at the start of a game in line 19, when the user clicks the button that says "Click to play the game".

�€€€€€€€€ The label named itsAnswer is declared in line 2 to initially have no text (indicated by the empty string "").� It displays the program's responses in lines 27, 30, and 33.

�€€€€€€€€ The integer value named itsSecret is the secret number the program chooses. The program checks it against the user's guess in lines 26 and 29.

�€€€€€€€€ The timer named itsTimer is declared in line 4. Its purpose is to choose a secret number. It does this by adding 13 to itsSecret number 50 times a second (i.e., a delay of 20 milliseconds between additions) until the user clicks the button to start the game. But when it goes over 100, it subtracts 100 to keep it in the range of 1 through 100 (lines 13-15). In general **x += k** changes x to be k more than it was, and **x -= k** changes x to be k less than it was.

This is a workable way to generate a number sort of at random in the range from 1 to 100. By adding 13 each time, which is a prime number, this method cycles through all 100 possibilities once each two seconds. Since the user will probably take one or more seconds from the time "Congratulations!" is displayed to the start of the next game, this is acceptably "random". Of course, nothing is actually random about the way a computer functions.

Another way to produce a random number is have an instance variable **java.util.Random randy = new java.util.Random();** then each call of **randy.nextInt(n)** produces a "random" non-negative number less than n. Exercise 7.3 illustrates its use.� Note that the ETimer is not added to the display, since there is nothing to show about this particular ETimer; but usually there is.

Line 25 has to obtain a number from the string of characters that the user typed in the textfield.� It uses the standard **Integer.parseInt(someString)** method to derive the number.

Listing 7 The GuessNumberGame program

**public class GuessNumberGame extends EPanel**

{

�� private EField itsInput = new PlayOneGame().width (3);����� // 1

�� private ELabel itsAnswer = new ELabel ("");���������������� // 2

�� private int itsSecret = 50;����������������������� ���������// 3

�� private ETimer itsTimer = new SecretTimer().delay (20);���� // 4

**� /\*\* The View:� Lay out the GUI components. \*/**

**�� public GuessNumberGame()**����������������������������������� // 5

�� {� add (new StartGame().text ("Click to play the game"));� �// 6

����� add (new ELabel ("Guess 1 to 100"));�������������������� // 7

����� add (itsInput);����������������������������������������� // 8

����� add (itsAnswer);���������������������������������������� // 9

����� itsInput.setEnabled (false);��������������������������� �//10

�� }� //======================

**� /\*\* Controllers: React to various events. \*/**

**�� private class SecretTimer extends ETimer�����������������** ��//11

�� {� public void onBeep()������������������������������������ //12

����� {� itsSecret += 13;�������������������� �����������������//13

�������� if (itsSecret > 100)��������������������������������� //14

����������� itsSecret -= 100;��������������������������������� //15

�������� this.delay (20);������������������������������������� //16

����� }���������������������������������������������� ���������//17

�� }� //======================

**�� private class StartGame extends EButton**�������������������� //18

�� {� public void onClick()����������������������������������� //19

����� {� itsTimer.stop();������������������������������������� //20

�������� itsInput.setEnabled (true);�������������������������� //21

����� }������������������������������������������������������� //22

�� }� //======================

**�� private class PlayOneGame extends EField�����������������** ��//23

�� {� public void onEnter()��������������������������� ��������//24

����� {� int guess = Integer.parseInt (this.getText());������� //25

�������� **if** (guess > itsSecret)������������������������������� //26

�������� {� itsAnswer.setText ("Too high");������������������� //27

�������� }���������������������������������������������������� //28

�������� **else if** (guess < itsSecret)�������������������������� //29

�������� {� itsAnswer.setText ("Too low");�������������������� //30

�������� }���������������������������������������������������� //31

�������� **else**� // don't continue, this game is over����������� //32

�������� {� itsAnswer.setText ("Right. Congratulations!");���� //33

����������� itsTimer.delay (20);������������������������������ //34

����������� itsInput.setEnabled (false);���������������������� //35

�������� }���������������������������������������������������� //36

����� }������������� ������������������������������������������//37

�� }� //======================

}

[**Exercise**](http://docs.google.com/EEexercises.html#7) **7.1** What is the first secret number the program chooses, if the user waits exactly 1.00 seconds before clicking the button? What is the second secret number, if the user waits exactly 4.04 seconds before clicking the button again?

**Exercise 7.2** Change GuessNumberGame to have another instance variable itsLimit, which is initially 100. The secret number is 1 up through itsLimit. Provide another button the user can click to change itsLimit to 15 (to make the game easier for small children) or back again to 100. How many guesses would it take you to get it right if itsLimit is 15?

**Exercise 7.3** Change GuessNumberGame to use a java.util.Random object instead of a timer.

**Exercise 7.4\*** You can get along without the setEnabled method if you have to. To do this, eliminate all references to it and then declare an instance variable isEnabled, set to true or false appropriately. Test isEnabled in the onEnter method to see if the textfield should react in any way.

**Exercise 7.5\*\*** Write a program that asks three questions of your choosing. Supply a textfield for each answer, but have the onEnter method do nothing for each textfield. At the end of 30 seconds, the program "calls time" and tells how many answers were right.

**Exercise 7.6\*\*** After you write the program for the preceding exercise, add a label that counts down to zero starting from 30, changing once each second.

Link to [next section](http://docs.google.com/EasyEvents_D.html) (Part D)