

Skills and Topics for TeenCoder: Windows Programming

Our Self-Study Approach

Our courses are **self-study** and can be completed on the student's own computer, at their own pace. You can steer your student in the right direction with no prior programming knowledge. Students only need typical computer usage skills to start; we will teach them programming from the ground up!

Each course comes with student activity starters, supplementary instructional documents, a **Solution Guide**, fully coded solutions for all activities, tests and answer keys, and guidance on evaluating projects.

Most questions about how to code individual activities are easily answered by referring to the Solution Guide (with or without parental involvement). We also provide **free technical support** to assist with any aspect of the courses!

Teachers who wish to closely monitor and grade student progress for credit purposes can administer chapter tests which are provided (with answer keys). We also provide advice and guidelines for evaluating student activities.

What Skills do Students Need to Begin?

All of our courses assume the student is already familiar with using a keyboard and mouse to select and run software, navigate the menus in a typical software program, and generally interact with their computer.

Students should understand how to use the built-in operating system software (Windows Explorer or Mac Finder) to find, save and retrieve files on their computer. It may also be helpful to have some familiarity with text editors (like Notepad or TextEdit) and some experience using web browsers to find information on the Internet.

We teach students how to program a computer from the ground up, but they should already know the basics about using one!

This course requires a **Windows** computer with CD-ROM or DVD-ROM drive.

Topics Covered In This Course

The following are some of the computer programming topics that are covered in this course. For a full list of topics and sections, please see the Table of Contents for this course.

- Introduction to the C# programming language
- Creating graphical Windows screens
- Using dialog controls
- C# data types and variables
- User input and flow control
- Math functions and string operations
- C# debugging and exception handling
- Object-oriented programming concepts
- Classes, inheritance, and polymorphism
- Collections, sorting, and recursion
- File Input/Output



TeenCoder™ Series

Windows Programming

A hands-on introduction to the field of Windows Programming for high school students.

Student Textbook

Third Edition

Copyright 2013 Homeschool Programming, Inc.

TeenCoder™ Series



TeenCoder™: Windows Programming

Student Textbook

Third Edition

Copyright 2013

Homeschool Programming, Inc.

TeenCoder™: Windows Programming

Third Edition

Copyright © 2013 by Homeschool Programming, Inc.

980 Birmingham Rd, Suite 501-128

Alpharetta, GA 30004

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means without written permission of the author.

ISBN: **978-0-9887033-1-5**

Terms of Use

This course is copyright protected. Copyright 2013 © Homeschool Programming, Inc. Purchase of this course constitutes your agreement to the Terms of Use. You are not allowed to distribute any part of the course materials by any means to anyone else. You are not allowed to make it available for free (or fee) on any other source of distribution media, including the Internet, by means of posting the file, or a link to the file on newsgroups, forums, blogs or any other location. You may reproduce (print or copy) course materials as needed for your personal use only.

Disclaimer

Homeschool Programming, Inc, and their officers and shareholders, assume no liability for damage to personal computers or loss of data residing on personal computers arising due to the use or misuse of this course material. Always follow instructions provided by the manufacturer of 3rd party programs that may be included or referenced by this course.

Contact Us

You may contact Homeschool Programming, Inc. through the information and links provided on our website: <http://www.HomeschoolProgramming.com>. We welcome your comments and questions regarding this course or other related programming courses you would like to study!

Other Courses

Homeschool Programming, Inc. currently has two product lines for students: KidCoder™ and TeenCoder™. Our KidCoder™ Series provides easy, step-by-step programming curriculum for 4th through 12th graders. The Visual Basic series teaches introductory programming concepts in a fun, graphical manner. The Web Design series lets students create their own websites in HTML. Our TeenCoder™ Series provides introductory programming curriculum for high-school students. These courses are college-preparatory material designed for the student who may wish to pursue a career in Computer Science or enhance their transcript with a technical elective. Students can learn C#, Java, game programming, and Android application development.

3rd Party Copyrights

This course uses Microsoft's Visual C# 2010 Express as the programming platform. Visual Studio, Visual Studio Express, Windows, and all related products are copyright Microsoft Corporation. Please see <http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express> for more details.

Instructional Videos

This course may be accompanied by optional Instructional Videos. These Flash-based videos will play directly from a DVD drive on the student's computer. Instructional Videos are supplements to the Student Textbook, covering every chapter and lesson with fun, animated re-enforcement of the main topics.

Instructional Videos are intended for students who enjoy a more audio-visual style of learning. They are not replacements for the Student Textbook, which is still required to complete this course! However by watching the Instructional Videos first, students may begin each textbook chapter and lesson already having some grasp of the material to be read. Where applicable, the videos will also show “screencasts” of a real programmer demonstrating some concept or activity within the software development environment.

This Student Textbook and accompanying material are entirely sufficient to complete the course successfully! Instructional Videos are optional for students who would benefit from the alternate presentation of the material. For more information or to purchase the videos separately, please refer to the product descriptions on our website: <http://www.HomeschoolProgramming.com>.

Table of Contents

Terms of Use.....	3
Disclaimer.....	3
Contact Us.....	3
Other Courses.....	3
3 rd Party Copyrights.....	3
Instructional Videos.....	4
Table of Contents.....	5
Before You Begin.....	11
Minimum Hardware and Software Requirements.....	11
Conventions Used in This Text.....	12
What You Will Learn and Do In This Course.....	13
What You Need to Know Before Starting.....	13
Software Versions.....	13
Getting Help.....	13
Course Errata.....	13
Chapter One: Introduction to Windows Programming.....	15
Lesson One: History of Windows.....	15
Lesson Two: The Evolution of Windows Programming.....	21
Lesson Three: Windows Programming Languages.....	24
Chapter Review.....	26
Activity: Install Visual C# 2010 Express.....	27
Chapter Two: Fundamentals of C#.....	31
Lesson One: Introduction to C#.....	31
Lesson Two: Visual C# 2010 Express Software.....	34
Lesson Three: Hello World.....	43

Chapter Review.....	48
Activity: Enhance Hello World.....	49
Chapter Three: Windows Programming Concepts.....	51
Lesson One: Common Windows Elements.....	51
Lesson Two: C# Syntax.....	54
Lesson Three: Event-Driven Programming.....	57
Lesson Four: Namespaces.....	61
Chapter Review.....	64
Activity: A More Personal Hello.....	65
Chapter Four: Data Types and Variables.....	67
Lesson One: Value Data Types.....	67
Lesson Two: Variables.....	71
Lesson Three: Reference Data Types.....	76
Lesson Four: Introducing Strings.....	78
Chapter Review.....	81
Activity: Experiment with Data Types.....	82
Chapter Five: Basic Flow Control.....	83
Lesson One: Logical Expressions.....	83
Lesson Two: Using the “if” Statement.....	87
Lesson Three: For() Loops.....	90
Lesson Four: While() Loops.....	93
Chapter Review.....	96
Activity: Jeepers, Beepers!.....	97
Chapter Six: User Input.....	99
Lesson One: Text Boxes.....	99

Lesson Two: List Boxes and Combo Boxes	101
Lesson Three: Radio Buttons and Check Boxes	105
Chapter Review.....	107
Activity: Telling Tall Tales	108
Chapter Seven: Math Functions in C#	109
Lesson One: Math Operators (+, -, *, /, %).....	109
Lesson Two: .NET Framework Math Functions.....	112
Lesson Three: A Simple Calculator	114
Chapter Review.....	118
Activity: Divide and Multiply	119
Chapter Eight: Working With Strings	121
Lesson One: Common String Operations.....	121
Lesson Two: Formatting Strings.....	125
Lesson Three: Converting Between Strings and Numbers	128
Chapter Review.....	130
Activity: Caesar’s Cipher	131
Chapter Nine: Methods.....	133
Lesson One: Writing and Calling Methods.....	133
Lesson Two: Method Parameters and Return Values	135
Lesson Three: Calling Methods	137
Chapter Review.....	143
Activity: What’s Your Birthday?	144
Chapter Ten: Debugging and Exceptions	145
Lesson One: The Visual C# 2010 Express Debugger	145
Lesson Two: Debugging Demonstration	148
Lesson Three: C# Runtime Exceptions	152
Chapter Review.....	155
Activity: Divide By Zero	156

Chapter Eleven: Collections	157
Lesson One: Arrays.....	157
Lesson Two: Linked Lists	161
Lesson Three: Enumeration and ForEach.....	165
Chapter Review.....	168
Activity: Your To-Do List	169
Chapter Twelve: Object-Oriented Programming.....	171
Lesson One: Object-Oriented Concepts.....	171
Lesson Two: History of OOP.....	173
Lesson Three: Defining an Object	174
Chapter Review.....	177
Activity: Creating Songs	178
Chapter Thirteen: Classes in C#.....	179
Lesson One: Defining a Class	179
Lesson Two: Properties and Methods	180
Lesson Three: Public vs. Private.....	182
Lesson Four: Constructors	183
Lesson Five: Static Members.....	185
Chapter Review.....	187
Activity: Your Song Player.....	188
Chapter Fourteen: Sorting and Recursion.....	189
Lesson One: Simple Sorting	189
Lesson Two: Recursion	192
Lesson Three: Recursive Sorting	196
Chapter Review.....	201

Activity: The Number Sort.....202

Chapter Fifteen: File Input and Output203

 Lesson One: Using Files in a Program203

 Lesson Two: Reading and Writing Text Files.....206

 Lesson Three: Reading and Writing Binary Files209

 Lesson Four: SaveFileDialog and OpenFileDialog212

 Chapter Review.....216

 Activity: Your Own Notepad.....217

Chapter Sixteen: Inheritance and Polymorphism.....219

 Lesson One: Base Classes and Derived Classes219

 Lesson Two: Using References to Base and Derived Classes.....224

 Lesson Three: Virtual Base Methods226

 Lesson Four: The “Object” Base Class228

 Lesson Five: Using Base Features from Derived Classes230

 Chapter Review.....233

 Activity: Creating the Chess Pieces234

Chapter Seventeen: Final Project.....235

 Lesson One: The Game of Chess.....235

 Lesson Two: The Starter Project238

 Activity One: Starting Your Chess Project243

 Activity Two: Initializing the Game243

 Activity Three: Finishing handleClick()244

 Activity Four: Moving Pawns.....244

 Activity Five: Moving Other Pieces, Testing for Check245

What's Next?.....247

Index249

Before You Begin

Please read the following topics before you begin the course.

Minimum Hardware and Software Requirements

This is a hands-on programming course! You will be installing Microsoft's Visual C# 2010 Express on your computer, which must meet the following minimum requirements:

Computer Hardware

Your computer hardware must meet the following minimum specifications:

	Minimum
CPU	1.6GHz or faster processor
RAM	1024 MB
Display	1024 x 768 or higher resolution
Graphics Card	Supports DirectX-10 or later
Hard Disk Size	3GB available space
DVD Drive	DVD-ROM drive

Operating Systems

Your computer operating system must match one of the following:

Windows XP (x86) with Service Pack 3 or above (except Starter Edition)
Windows Vista (x86 and x64) with Service Pack 2 or above (except Starter Edition)
Windows 7 (x86 and x64)
Windows 8 or Windows 8 Pro (excluding Windows 8 RT)

Conventions Used in This Text

This course will use certain styles (fonts, borders, etc) to highlight text of special interest.

Source code will be in 11-point Consolas font, in a single box like this.

Variable names will be in **12-point Consolas bold** text, similar to the way they will look in your development environment. For example: **myVariable**.

Function names, properties and keywords will be in **bold face** type so they are easily readable.



This picture highlights important concepts within a lesson.



Sidebar may contain additional information, tips, or background material.



Chapter Review sections will highlight key elements from each chapter.



Each chapter includes an activity that allows you to practice the concepts you have learned.

What You Will Learn and Do In This Course

TeenCoder™: Windows Programming will teach you the fundamentals of writing your own computer programs. You will be writing graphical Windows programs using Microsoft's C# programming language. This course is geared for high-school students who have expressed an interest in computer programming or who are looking for college-preparatory material.

Starting with the second chapter, you will complete a hands-on programming project at the end of each chapter. These projects will increase in complexity as you learn more about the C# language!

What You Need to Know Before Starting

You are expected to already know the basics of computer use before beginning this course. You need to know how to use the keyboard and mouse to select and run programs, use application menu systems, and work with the Windows operating system. You should understand how to store and load files on your hard disk, and how to use the Windows Explorer to walk through your file system and directory structures. You should also have some experience with using text editors and using web browsers to find helpful information on the Internet.

Software Versions

You will be using the *Microsoft Visual C# 2010 Express* software to complete this course. This program can be freely downloaded from Microsoft's website. Your course will contain links to download and install instructions in PDF format on our website, <http://www.HomeschoolProgramming.com>. Microsoft may from time to time change their website or download process or release newer versions of the product. Our website will contain updated versions of the instructions as needed.

Getting Help

All courses come with a Solution Guide PDF and fully coded solutions for all activities. Simply install the "Solution Files" from your course setup program and you will be able to refer to the solutions as needed from the "Solution Menu". If you are confused about any activity you can see how we solved the problem!

Course Errata

We welcome your feedback regarding any course details that are unclear or that may need correction. Please contact us using our online "Getting Help" form. You can find a list of course errata for this edition on our website, <http://www.HomeschoolProgramming.com>.

SAMPLE STUDENT LESSON

**The following pages contain a sample student lesson from
the TeenCoder: Windows Programming textbook.**

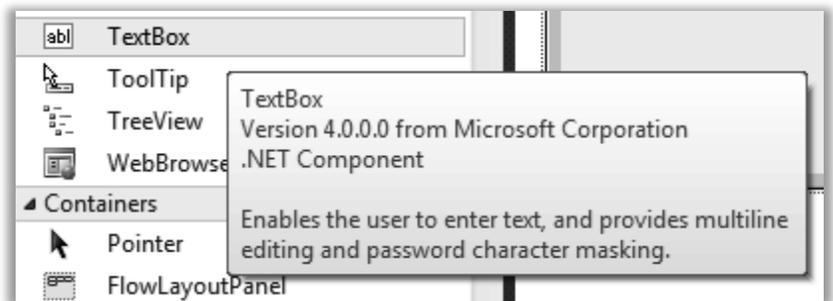


Chapter Six: User Input

In this chapter, we will take a more in-depth look at the different controls that are used to retrieve user input. You will learn about text boxes, list and combo boxes, and radio buttons and check boxes.

Lesson One: Text Boxes

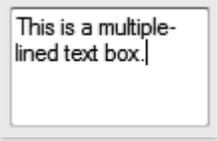
A common user input control is the text box, which allows the user to enter a line of text into a program. The text box control is found in the Toolbox pane of your IDE when you are looking at the Form Design tab. Look under the heading “Common Controls” to find the “TextBox” item.



To add this control to your form, you can double-click it in the Toolbox or click and drag it from the Toolbox to the Form Design screen. You can click on the control’s outline and then drag the mouse to resize or move the text box on the form.

The C# data type representing a text box in code is called **TextBox**. There are many different properties for the **TextBox** control. To view the properties, you can click on the control and then look at the Property Sheet panel. If the panel is not shown on the screen, just right-click the text box within the form and choose “Properties” from the menu. Here is a list of some of the more important properties:

Property Name	Description
(Name)	This is the name of the control. You will use this name to access the control’s information in a program, so make sure this is a descriptive name that follows variable-naming rules.
Text	This is the value displayed in the control. You can use this property at design time to set the default text for the control. At runtime, you can use this property to read in the text that the user has entered, or set this value to your own string.
Enabled	This Boolean value determines whether or not the user can use this control on the form. If it is set to false , the text box will be grayed-out and the user will not be able to use it.
Visible	This Boolean value determines whether or not the user can see this control on the form. If this value is false , the text box will not appear on the form
ForeColor and BackColor	These properties set the foreground (text) and background colors for the control

Multiline	This property determines if more than one line of text is allowed in the control. If false , only one line is permitted. If true , the text box can be resized to hold many lines.	
------------------	--	---

There are many other properties for the **TextBox** control. To see what any of these properties do, just click on the property name and then look at the descriptive text at the bottom of the Properties panel.

If you double-click on the text box control in the form design window, the form code window will appear and a **TextChanged()** event handler function will be automatically created for you.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
}

```

This function will be called any time the user changes the text in the box. This is useful when you need to process the user input character-by-character for some reason. Or, if you need to update another area of the form to reflect the text in a text box control, you will want to know when the user has changed that text.

Using the **TextBox** Variable

Every **TextBox** on the form has a **(Name)** property, and that property determines a variable name for the control within your code. In the example above we used “textBox1” as the **(Name)** property. You can access all of the control’s properties in code by writing “**textBox1.<property name>**”.

The **Text** property is used to get or set the value in a text box. To retrieve the information a user has typed in, you can set a **string** variable equal to the **Text** property:

```
string input = textBox1.Text; // gets current textBox1 contents

```

To set or display information in the text box control, you would just set the **Text** property to a string:

```
textBox1.Text = "Blue"; // set new textBox1 contents

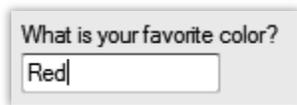
```

These statements are valid from within any of your form methods, including the **TextChanged()** event handler demonstrated above and other event handlers such as button click events.

The text box control will hold only one line of text by default. If you want to change the height of the text box to hold more than one text line, you will need to find the **Multiline** property in the Property panel and

change its value to **true**. Then you can click and drag the control to change its height. If you do not change this **Multiline** property, you will not be able to resize the text box's height.

If you decide to use the text box control to hold multiple lines of text, or you think the user may enter more characters than can be shown on the screen, you may want to add scroll bars to the control. Scroll bars are a set of arrows with a slider in between. A text box can contain horizontal scroll bars, vertical scroll bars or both. Horizontal scroll bars will scroll the text left and right. Vertical scroll bars will scroll up and down. Scroll bars can be enabled or disabled from the Properties sheet.

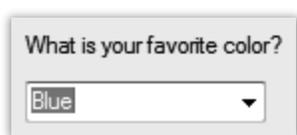
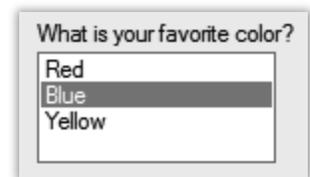


Finally, it is extremely good Windows design to use a descriptive label for any text box controls on your form. Without this, the user will not know what type of information you are expecting! So remember to drag a **Label** control from the Toolbox and position it above or to the side of the text box. Then change the **Text** property of the label with a descriptive string. You do not normally access labels from within the code at runtime. Therefore it is not usually necessary to give the label controls good variable names, because the variables will not be accessed from elsewhere in the program. If you want to change the labels while the program is running for some reason then go ahead and follow standard variable naming procedures.

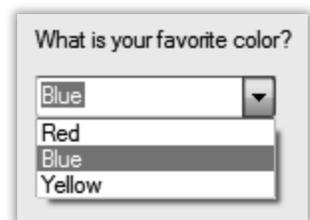
Lesson Two: List Boxes and Combo Boxes

The list box control and the combo box control are useful user input controls. These controls both offer the user a choice of a list of items.

The list box control is used to display multiple items that can be selected by the user. A user can select items in the list by clicking on them. The list box can be handy when you want the user to make one or more selections from a pre-defined list of choices.



The combo box control is very similar to the list box control. Both controls allow a user to select from a list of items. A combo box, however, will only show the currently selected item in the text window. The rest of the list is hidden until the user hits a down arrow on the right. The first image shows the combo box control in its default state, while the second image shows the combo box's full drop-down list. The combo box is very useful in situations where you need to have the user select a single item from a list, but you do not have the space on your form to hold a full list box.

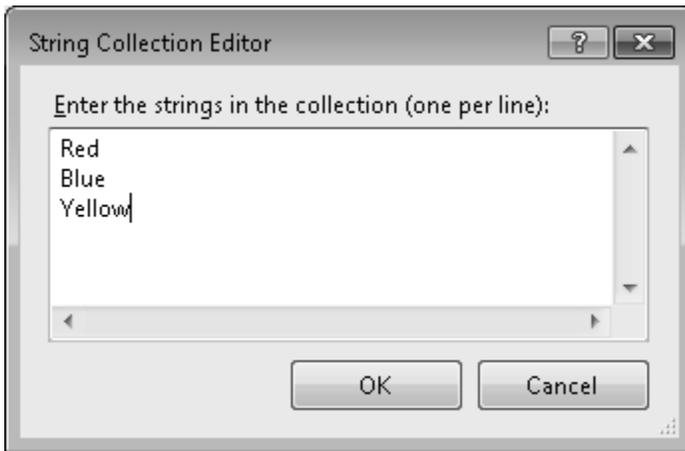
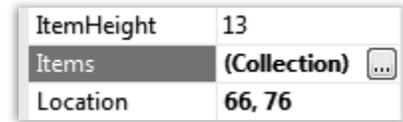


You will find the list box and combo box controls in the Toolbox under "Common Controls" just like the text box. Follow the same drag-n-drop procedure to add either control to your form.

Using the List Box and Combo Box Variables

The C# data types representing list and combo boxes in code are called **ListBox** and **ComboBox**. The list box and combo box controls on the form have a **(Name)** property that works as you might expect. Set the **(Name)** to a meaningful variable name so you can access the control variable in your code.

The list of items in a list box or combo box is set by the **Items** property. You'll note in the Property Sheet that the **Items** property is listed as a "(Collection)" with a little button to the right. To configure your collection of items, click on the "..." button to the right.



Once you click on the Collection button, a "String Collection Editor" window will pop-up on the screen. Here you can enter in as many strings as you need for your list box or combo box. Each string should be separated by hitting the enter key. When your program is run, these strings will show up in the control.

You can also programmatically set or clear the **Items** property. When you are designing a program, you may not know the list of items that you will need to put in your list box. In this case, you may need to wait until the program is running to generate the items in the list. Use the **Clear()** method on the **Items** property to remove all entries from the collection. Then call the **Add()** method to put new items in the collection.

Once you click on the Collection button, a "String Collection Editor" window will pop-up on the screen. Here you can enter in as many strings as you need for your list box or combo box. Each string should be separated by hitting the enter key. When your program is run, these strings will show up in the control.

```
listBox1.Items.Clear();           // remove all items
listBox1.Items.Add("Choice 1");  // add an item
listBox1.Items.Add("Choice 2");  // add another item
```

The items in your collection will by default appear in the order you enter them. If you wish the items to be stored alphabetically, set the **Sorted** property to **true**.

```
listBox1.Sorted = true;
```

Working with Selected Items

There are a few different ways to retrieve items that the user has selected in a list box or combo box. By default, these controls will only allow a user to choose one item at a time. The easiest method to get a single, selected item is to read the control's **Text** property. This will give you the text for the currently selected item

in the list. If no items are currently selected then the **Text** property will contain an empty string (“”). Similarly, you can use the **SelectedItem** property to retrieve the same value, but you have to cast the result to a **string** before assigning to a variable. Both of these statements will give you the same result:

```
string myString1 = myListBox.Text;
string myString2 = (string)myListBox.SelectedItem;
```

You can also set the currently selected item using the same properties:

```
// set the currently selected item
myListBox.Text = "Red";
myListBox.SelectedItem = "Red";
```

The items in a list box or combo box are also represented by an index value. The first item in the collection is index 0; the second is index 1, and so on. In your program, you can get or set the currently selected index by reading or assigning the **SelectedIndex** property. If nothing in the control is selected, the property will be -1.

```
// gets index of currently selected item,
// or -1 if nothing is selected.
int index = myListBox.SelectedIndex;

if (index != -1)        // safety check!
{
    // get the text selected using the index
    string selectedText = (string)listBox1.Items[index];

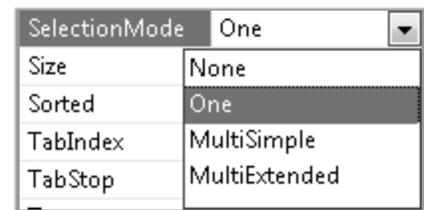
    // show the text we retrieved
    MessageBox.Show(selectedText);
}

// sets the currently selected item index
myListBox.SelectedIndex = 2;
```

You can use the selected index within brackets after the **Items** property to access the item text. We will discuss this “array” notation in more detail in a later chapter, but this demonstration is enough to show you how to pull text from the collection by index. Also note that the **Items** collection actually contains **Objects**, and not **strings**. So you need to cast the resulting **Object** to a **string** before storing it in a **string** variable.

Multiple Selections in a List Box

So far in this lesson we have focused on list boxes and combo boxes where a user can only pick one item at a time. This is the default setting for a list box and your only choice for a combo box! However, it is possible to create a list box that allows multiple items to be selected. To do this, you will need to change the **SelectionMode** property of the list box control from “One” to either “MultiSimple” or “MultiExtended”.



The “MultiSimple” mode will allow any mouse-click or press of the space bar to toggle the selection state of the currently selected item in a list box. This means that if the item “Red” was selected in the example list box on a previous page, you could un-select it by clicking on it with the mouse or by pressing the space bar key. You can also select any other item in the list simultaneously. The “MultiExtended” mode will allow the user to select multiple items by holding down the SHIFT key and then clicking as many items as they want to select with the mouse or arrow keys.

Regardless of the multiple selection mode you choose, a slightly different technique is needed to retrieve all of the selected items. A single **SelectedIndex** or **SelectedItem** property can’t represent everything the user might have selected. For any multiple-selection mode you should use two different properties to access the selected indices or selected items: **SelectedIndices** or **SelectedItems**. Both of these properties are collections just like the **Items** list. There will be one entry in the collection for each item that is currently selected, or the collection will be empty if nothing is selected.

We will discuss collections in more detail later, but the following example code should give you a good idea how to use both **SelectedIndices** and **SelectedItems**. First we examine the length of the collection to make sure at least one item is selected, and then display the first and last selected item in the collection.

```
int numSelected = listBox1.SelectedItems.Count;
if (numSelected > 0) // if at least one is selected
{
    // get the first selected item in the list using SelectedItems
    string firstSelection = (string)listBox1.SelectedItems[0];
    MessageBox.Show(firstSelection);

    // get the last selected index in the list using SelectedIndices
    int index = listBox1.SelectedIndices[numSelected - 1];

    // get the corresponding text from the Items collection
    string lastSelection = (string)listBox1.Items[index];
    MessageBox.Show(lastSelection);
}
```

SAMPLE SOLUTION GUIDE

The following pages contain sample solution material for an activity in the TeenCoder: Windows Programming textbook.

Chapter Six Activity (Telling Tall Tales)

In this activity, the student will create a program that allows users to write a funny story by selecting options from the controls they have learned about in the chapter. The student should begin by creating their own project called “Tall Tales”.

The student will need to add several different types of controls to the form for this activity. The activity description shows a suggested final form layout as well as control types and variable names. The student may use their creativity when adding string contents and options to the controls!

There should be one button on the form called “Create Story”. The student should create the button event handler function by double-clicking on this button. All of the code for this activity will be written within the button’s event handler function.

The key concept is to build up the story in the **FinalStoryText** control by adding new parts to the **FinalStoryText.Text** property based on the **Text** properties of the other controls.

```
private void StoryButton_Click(object sender, EventArgs e)
{
    // Clear the text in the FinalStoryText text box.
    FinalStoryText.Text = "";

    // Write the first line of text to our Final Story.
    string myStoryText = "Once upon a time, there was a ";

    // Append the current value in the SpeciesComboBox to the end of the text
    myStoryText += SpeciesComboBox.Text;

    // Write the next line in our story, appending the name given by the user in
    // the NameTextBox and then adding a period.
    myStoryText += " named " + NameTextBox.Text + ". ";

    // Write the next line in our story, appending the activity chosen by the user
    // in the Activity list.
    myStoryText += "This creature was always " + ActivityList.Text;
```

```
// Check to see if the user has selected the first checkbox.
if (checkBox1.Checked == true)
{
    // If the first checkbox is checked, add 'and ' and the text that was
    // selected.
    myStoryText += " and " + checkBox1.Text;
}

// Check to see if the user has selected the second checkbox.
if (checkBox2.Checked == true)
{
    // If the second checkbox is checked, add 'and ' and the text that was
    // selected.
    myStoryText += " and " + checkBox2.Text;
}

// Check to see if the user has selected the third checkbox.
if (checkBox3.Checked == true)
{
    // If the third checkbox is checked, add 'and ' and the text that was
    // selected.
    myStoryText += " and " + checkBox3.Text;
}

// Write the next line in our story.
myStoryText += ". One day this creature saw a ";

// Check to see which RadioButton was selected by the user.
if (radioButton1.Checked == true)
{
    // If the first RadioButton is selected, add its text to the story.
    myStoryText += radioButton1.Text;
}
else if (radioButton2.Checked == true)
{
    // If the second RadioButton is selected, add its text to the story.
    myStoryText += radioButton2.Text;
}
```

```
else if (radioButton3.Checked == true)
{
    // If the third RadioButton is selected, add its text to the story.
    myStoryText += radioButton3.Text;
}

// Write the final line of our story, appending the user's choice of either
// 'good' or 'bad' from the GoodBadList listbox.
myStoryText += "This was a " + GoodBadList.Text + " day.";

// update our output text box with the final story string
FinalStoryText.Text = myStoryText;
}
```

The completed project for this activity is located in the “Activity Solutions\Tall Tales” folder underneath the Solution Files installation directory.