Tutorial 8 – **Car Payment Calculator** Application

Introducing the `while` Repetition Statement

Objectives

- In this tutorial, you will learn to:
  - Use the `while` repetition statement to repeatedly execute statements in an application.
  - Use counter-controlled repetition.
  - Use the increment and decrement operators.
  - Display information in `JTextArea`s.

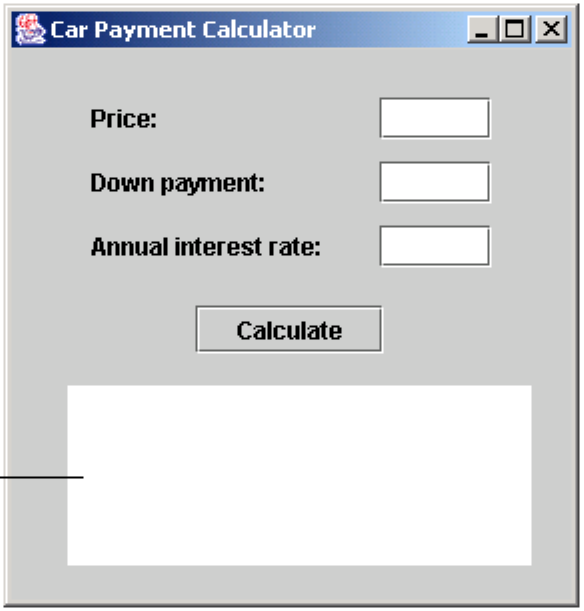
8.1 Test-Driving the **Car Payment Calculator** Application

Application Requirements

Typically, banks offer car loans for periods ranging from two to five years (24 to 60 months). Borrowers repay the loans in fixed monthly payments. The amount of each monthly payment is based on the length of the loan, the amount borrowed and the interest rate. Create an application that allows the customer to enter the price of a car, the down payment amount and the annual interest rate of the loan. Your application should display the loan's duration in months and the monthly payments for two-, three-, four- and five-year loans.

8.1 Test-Driving the **Car Payment Calculator** Application (Cont.)

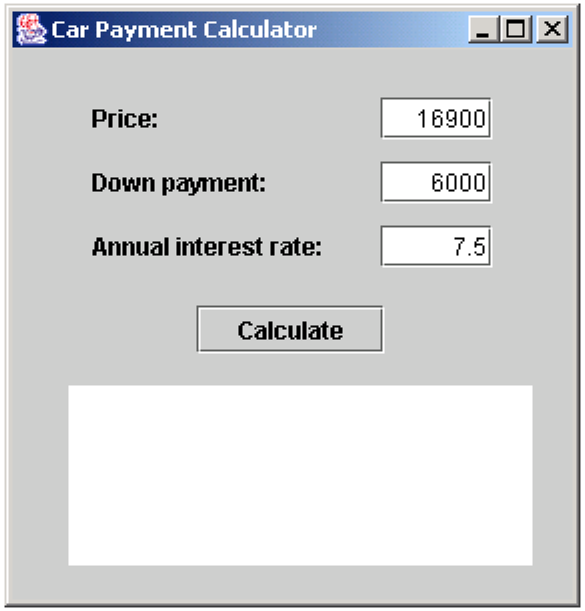
Figure 8.1 **Car Payment Calculator** application before data has been entered.



`JTextArea` component

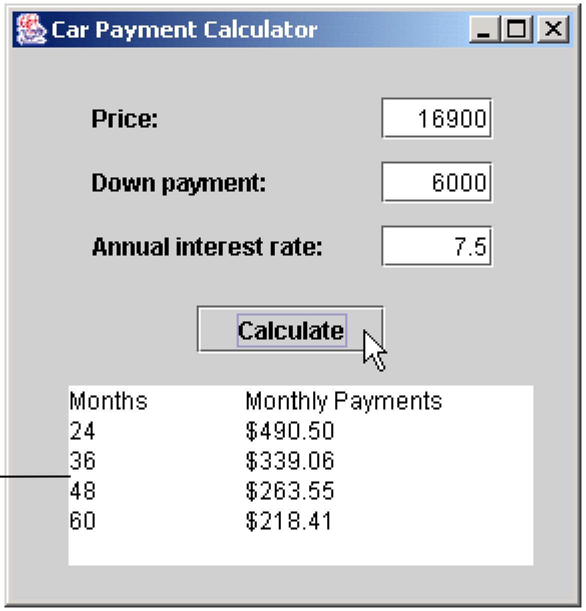
8.1 Test-Driving the **Car Payment Calculator** Application (Cont.)

Figure 8.2 **Car Payment Calculator** application after data has been entered.



8.1 Test-Driving the **Car Payment Calculator** Application (Cont.)

Figure 8.3 **Car Payment Calculator** application displaying calculation results.



Price: 16900
Down payment: 6000
Annual interest rate: 7.5

```
Months         Monthly Payments
24             $490.50
36             $339.06
48             $263.55
60             $218.41
```

Results displayed in tabular format

8.2 while Repetition Statement

- Pseudocode

```
While there are still items on my shopping list
    Purchase next item
    Cross it off my list
```

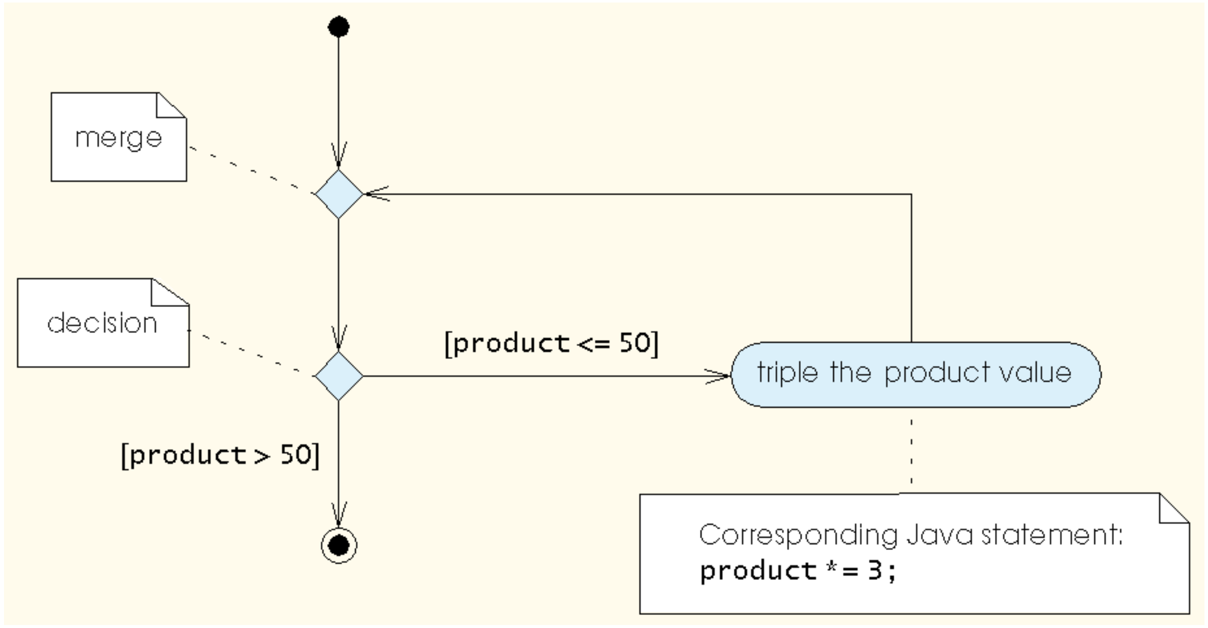
- Find the first power of 3 greater than 50

```
int product = 3;

while ( product <= 50 )
{
   product *= 3;
}
```

- Repitition statement – repeats actions, depending on the value of a condition
- Loop-continuation condition
  - Loop executes while condition remains `true`

8.2 while Repetition Statement (Cont.)

Figure 8.4 while repetition statement UML activity diagram.



8.3 Increment and Decrement Operators

- Unary increment operator (++)
  - preincrement
  - postincrement
- Unary decrement operator (--)
  - predecrement
  - postdecrement

8.3 Increment and Decrement Operators (Cont.)

Operator	Called	Sample Expression	Explanation
`++`	preincrement	`++counter`	Increment `counter` by 1, then use the new value of `counter` in the expression in which `counter` resides.
`++`	postincrement	`counter++`	Use the current value of `counter` in the expression in which `counter` resides, then increment `counter` by 1.
`--`	predecrement	`--counter`	Decrement `counter` by 1, then use the new value of `counter` in the expression in which `counter` resides.
`--`	postdecrement	`counter--`	Use the current value of `counter` in the expression in which `counter` resides, then decrement `counter` by 1.

Figure 8.5 Increment and decrement operators.

8.4 Constructing the **Car Payment Calculator** Application

```
When the user clicks the Calculate JButton
    Initialize loan length to two years
    Clear the JTextArea of any previous text
    Get car price, down payment and annual interest rate
    Calculate loan amount
    Calculate monthly interest rate

    While loan length is less than or equal to five years
        Calculate number of months
        Calculate monthly payment based on loan amount, monthly interest rate
        and loan length in months
        Display result
        Increment loan length in years by one year
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

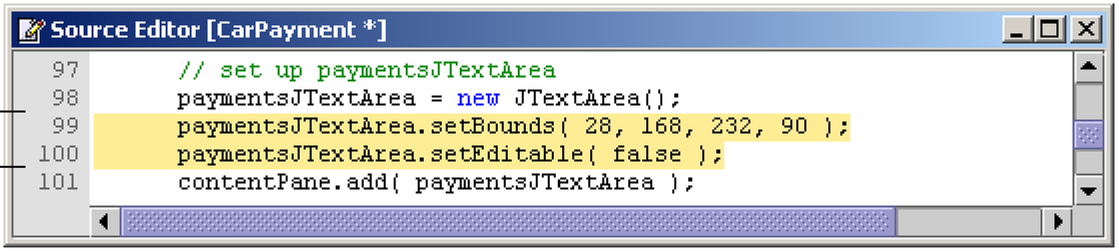
Action	Component	Event
Label all the application's components	`priceJLabel`, `downPaymentJLabel`, `interestJLabel`	
Clear the JTextArea of any previous text	`paymentsJTextArea`	User clicks the **Calculate** `JButton`
Get car price, down payment and annual interest rate	`priceJTextField`, `downPaymentJTextField`, `interestJTextField`	
Calculate the monthly payment and display result	`paymentsJTextArea`	

Figure 8.6 Car **Payment Calculator** application ACE table.

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.7 Customize a `JTextArea` component.

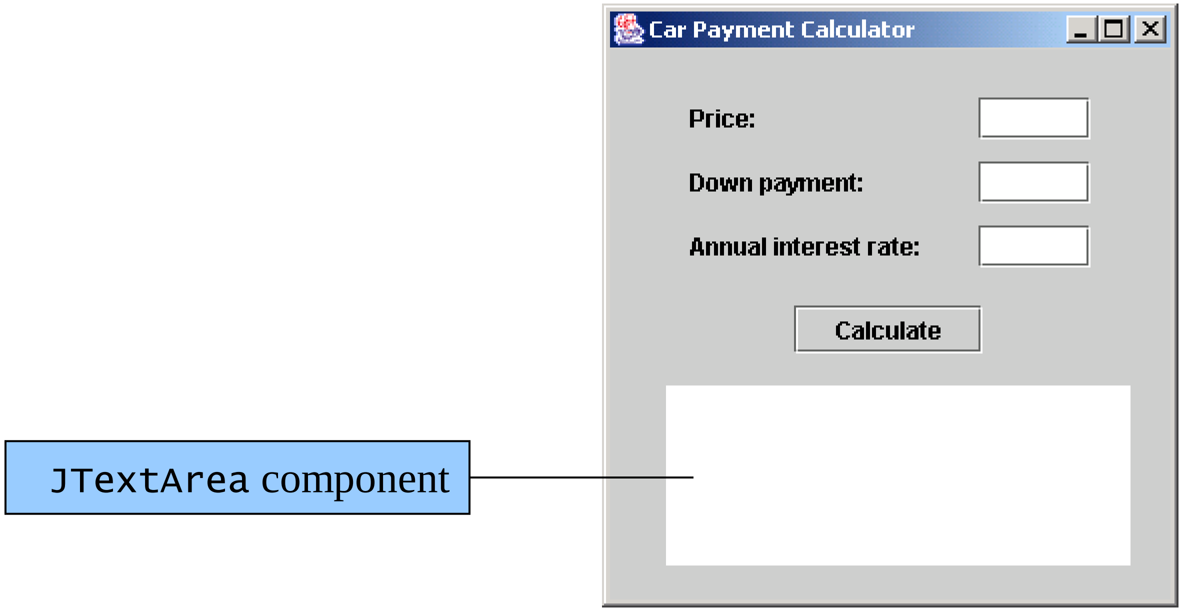
Customizing `paymentsJTextArea`



```
// set up paymentsJTextArea
paymentsJTextArea = new JTextArea();
paymentsJTextArea.setBounds( 28, 168, 232, 90 );
paymentsJTextArea.setEditable( false );
contentPane.add( paymentsJTextArea );
```

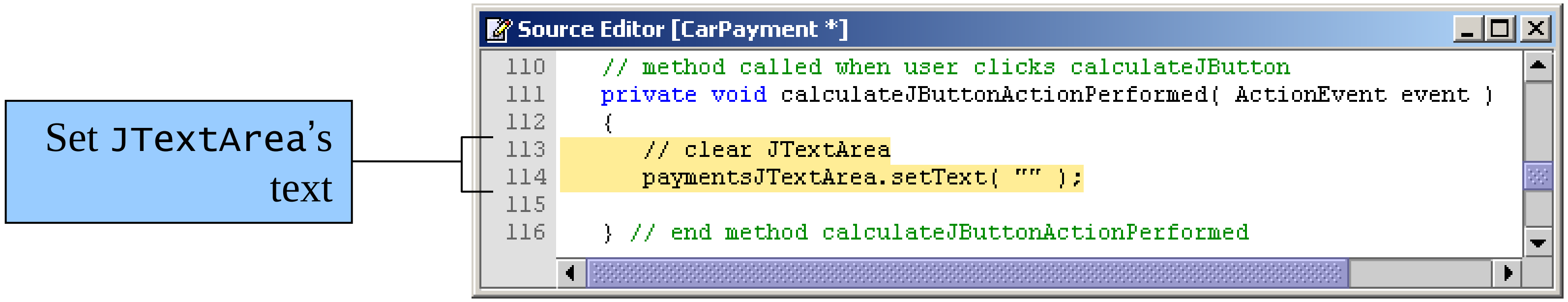
8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.8 `JTextArea` added to **Car Payment Calculator** application's `JFrame`.



8.4 Constructing the **Car Payment Calculator** Application (Cont.)

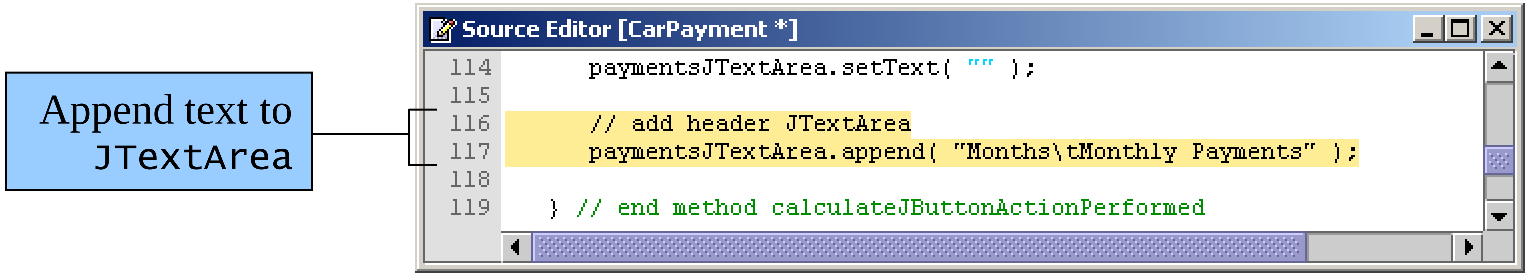
Figure 8.9 Clearing the contents of a `JTextArea`.



```
   // method called when user clicks calculateJButton
   private void calculateJButtonActionPerformed( ActionEvent event )
   {
      // clear JTextArea
      paymentsJTextArea.setText( "" );

   } // end method calculateJButtonActionPerformed
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.10 Adding a header to a `JTextArea`.

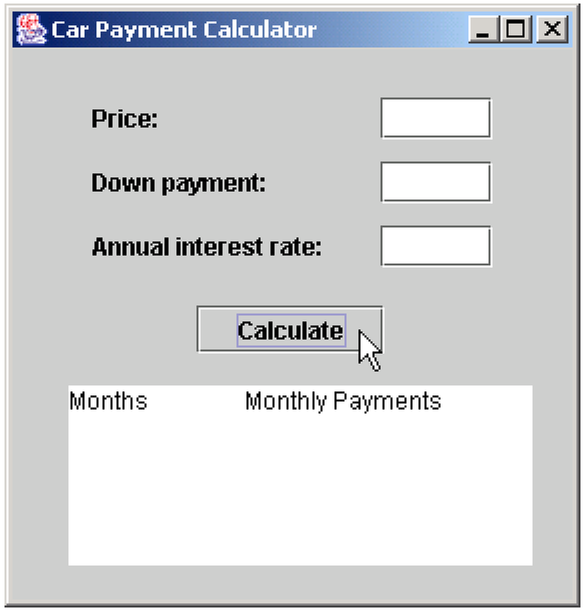


```
        paymentsJTextArea.setText( "" );

        // add header JTextArea
        paymentsJTextArea.append( "Months\tMonthly Payments" );

    } // end method calculateJButtonActionPerformed
```

- Use the `append` method to add text to a `JTextArea`
- Escape character (\)
  - Combines with the next character to form an escape sequence
  - `\t` – tab character

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

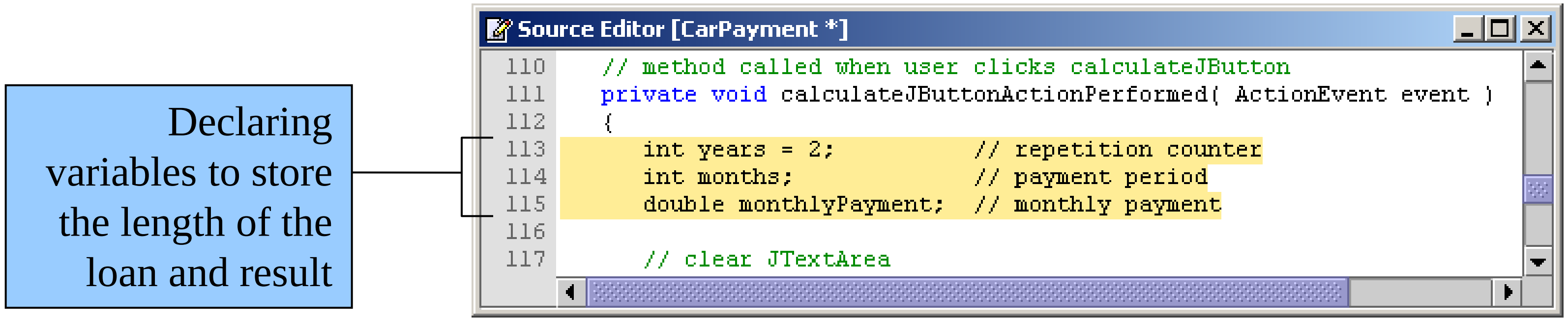
Figure 8.11 Header displayed in the `JTextArea`.



```
Months          Monthly Payments
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

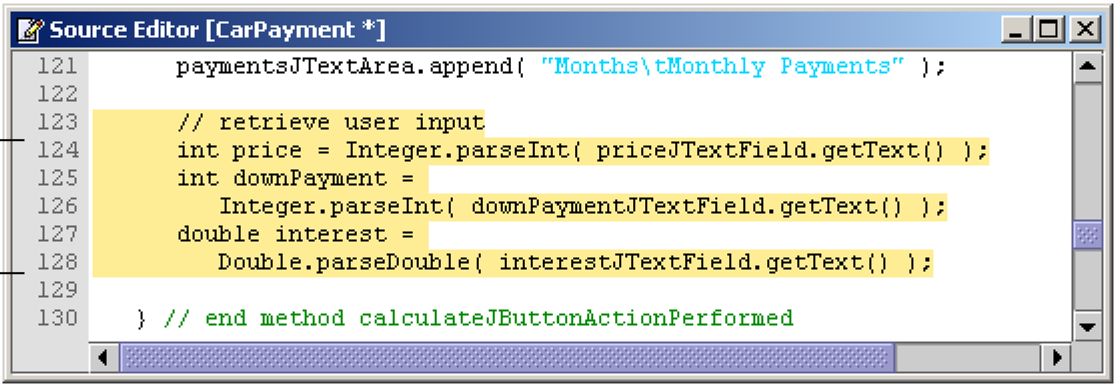
Figure 8.12 Variables for the **Car Payment Calculator** application.


Declaring variables to store the length of the loan and result
Source Editor [CarPayment *]

```
   // method called when user clicks calculateJButton
   private void calculateJButtonActionPerformed( ActionEvent event )
   {
      int years = 2;            // repetition counter
      int months;               // payment period
      double monthlyPayment;    // monthly payment

      // clear JTextArea
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

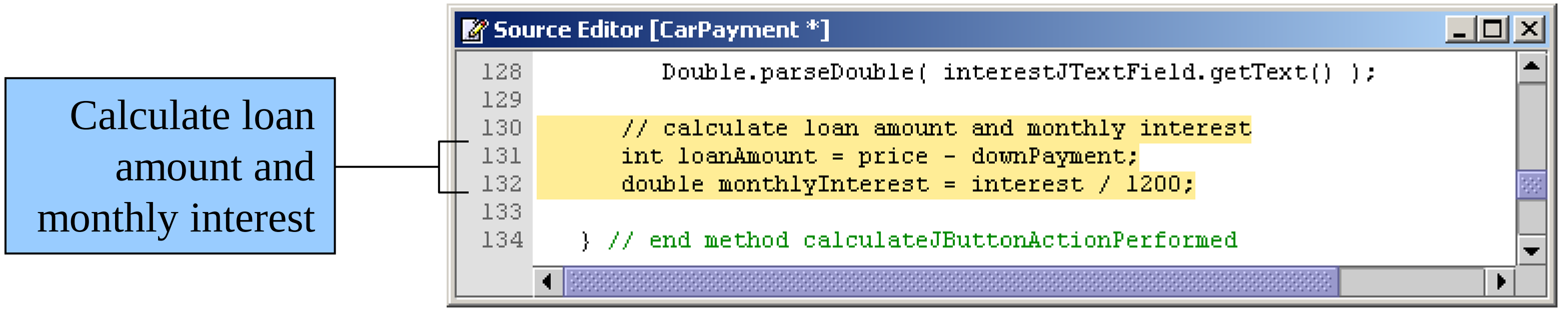
Figure 8.13 Retrieving input in the **Car Payment Calculator** application.



```
      paymentsJTextArea.append( "Months\tMonthly Payments" );

      // retrieve user input
      int price = Integer.parseInt( priceJTextField.getText() );
      int downPayment =
         Integer.parseInt( downPaymentJTextField.getText() );
      double interest =
         Double.parseDouble( interestJTextField.getText() );

   } // end method calculateJButtonActionPerformed
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.14 Determining amount borrowed and monthly interest rate.

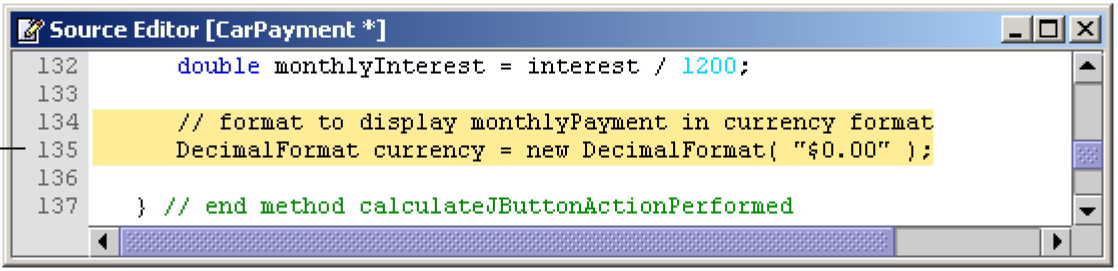

Calculate loan amount and monthly interest
Source Editor [CarPayment *]

```
            Double.parseDouble( interestJTextField.getText() );

         // calculate loan amount and monthly interest
         int loanAmount = price - downPayment;
         double monthlyInterest = interest / 1200;

      } // end method calculateJButtonActionPerformed
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.15 Declaring `DecimalFormat` `currency` for displaying the result in currency format.

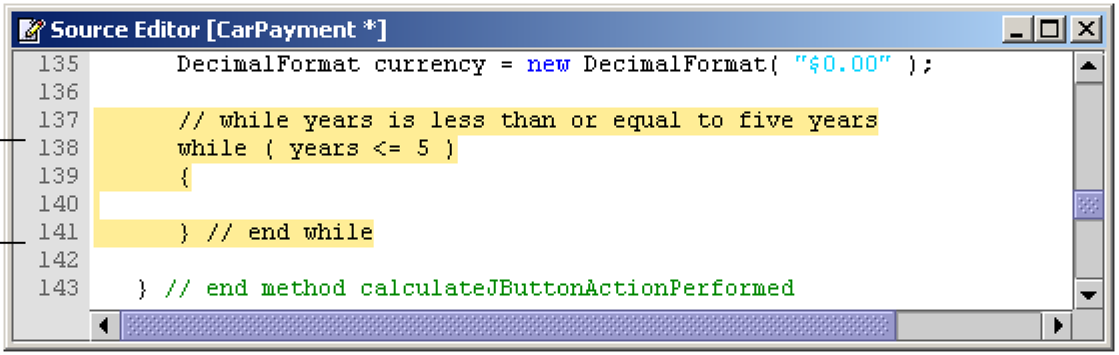
Create `DecimalFormat` for dollar amounts



```
      double monthlyInterest = interest / 1200;

      // format to display monthlyPayment in currency format
      DecimalFormat currency = new DecimalFormat( "$0.00" );

   } // end method calculateJButtonActionPerformed
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.16 Adding the `while` statement.

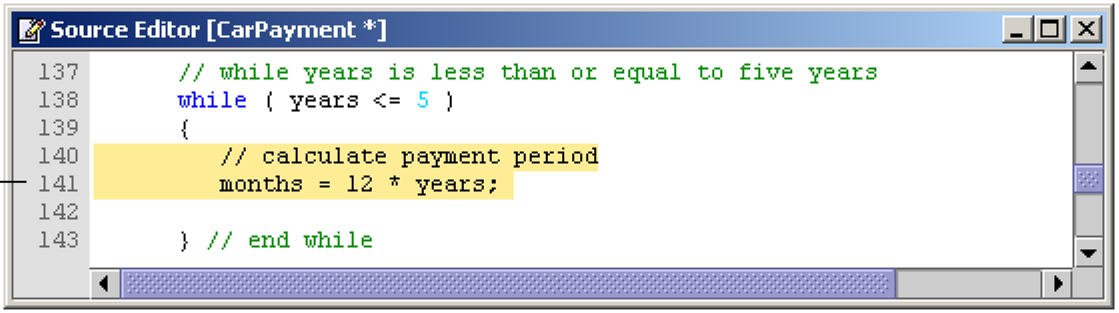


```
      DecimalFormat currency = new DecimalFormat( "$0.00" );

      // while years is less than or equal to five years
      while ( years <= 5 )
      {

      } // end while

   } // end method calculateJButtonActionPerformed
```

- **Counter-controlled** repetition
  - **Counter** controls the number of times a set of statements will execute
  - Also known as **definite repetition** because number of repetitions is known prior to execution

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.17 Converting the loan duration from years to months.



```
      // while years is less than or equal to five years
      while ( years <= 5 )
      {
         // calculate payment period
         months = 12 * years;

      } // end while
```

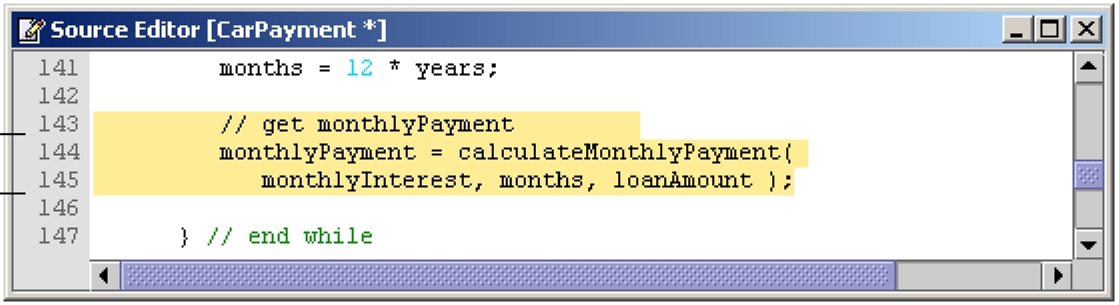
Determine number of months in loan period

- `months` is set to a different value, depending on which iteration of the loop is being executed

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.18 Method `calculateMonthlyPayment` returns monthly payment.

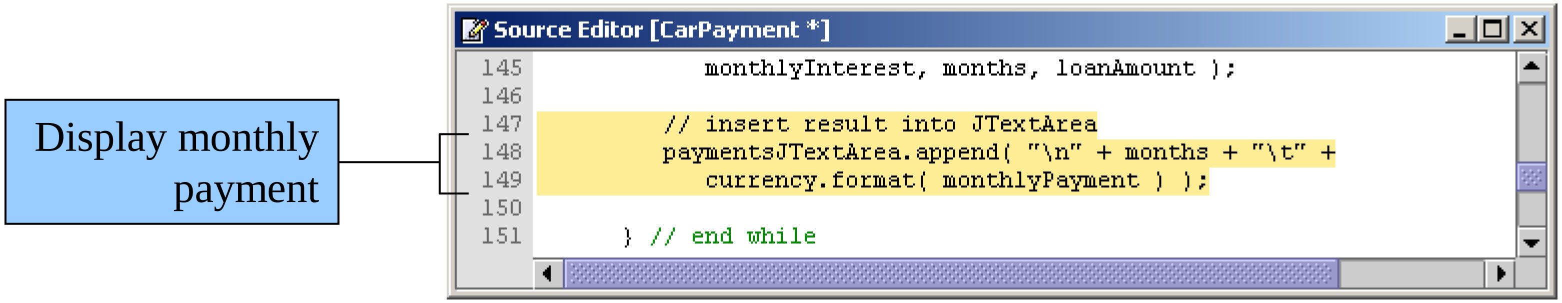
Calculate monthly payment



```
         months = 12 * years;

         // get monthlyPayment
         monthlyPayment = calculateMonthlyPayment(
            monthlyInterest, months, loanAmount );

      } // end while
```

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.19 Displaying the number of months and the amount of each monthly payment.

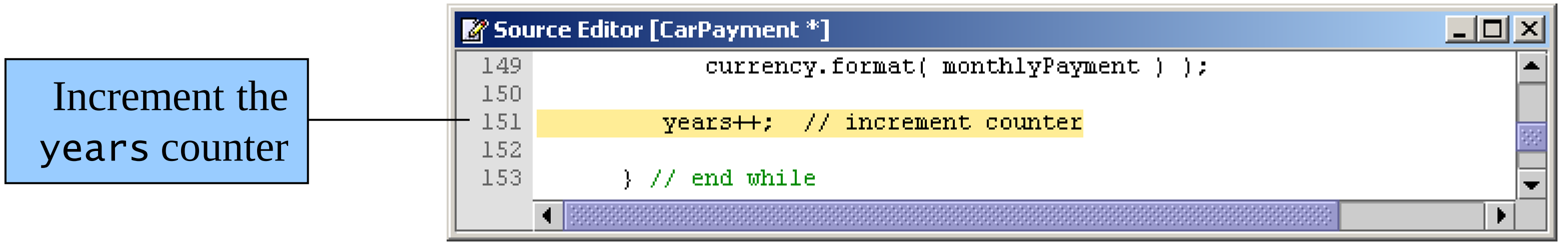


```
                   monthlyInterest, months, loanAmount );

              // insert result into JTextArea
              paymentsJTextArea.append( "\n" + months + "\t" +
                 currency.format( monthlyPayment ) );

         } // end while
```

- String-concatenation operator
- Escape sequence \n – newline character

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.20 Incrementing the counter.

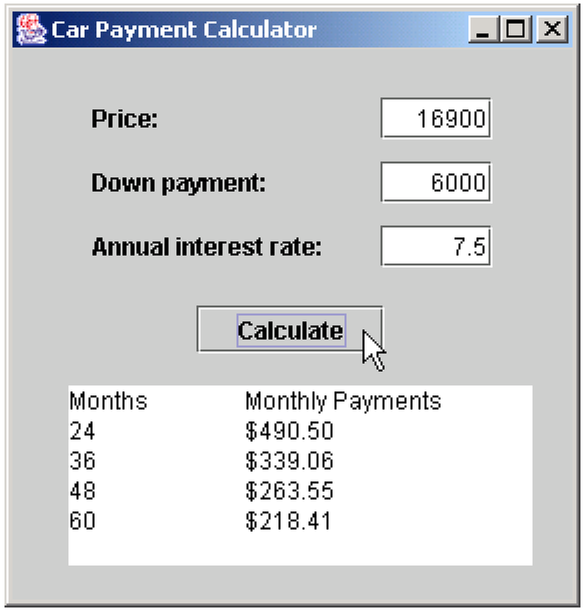


```
            currency.format( monthlyPayment ) );

         years++;  // increment counter

      } // end while
```

- The counter is incremented at the end of the loop so the loop will eventually end

8.4 Constructing the **Car Payment Calculator** Application (Cont.)

Figure 8.21 Running the completed application.



Price: 16900
Down payment: 6000
Annual interest rate: 7.5

```
Months          Monthly Payments
24              $490.50
36              $339.06
48              $263.55
60              $218.41
```

```java
// Tutorial 8: CarPayment.java
// Calculate different billing plans for a car loan.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.text.DecimalFormat;

public class CarPayment extends JFrame
{
   // JLabel and JTextfield for price
   private JLabel priceJLabel;
   private JTextField priceJTextField;

   // JLabel and JTextfield for down payment
   private JLabel downPaymentJLabel;
   private JTextField downPaymentJTextField;

   // JLabel and JTextfield for interest
   private JLabel interestJLabel;
   private JTextField interestJTextField;

   // JButton to initiate calculation
   private JButton calculateJButton;

```

CarPayment.java (1 of 8)

```
   // JTextArea to display results
   private JTextArea paymentsJTextArea;

   // no-argument constructor
   public CarPayment()
   {
      createUserInterface();
   }

   // create and position GUI components; register event handlers
   private void createUserInterface()
   {
      // get content pane and set layout to null
      Container contentPane = getContentPane();
      contentPane.setLayout( null );

      // set up priceJLabel
      priceJLabel = new JLabel();
      priceJLabel.setBounds( 40, 24, 80, 21 );
      priceJLabel.setText( "Price:" );
      contentPane.add( priceJLabel );

```

CarPayment.java
(2 of 8)

Outline

```
      // set up priceJTextField
      priceJTextField = new JTextField();
      priceJTextField.setBounds( 184, 24, 56, 21 );
      priceJTextField.setHorizontalAlignment( JTextField.RIGHT );
      contentPane.add( priceJTextField );

      // set up downPaymentJLabel
      downPaymentJLabel = new JLabel();
      downPaymentJLabel.setBounds( 40, 56, 96, 21 );
      downPaymentJLabel.setText( "Down payment:" );
      contentPane.add( downPaymentJLabel );

      // set up downPaymentJTextField
      downPaymentJTextField = new JTextField();
      downPaymentJTextField.setBounds( 184, 56, 56, 21 );
      downPaymentJTextField.setHorizontalAlignment(
         JTextField.RIGHT );
      contentPane.add( downPaymentJTextField );

      // set up interestJLabel
      interestJLabel = new JLabel();
      interestJLabel.setBounds( 40, 88, 120, 21 );
      interestJLabel.setText( "Annual interest rate:" );
      contentPane.add( interestJLabel );

```

CarPayment.java
(3 of 8)

Outline

CarPayment.java
(4 of 8)

```
      // set up interestJTextField
      interestJTextField = new JTextField();
      interestJTextField.setBounds( 184, 88, 56, 21 );
      interestJTextField.setHorizontalAlignment( JTextField.RIGHT );
      contentPane.add( interestJTextField );

      // set up calculateJButton and register its event handler
      calculateJButton = new JButton();
      calculateJButton.setBounds( 92, 128, 94, 24 );
      calculateJButton.setText( "Calculate" );
      contentPane.add( calculateJButton );
      calculateJButton.addActionListener(

         new ActionListener() // anonymous inner class
         {
            // event handler called when user clicks calculateJButton
            public void actionPerformed( ActionEvent event )
            {
               calculateJButtonActionPerformed( event );
            }

         } // end anonymous inner class

      ); // end call to addActionListener

      // set up paymentsJTextArea
      paymentsJTextArea = new JTextArea();
      paymentsJTextArea.setBounds( 28, 168, 232, 90 );
      paymentsJTextArea.setEditable( false );
      contentPane.add( paymentsJTextArea );

      // set properties of window
      setTitle( "Car Payment Calculator" ); // set window's title
      setSize( 288, 302 );                  // set window's size
      setVisible( true );                   // display window

   } // end method createUserInterface

   // method called when user clicks calculateJButton
   private void calculateJButtonActionPerformed( ActionEvent event )
   {
      int years = 2;             // repetition counter
      int months;                // payment period
      double monthlyPayment;     // monthly payment

      // clear JTextArea
      paymentsJTextArea.setText( "" );

```

Outline

CarPayment.java (5 of 8)

Customizing the `JTextArea`

Declaring variables

Clearing `JTextArea`

Outline

CarPayment.java (6 of 8)

```
      // add header JTextArea
      paymentsJTextArea.append( "Months\tMonthly Payments" );

      // retrieve user input
      int price = Integer.parseInt( priceJTextField.getText() );
      int downPayment =
         Integer.parseInt( downPaymentJTextField.getText() );
      double interest =
         Double.parseDouble( interestJTextField.getText() );

      // calculate loan amount and monthly interest
      int loanAmount = price - downPayment;
      double monthlyInterest = interest / 1200;

      // format to display monthlyPayment in currency format
      DecimalFormat currency = new DecimalFormat( "$0.00" );

      // while years is less than or equal to five years
      while ( years <= 5 )
      {
         // calculate payment period
         months = 12 * years;

```

Adding header to `JTextArea`

Obtaining user input

Calculating loan amount and monthly interest

Declaring `DecimalFormat`

Outline

CarPayment.java (7 of 8)

```
         // get monthlyPayment
         monthlyPayment = calculateMonthlyPayment(
            monthlyInterest, months, loanAmount );

         // insert result into JTextArea
         paymentsJTextArea.append( "\n" + months + "\t" +
            currency.format( monthlyPayment ) );

         years++;  // increment counter

      } // end while

   } // end method calculateJButtonActionPerformed

   // calculate monthlyPayment
   private double calculateMonthlyPayment( double monthlyInterest,
      int months, int loanAmount )
   {
      double base = Math.pow( 1 + monthlyInterest, months );
      return loanAmount * monthlyInterest / ( 1 - ( 1 / base ) );
   }

```

Calling method `calculateMonthlyPayment` to get monthly payment

Incrementing counter

Outline

CarPayment.java (8 of 8)

```
   // main method
   public static void main( String [] args )
   {
      CarPayment application = new CarPayment();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

   } // end method main

} // end class CarPayment
```