Programming Logic and Design
Ninth Edition

Chapter 12

Event-Driven GUI Programming, Multithreading, and Animation

Objectives

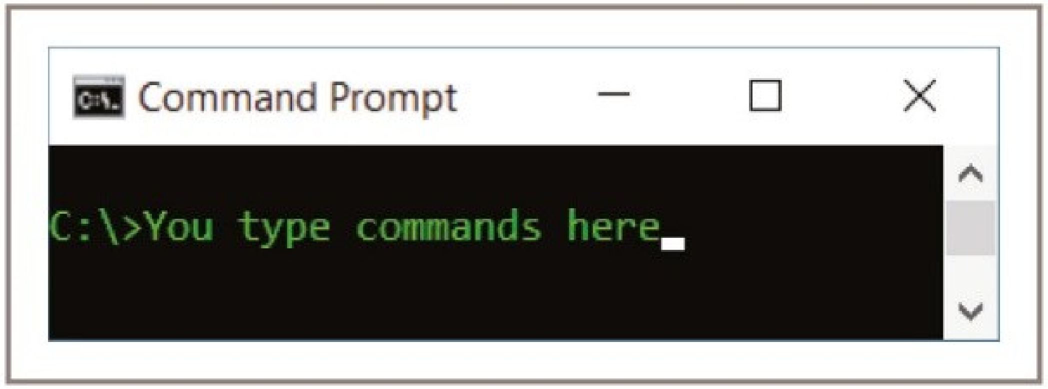
In this chapter, you will learn about:

- The principles of event-driven programming
- User-initiated actions and GUI components
- Designing graphical user interfaces
- Developing an event-driven application
- Threads and multithreading
- Creating animation

Understanding Event-Driven Programming

- **Operating system**
  - The software used to run a computer and manage its resources
- Early days of computing
  - Command line entry: the **DOS prompt**
  - Interacting with a computer operating system was difficult
  - The user had to know the exact syntax to use when typing commands

Understanding Event-Driven Programming (continued -1)



```
C:\>You type commands here_
```

Figure 12-1 Command prompt screen

Understanding Event-Driven Programming (continued -2)

- Modern operating system software
  - Allows use of mouse, fingers or other pointing device to select **icons**
  - GUI
- Computer users can expect to see a standard interface
- **Event**
  - Generates a message sent to an object

Understanding Event-Driven Programming (continued -3)

- **Event-driven** or **event-based** program
  - Actions occur in response to user-initiated events such as clicking a mouse button or tapping a screen
  - Emphasis is on the objects that the user can manipulate
  - Programmer writes instructions within modules that execute each type of event

Understanding Event-Driven Programming

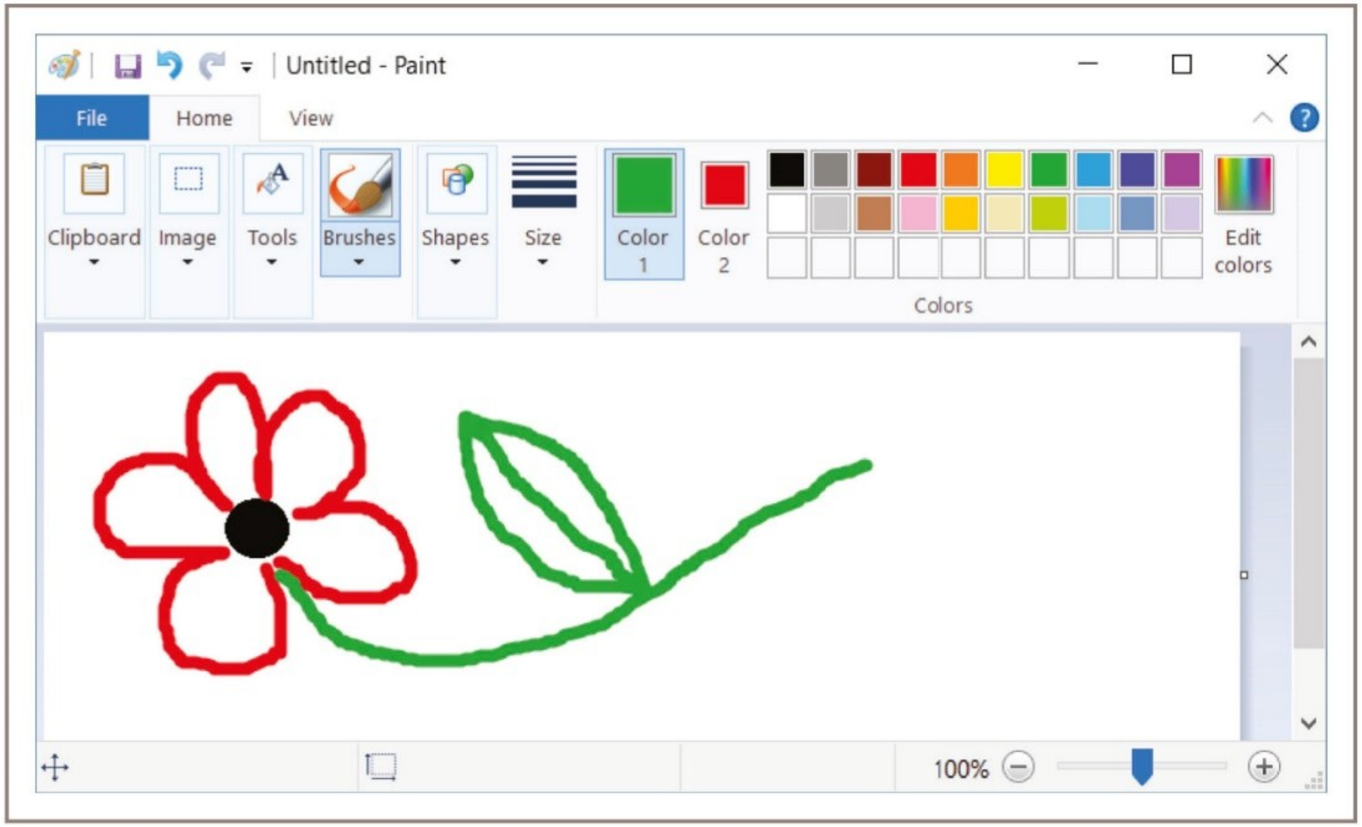


Figure 12-2 A GUI application that contains buttons and icons

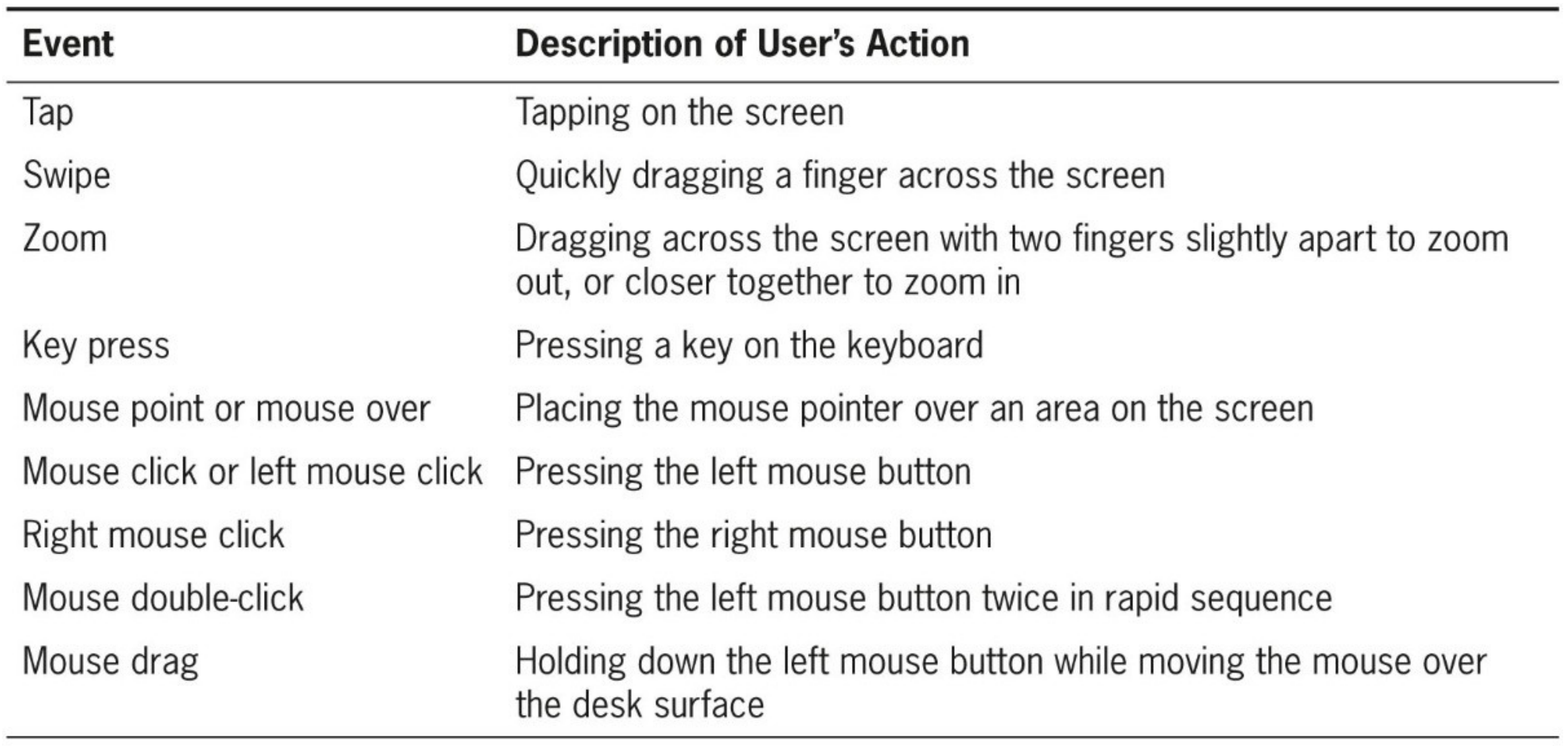
Understanding Event-Driven Programming (continued -5)

- Procedural application
  - Programmer controls order of program statements
- Event-driven programs
  - User might initiate any number of events in any order
  - More flexible than procedural counterparts
- **Source of the event**
  - The component from which an event is generated
- **Listener**
  - An object that is “interested in” an event to

Understanding Event-Driven Programming (continued -6)

- Not all objects can receive all events
- Event-driven programming
  - Newer than procedural programming
  - But uses many similar techniques to procedural
- In object-oriented languages, the procedural modules that depend on user-initiated events are called **scripts**

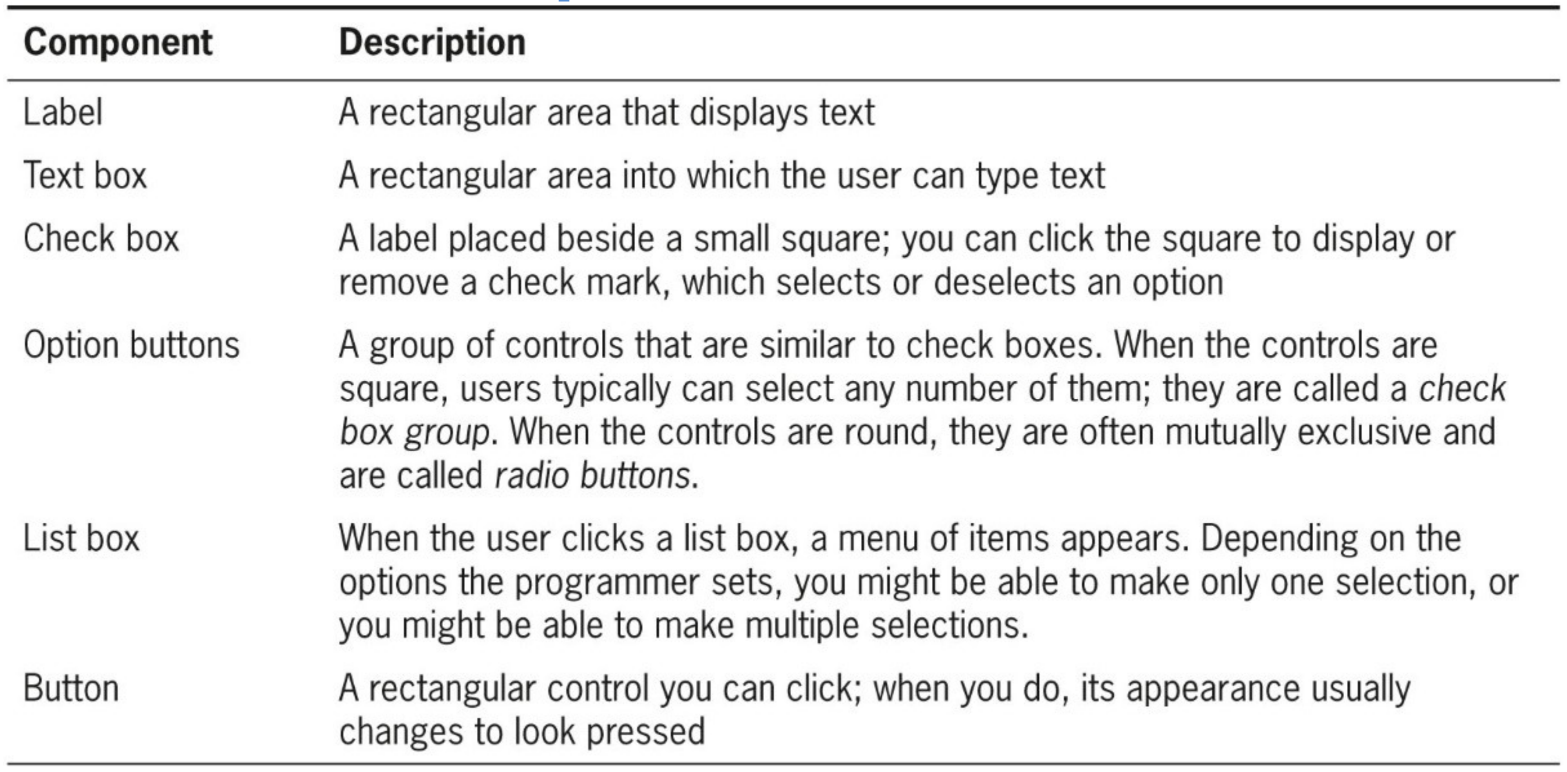
User-Initiated Actions and GUI Components



Event	Description of User's Action
Tap	Tapping on the screen
Swipe	Quickly dragging a finger across the screen
Zoom	Dragging across the screen with two fingers slightly apart to zoom out, or closer together to zoom in
Key press	Pressing a key on the keyboard
Mouse point or mouse over	Placing the mouse pointer over an area on the screen
Mouse click or left mouse click	Pressing the left mouse button
Right mouse click	Pressing the right mouse button
Mouse double-click	Pressing the left mouse button twice in rapid sequence
Mouse drag	Holding down the left mouse button while moving the mouse over the desk surface

Table 12-1 Common user-initiated events

User-Initiated Actions and GUI Components (continued -1)



Component	Description
Label	A rectangular area that displays text
Text box	A rectangular area into which the user can type text
Check box	A label placed beside a small square; you can click the square to display or remove a check mark, which selects or deselects an option
Option buttons	A group of controls that are similar to check boxes. When the controls are square, users typically can select any number of them; they are called a *check box group*. When the controls are round, they are often mutually exclusive and are called *radio buttons*.
List box	When the user clicks a list box, a menu of items appears. Depending on the options the programmer sets, you might be able to make only one selection, or you might be able to make multiple selections.
Button	A rectangular control you can click; when you do, its appearance usually changes to look pressed

Table 12-2 Common GUI components

User-Initiated Actions and GUI Components (continued -2)

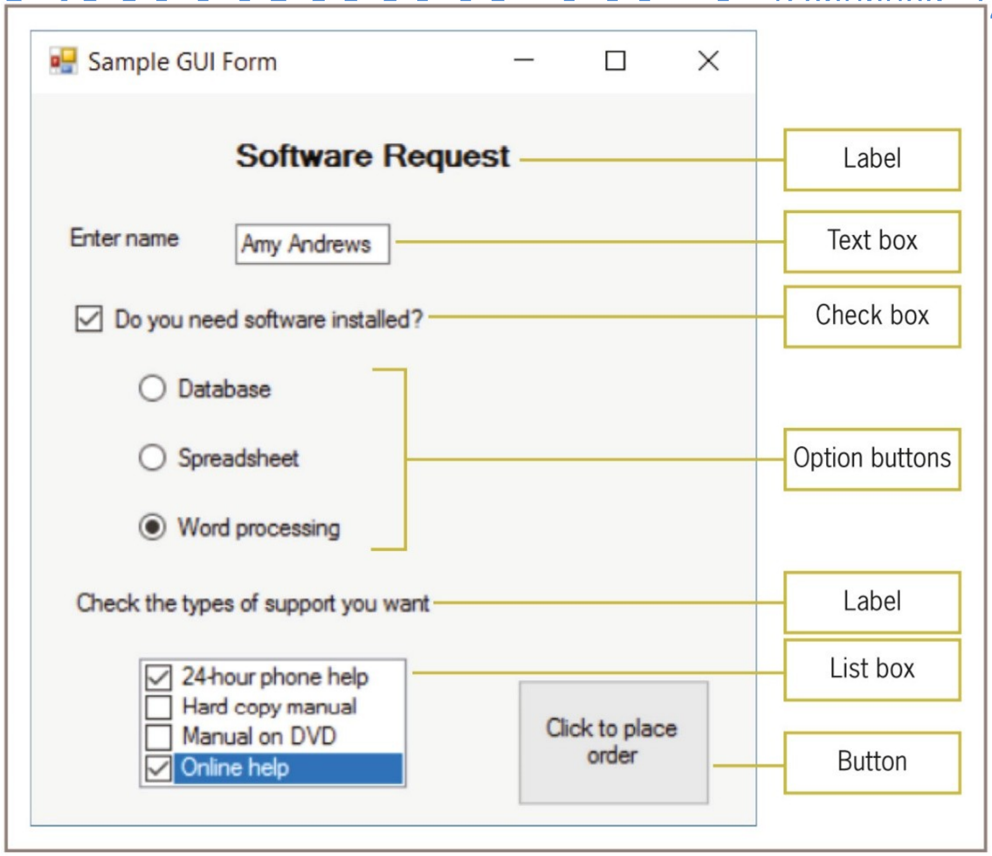
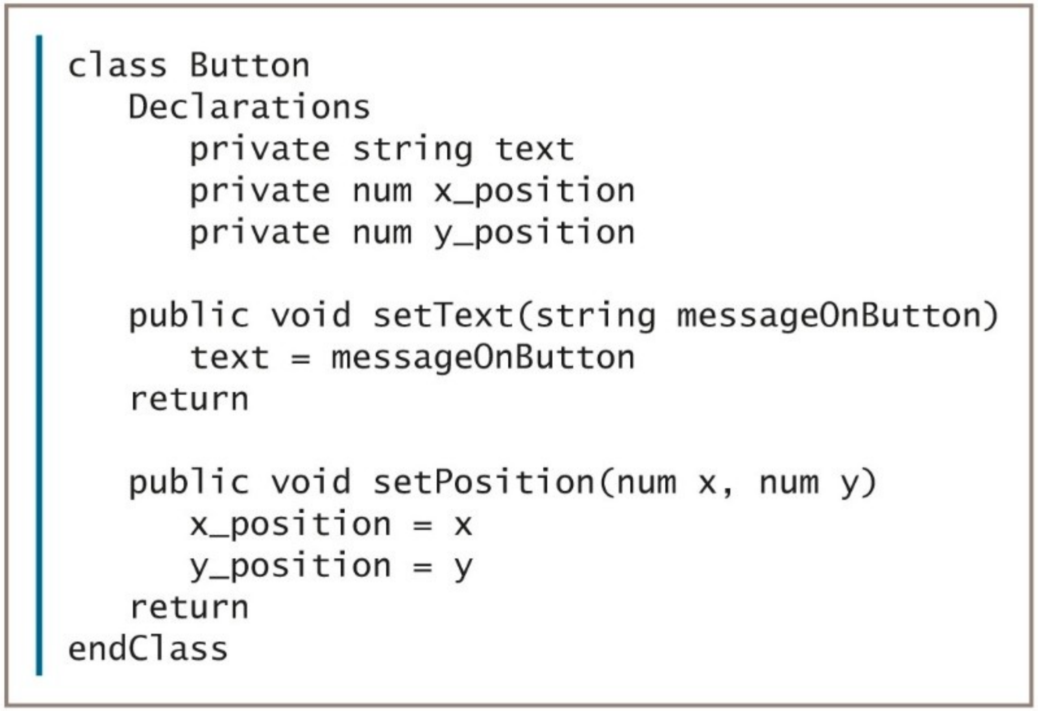


Figure 12-3 Common GUI components

User-Initiated Actions and GUI Components (continued -3)

- Do not create the GUI components you need from scratch
  - Call prewritten methods that draw the GUI components on the screen for you
- Components themselves are constructed using existing classes complete with names, attributes, and methods
- Text or graphical environment

User-Initiated Actions and GUI Components (continued -4)



```
class Button
   Declarations
      private string text
      private num x_position
      private num y_position

   public void setText(string messageOnButton)
      text = messageOnButton
   return

   public void setPosition(num x, num y)
      x_position = x
      y_position = y
   return
endClass
```

Figure 12-4 Button class

User-Initiated Actions and GUI Components (continued -5)

- The x_position and y_position of the Button object refer to horizontal and vertical coordinates where the Button appears on the screen
- A **pixel** is one of the tiny dots of light that form a grid on your screen
- Together, the *x* and *y* coordinates locate any specific pixel location on the screen

User-Initiated Actions and GUI Components (continued -6)

- Create a `Button` object
  - `Button myProgramButton`
- Use `Button`’s methods
  - `myProgramButton.setText("Click here")`
  - `myProgramButton.setPosition(10, 30)`
- Different GUI classes support different attributes and methods

Designing Graphical User Interfaces

- The interface should be natural and predictable
- The interface should be attractive, easy to read, and nondistracting
- To some extent, it's helpful if the user can customize your applications
- The program should be forgiving
- The GUI is only a means to an end

The Interface Should Be Natural and Predictable

- Represent objects like their real-world counterparts
- Actions should be predictable
  - Similar to other programs
- Be predictable in layout
  - Put related items together in proper order
  - Example: locate menu at top

Attractive,
Easy to Read, and
Nondistracting

- Attractive
  - People are more likely to use it
- Easy to read
  - Less likely to make mistakes
- Screen designs should not be distracting
  - Fancy fonts and weird color combinations are the signs of amateur designers

To Some Extent, It’s Helpful If the User Can Customize Your Applications

- It’s helpful if users can position components in the order that’s easiest for them
- Users appreciate being able to change features like color schemes
- **Accessibility**
  - Screen design issues that make programs easier to use for people with physical limitations
- Possibly allow for choice of language and currency

The Program Should Be Forgiving

- Always provide an escape route
  - Accommodate users who make bad choices or change their minds
  - Back button or functional Escape key
  - Option to revert to the default settings
  - Allow users to perform tasks in a variety of ways:
    - Mouse click
    - Touch
    - Keyboard shortcut
    - Menu item
    - Disability options

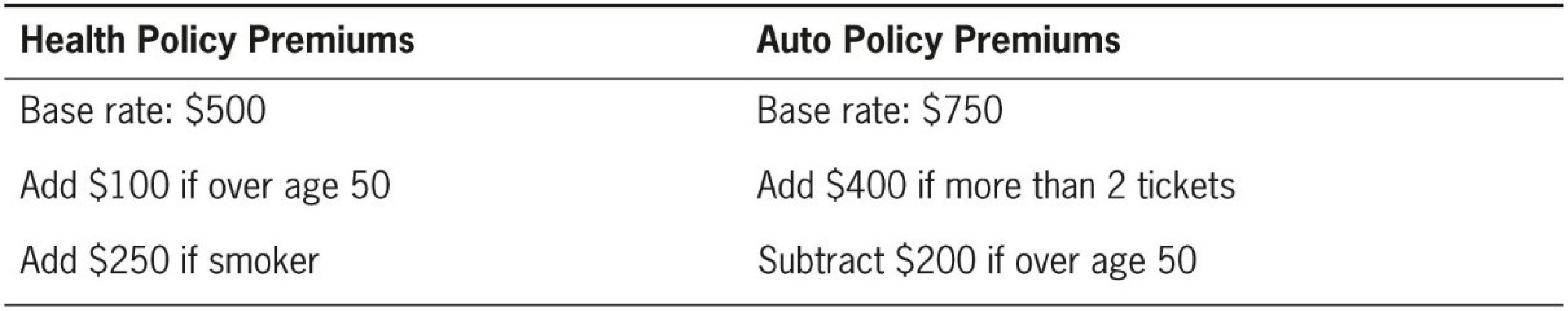
The GUI Is Only a Means to an End

- The GUI is only an interface
- The point of a graphical interface is to help people be more productive
- The real work of any GUI program is done after the user clicks a button or makes a list box selection

Developing an Event-Driven Application

- Steps to developing an event-driven application
  1. Understanding the problem
  2. Planning the logic
     - 2a. Creating wireframes
     - 2b. Creating storyboards
     - 2c. Defining the objects
     - 2d. Defining the connections between the screens the user will see
  3. Coding the program
  4. Translating the program into machine language
  5. Testing the program
  6. Putting the program into production

Developing an Event-Driven Application (continued)



Health Policy Premiums	Auto Policy Premiums
Base rate: $500	Base rate: $750
Add $100 if over age 50	Add $400 if more than 2 tickets
Add $250 if smoker	Subtract $200 if over age 50

Table 12-3 Insurance premiums based on customer characteristics

Creating Wireframes

- **Wireframe**
  - A picture or sketch of a screen the user will see when running a program
  - Also called a **page schematic** or **screen blueprint**
  - Can be pencil sketches or produced by software applications

Creating Storyboards

- **Storyboard**
  - Contains a series of wireframes that represent a user’s experience with the proposed software
- GUI storyboards
  - Represent “snapshot” views of the screens the user will encounter during the run of a program

Creating Storyboards (continued)

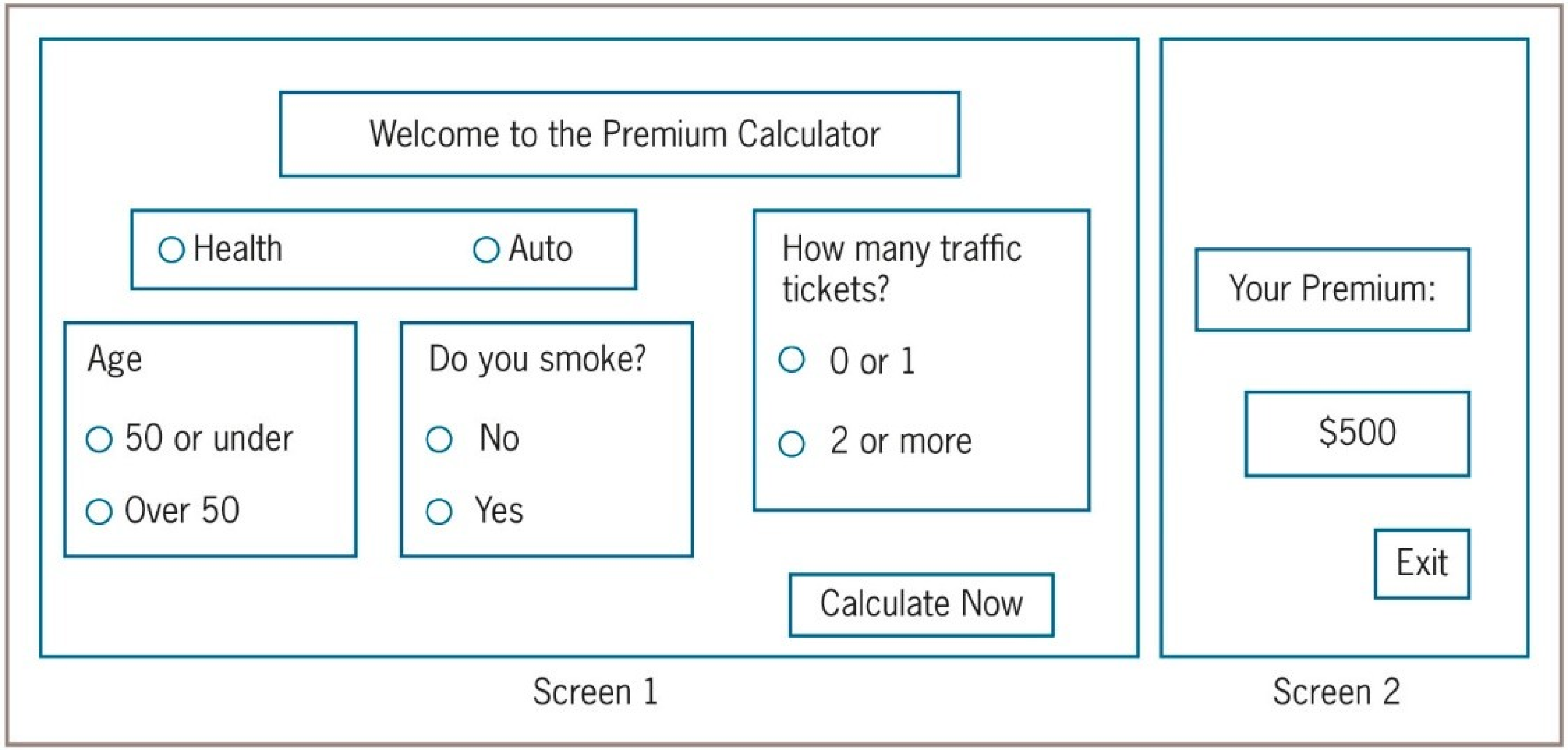
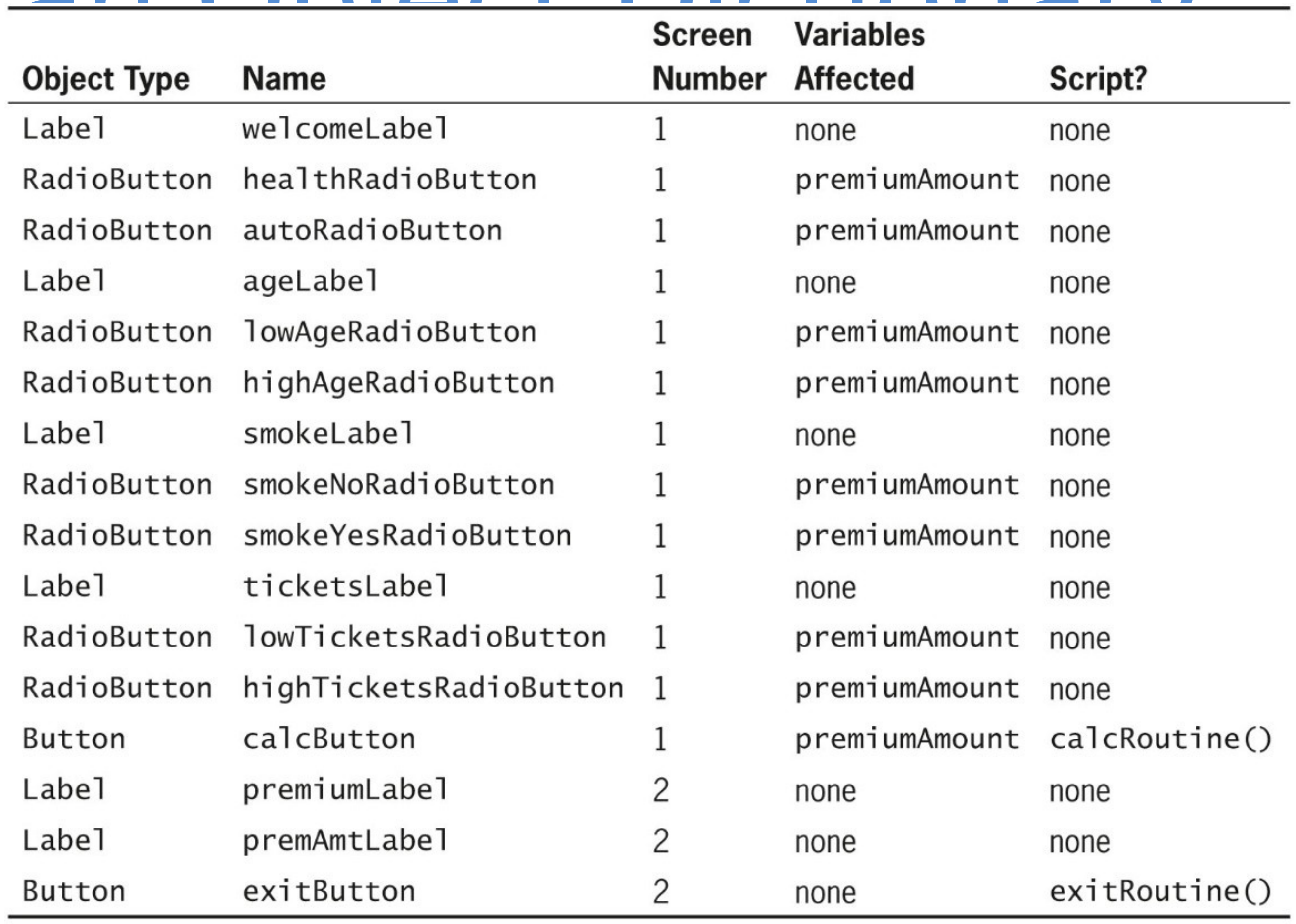


Figure 12-5 Storyboard for the insurance program, which is composed of two wireframes

Defining the Storyboard Objects in an Object Dictionary

- An event-driven program may contain dozens or even hundreds of objects
- **Object dictionary**
  - A list of the objects used in a program
  - Includes which screens they are used on and whether any code, or script, is associated with them

Defining the Storyboard Objects in an Object Dictionary ntinued)



Object Type	Name	Screen Number	Variables Affected	Script?
Label	welcomeLabel	1	none	none
RadioButton	healthRadioButton	1	premiumAmount	none
RadioButton	autoRadioButton	1	premiumAmount	none
Label	ageLabel	1	none	none
RadioButton	lowAgeRadioButton	1	premiumAmount	none
RadioButton	highAgeRadioButton	1	premiumAmount	none
Label	smokeLabel	1	none	none
RadioButton	smokeNoRadioButton	1	premiumAmount	none
RadioButton	smokeYesRadioButton	1	premiumAmount	none
Label	ticketsLabel	1	none	none
RadioButton	lowTicketsRadioButton	1	premiumAmount	none
RadioButton	highTicketsRadioButton	1	premiumAmount	none
Button	calcButton	1	premiumAmount	calcRoutine()
Label	premiumLabel	2	none	none
Label	premAmtLabel	2	none	none
Button	exitButton	2	none	exitRoutine()

Figure 12-6 Object dictionary for insurance premium program

Defining Connections Between the User Screens

- Draw the connections between the screens to show how they interact
- **Interactivity diagram**
  - Shows the relationship between screens in an interactive GUI program
- One screen may lead to different screens depending on the options the user selects at any one screen

Defining Connections Between the User Screens (continued)

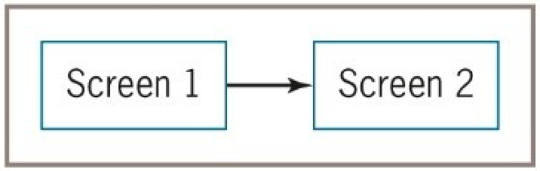


Figure 12-7 Interactivity diagram for insurance premium program

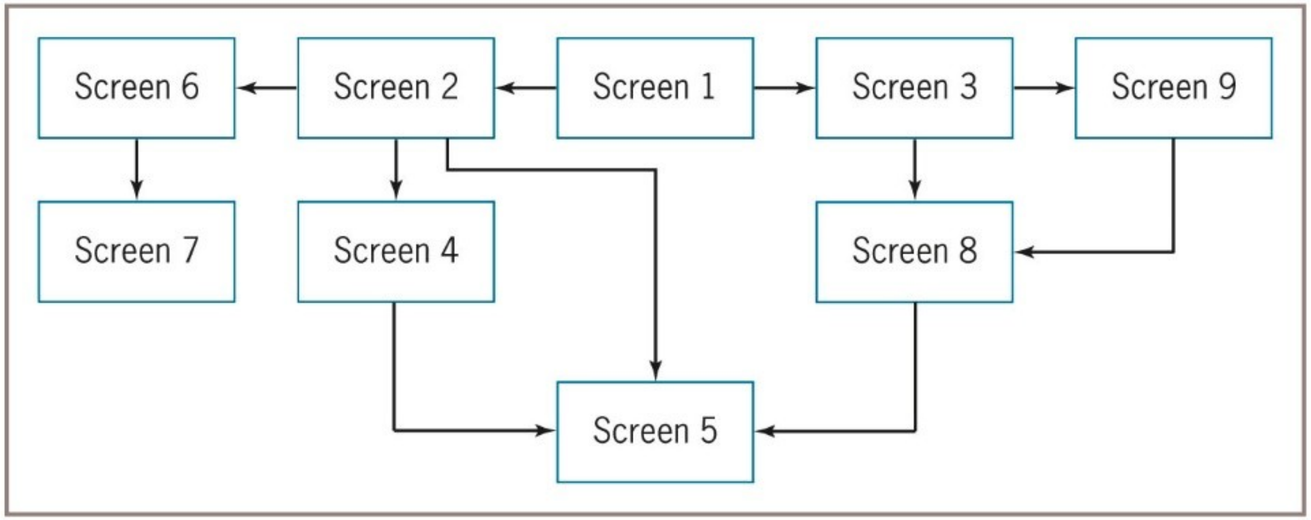


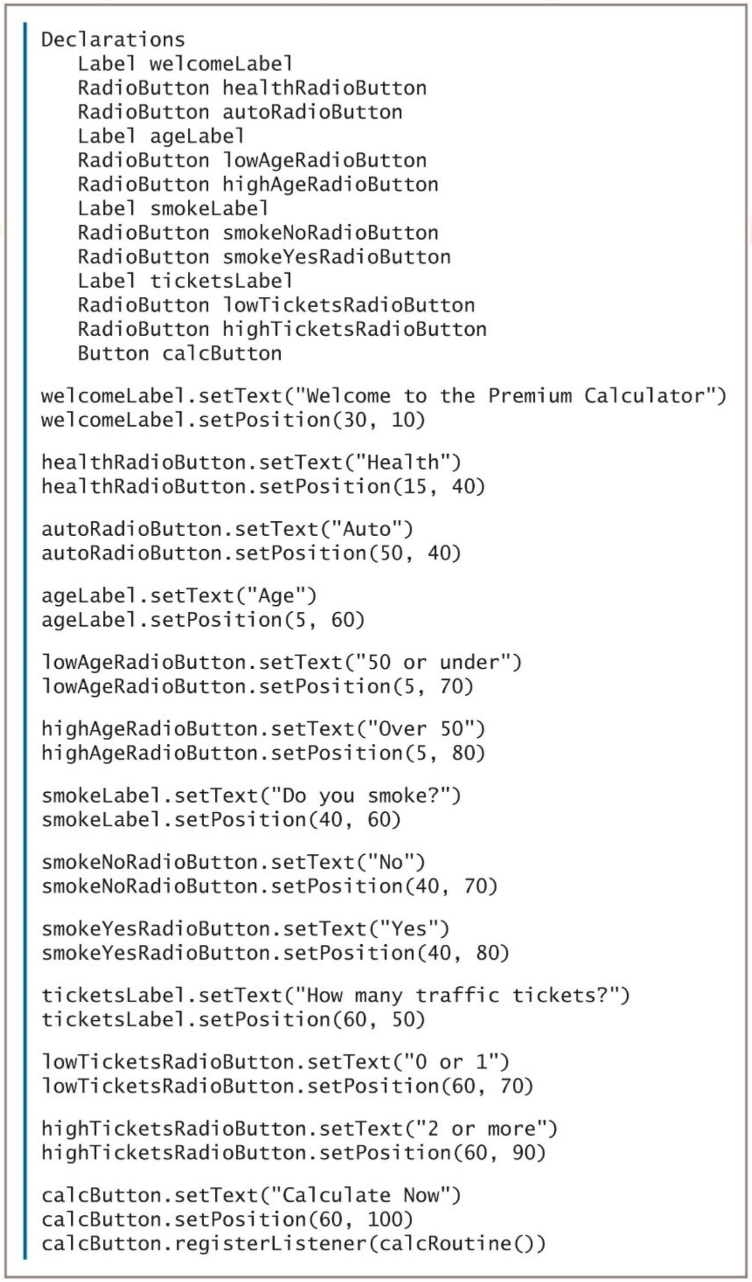
Figure 12-8 Interactivity diagram for a complicated program

Planning the Logic

- Design the screens
- Define the objects
- Define how the screens will connect
- Then start to plan the client program
- Create the component onto which all the GUI elements are placed
  - Might use a class with a name such as `Screen`, `Form`, or `Window`

Planning the Logic

(continued -1)



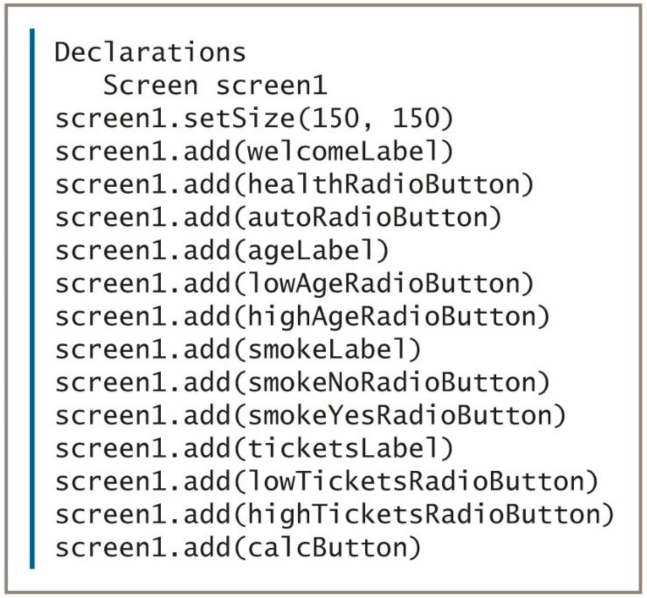
```
Declarations
   Label welcomeLabel
   RadioButton healthRadioButton
   RadioButton autoRadioButton
   Label ageLabel
   RadioButton lowAgeRadioButton
   RadioButton highAgeRadioButton
   Label smokeLabel
   RadioButton smokeNoRadioButton
   RadioButton smokeYesRadioButton
   Label ticketsLabel
   RadioButton lowTicketsRadioButton
   RadioButton highTicketsRadioButton
   Button calcButton

welcomeLabel.setText("Welcome to the Premium Calculator")
welcomeLabel.setPosition(30, 10)

healthRadioButton.setText("Health")
healthRadioButton.setPosition(15, 40)

autoRadioButton.setText("Auto")
autoRadioButton.setPosition(50, 40)

ageLabel.setText("Age")
ageLabel.setPosition(5, 60)

lowAgeRadioButton.setText("50 or under")
lowAgeRadioButton.setPosition(5, 70)

highAgeRadioButton.setText("Over 50")
highAgeRadioButton.setPosition(5, 80)

smokeLabel.setText("Do you smoke?")
smokeLabel.setPosition(40, 60)

smokeNoRadioButton.setText("No")
smokeNoRadioButton.setPosition(40, 70)

smokeYesRadioButton.setText("Yes")
smokeYesRadioButton.setPosition(40, 80)

ticketsLabel.setText("How many traffic tickets?")
ticketsLabel.setPosition(60, 50)

lowTicketsRadioButton.setText("0 or 1")
lowTicketsRadioButton.setPosition(60, 70)

highTicketsRadioButton.setText("2 or more")
highTicketsRadioButton.setPosition(60, 90)

calcButton.setText("Calculate Now")
calcButton.setPosition(60, 100)
calcButton.registerListener(calcRoutine())
```

Figure 12-9 Component definitions for first screen of insurance program

Planning the Logic (continued -2)

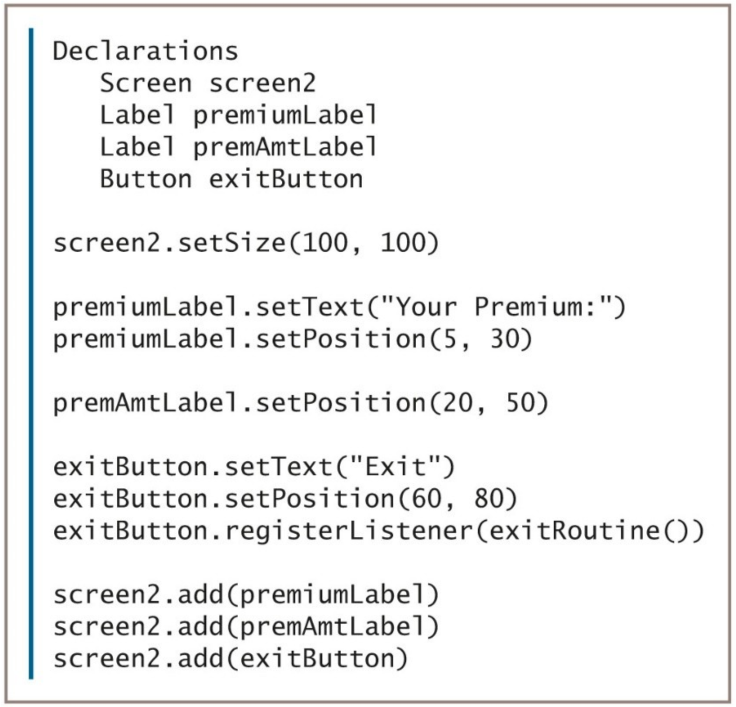
- **Register components**
  - Sign them up so that they can react to events initiated by other components
- **Container**
  - A class of objects whose main purpose is to hold other elements
  - Contains methods that allow you to set physical properties such as height and width
  - Contains methods that allow you to add the appropriate components to a container

Planning the Logic (continued -3)



```
Declarations
   Screen screen1
screen1.setSize(150, 150)
screen1.add(welcomeLabel)
screen1.add(healthRadioButton)
screen1.add(autoRadioButton)
screen1.add(ageLabel)
screen1.add(lowAgeRadioButton)
screen1.add(highAgeRadioButton)
screen1.add(smokeLabel)
screen1.add(smokeNoRadioButton)
screen1.add(smokeYesRadioButton)
screen1.add(ticketsLabel)
screen1.add(lowTicketsRadioButton)
screen1.add(highTicketsRadioButton)
screen1.add(calcButton)
```

Figure 12-10 Statements that create `screen1`



```
Declarations
   Screen screen2
   Label premiumLabel
   Label premAmtLabel
   Button exitButton

screen2.setSize(100, 100)

premiumLabel.setText("Your Premium:")
premiumLabel.setPosition(5, 30)

premAmtLabel.setPosition(20, 50)

exitButton.setText("Exit")
exitButton.setPosition(60, 80)
exitButton.registerListener(exitRoutine())

screen2.add(premiumLabel)
screen2.add(premAmtLabel)
screen2.add(exitButton)
```

Figure 12-11 Statements that define and create `screen2` and its components

Planning the Logic (continued -4)

```
public void calcRoutine()
   Declarations
      num HEALTH_AMT = 500
      num HIGH_AGE = 100
      num SMOKER = 250
      num AUTO_AMT = 750
      num HIGH_TICKETS = 400
      num HIGH_AGE_DRIVER_DISCOUNT = 200
      num premiumAmount
   if healthRadioButton.getChecked() then
      premiumAmount = HEALTH_AMT
      if highAgeRadioButton.getChecked() then
         premiumAmount = premiumAmount + HIGH_AGE
      endif
      if smokeYesRadioButton.getChecked() then
         premiumAmount = premiumAmount + SMOKER
      endif
   else
      premiumAmount = AUTO_AMT
      if highTicketsRadioButton.getChecked() then
         premiumAmount = premiumAmount + HIGH_TICKETS
      endif
      if highAgeRadioButton.getChecked() then
         premiumAmount = premiumAmount - HIGH_AGE_DRIVER_DISCOUNT
      endif
   endif
   premAmtLabel.setText(premiumAmount)
   screen1.remove()
   screen2.display()
return
```

Figure 12-12 Pseudocode for `calcRoutine()` method of insurance premium program

```
start
   screen1.display()
stop

public void exitRoutine()
   screen2.remove()
return
```

Figure 12-13 The main program and `exitRoutine()` method for the insurance program

Understanding Threads and Multithreading

- **Thread**
  - The flow of execution of one set of program statements
- Many applications follow a single thread
- A computer’s central processing unit (CPU) can execute only one statement at a time regardless of its processor speed
- A computer with multiple CPUs can execute multiple instructions simultaneously

Understanding Threads and Multithreading (continued -1)

- **Multithreading**
  - Using multiple threads of execution
  - Multiple threads share the CPU’s time for a single processor
- Use multithreading to improve the performance of your programs
  - More user friendly
- Object-oriented languages often contain a built-in `Thread` class
  - Contains methods to help handle multiple threads

Understanding Threads and Multithreading (continued -2)

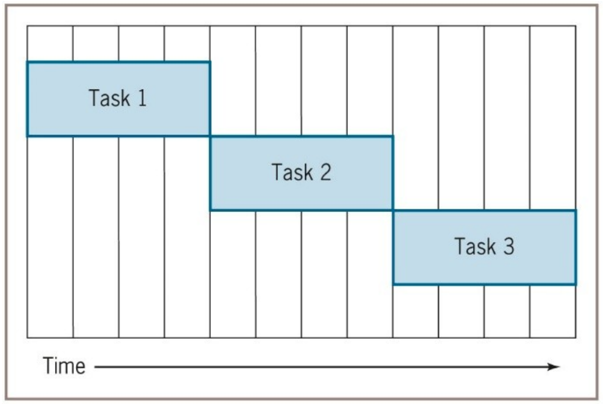


Figure 12-14 Executing multiple tasks as single threads in a single-processor system

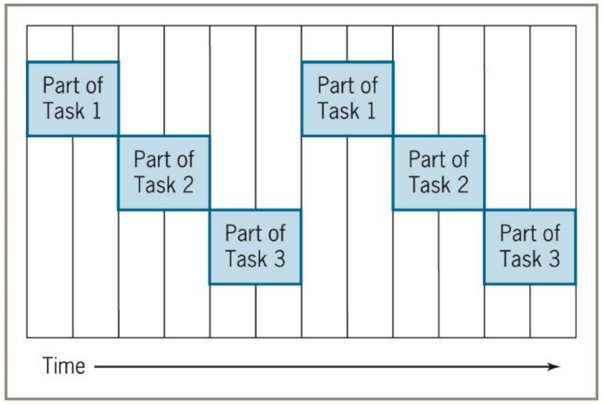


Figure 12-15 Executing multiple threads in a single-processor system

Understanding Threads and Multithreading (continued -3)

- Code carefully to avoid **deadlock** and **starvation**
  - Deadlock occurs when two or more threads wait for each other
  - Starvation occurs when a thread is abandoned because other threads occupy all computer resources
- Techniques like **thread synchronization** avoid problems such as:
  - Two clerks accessing the same record, and the first one updating it while the other is still viewing it
  - The second user still thinks the data is

Creating Animation

- **Animation**
  - A rapid sequence of still images
  - Each is slightly different from the previous one
  - Produces the illusion of movement
- Object-oriented languages offer built-in classes used to draw geometric figures on the screen
- Position
  - Horizontal: **x-axis**, **x-coordinate**
  - Vertical: **y-axis**, **y-coordinate**

Creating Animation (continued -1)

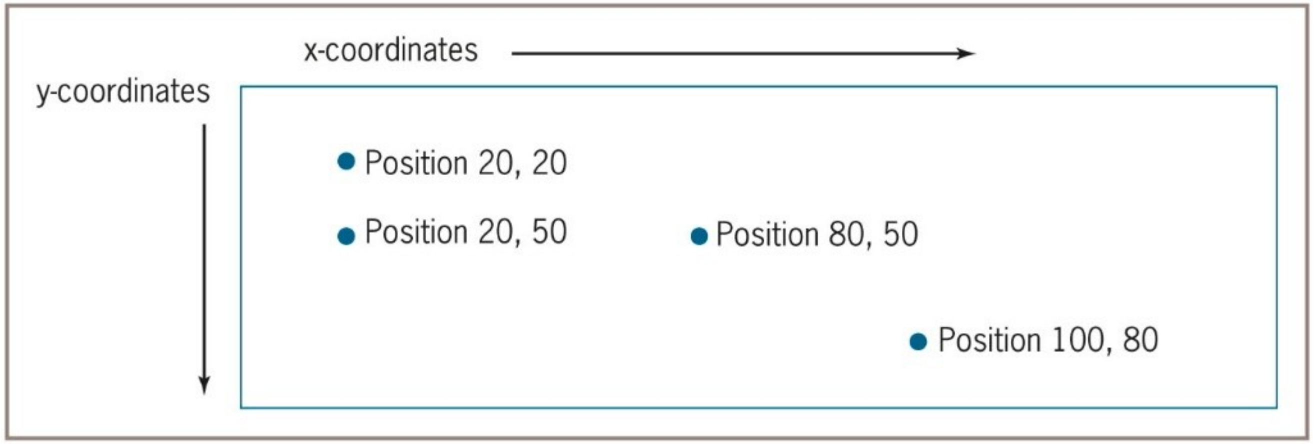
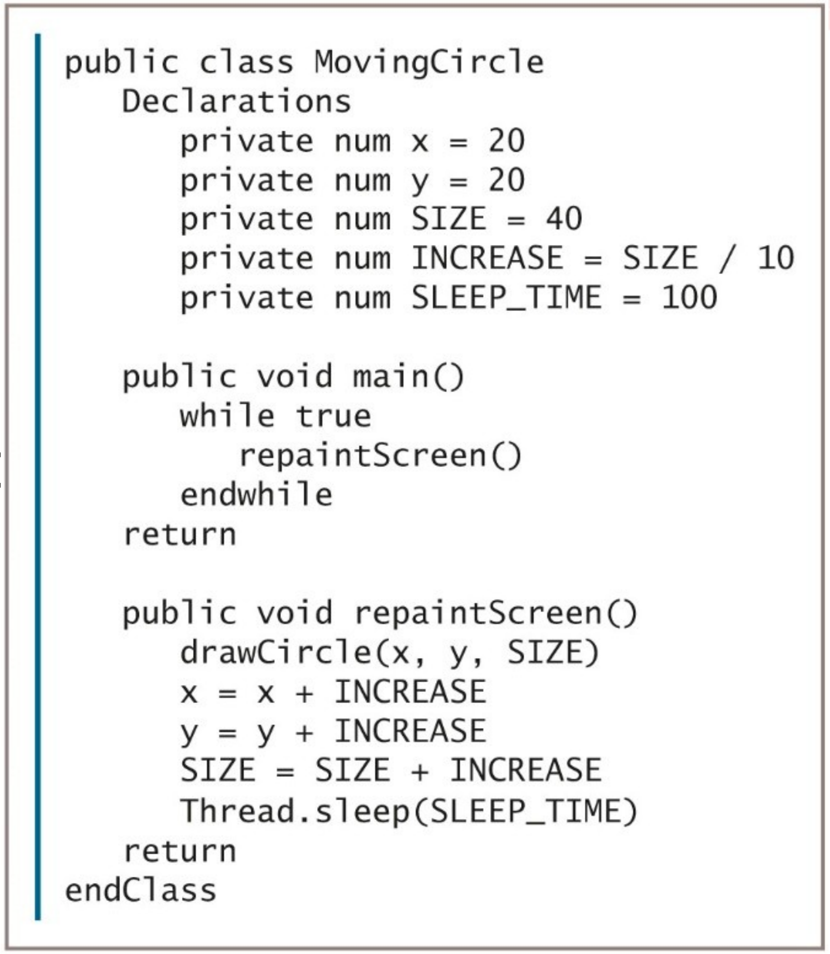


Figure 12-16 Selected screen coordinate positions

Creating Animation (continued -2)

- `MovingCircle` class
  - Moves a circle across the screen
  - Constants `SIZE` an
    `INCREASE`
  - `drawCircle()` method
  - `sleep()` method



```
public class MovingCircle
   Declarations
      private num x = 20
      private num y = 20
      private num SIZE = 40
      private num INCREASE = SIZE / 10
      private num SLEEP_TIME = 100

   public void main()
      while true
         repaintScreen()
      endwhile
   return

   public void repaintScreen()
      drawCircle(x, y, SIZE)
      x = x + INCREASE
      y = y + INCREASE
      SIZE = SIZE + INCREASE
      Thread.sleep(SLEEP_TIME)
   return
endClass
```

Figure 12-17 The `MovingCircle` class

Creating Animation (continued -3)

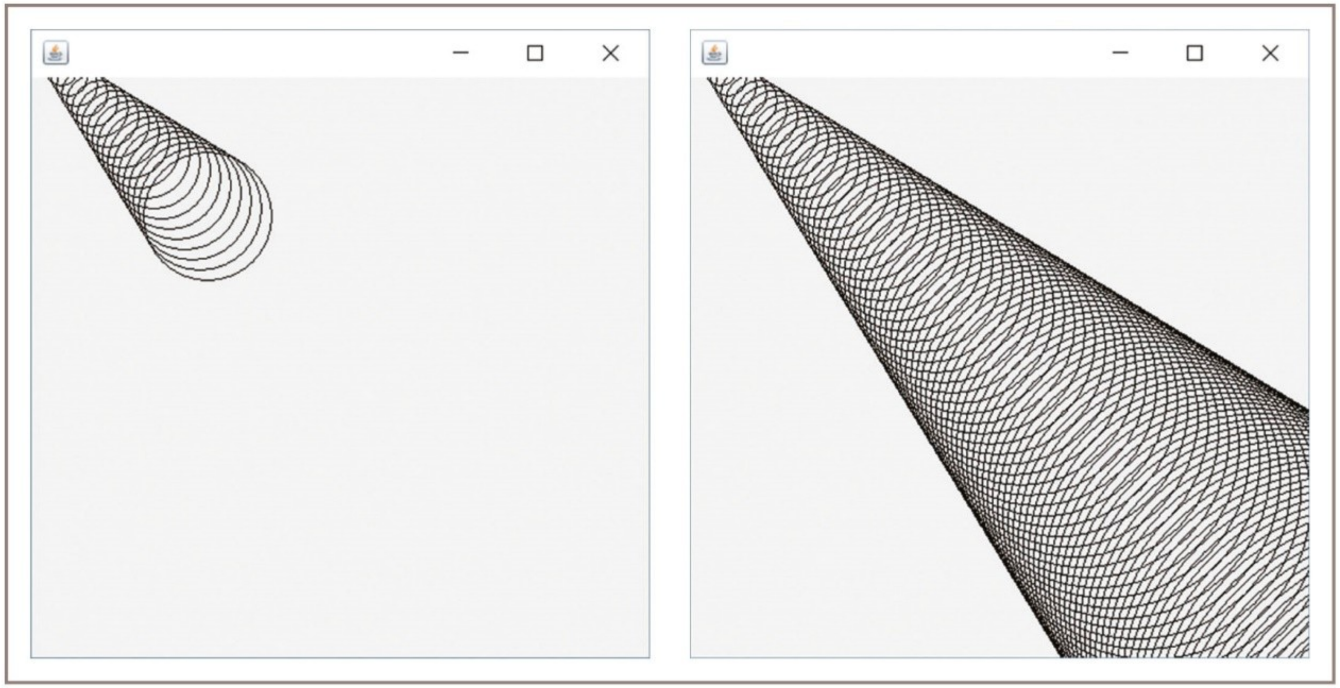


Figure 12-18 Output of the `MovingCircle` application at two points in time

Summary

- Graphical user interface (GUI)
  - Users manipulate objects such as buttons and menus
  - Listeners are "interested in" an event
  - Common user events involve tapping a screen or clicking a mouse
- Common GUI components include labels, text boxes, buttons, check boxes, check box groups, option buttons, and list boxes
- Interface should be natural, predictable, attractive, easy to read, and nondistracting

Summary (continued)

- Event-driven applications require creating wireframes and storyboards, defining objects and dictionaries for them, and defining connections between screens the user will see
- Thread
  - The flow of execution of one set of program statements
- Animation
  - A rapid sequence of still images that produces the illusion of movement