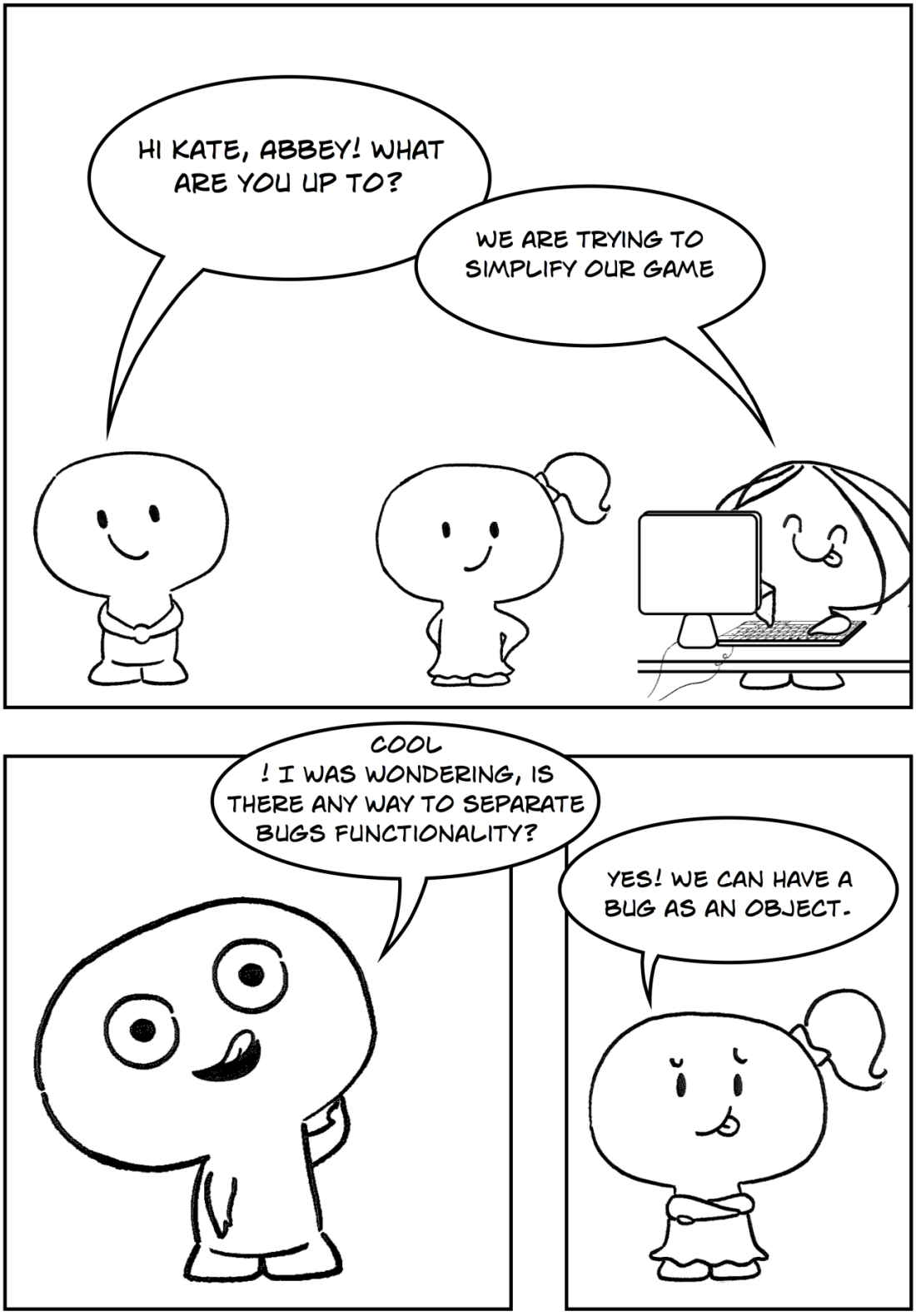
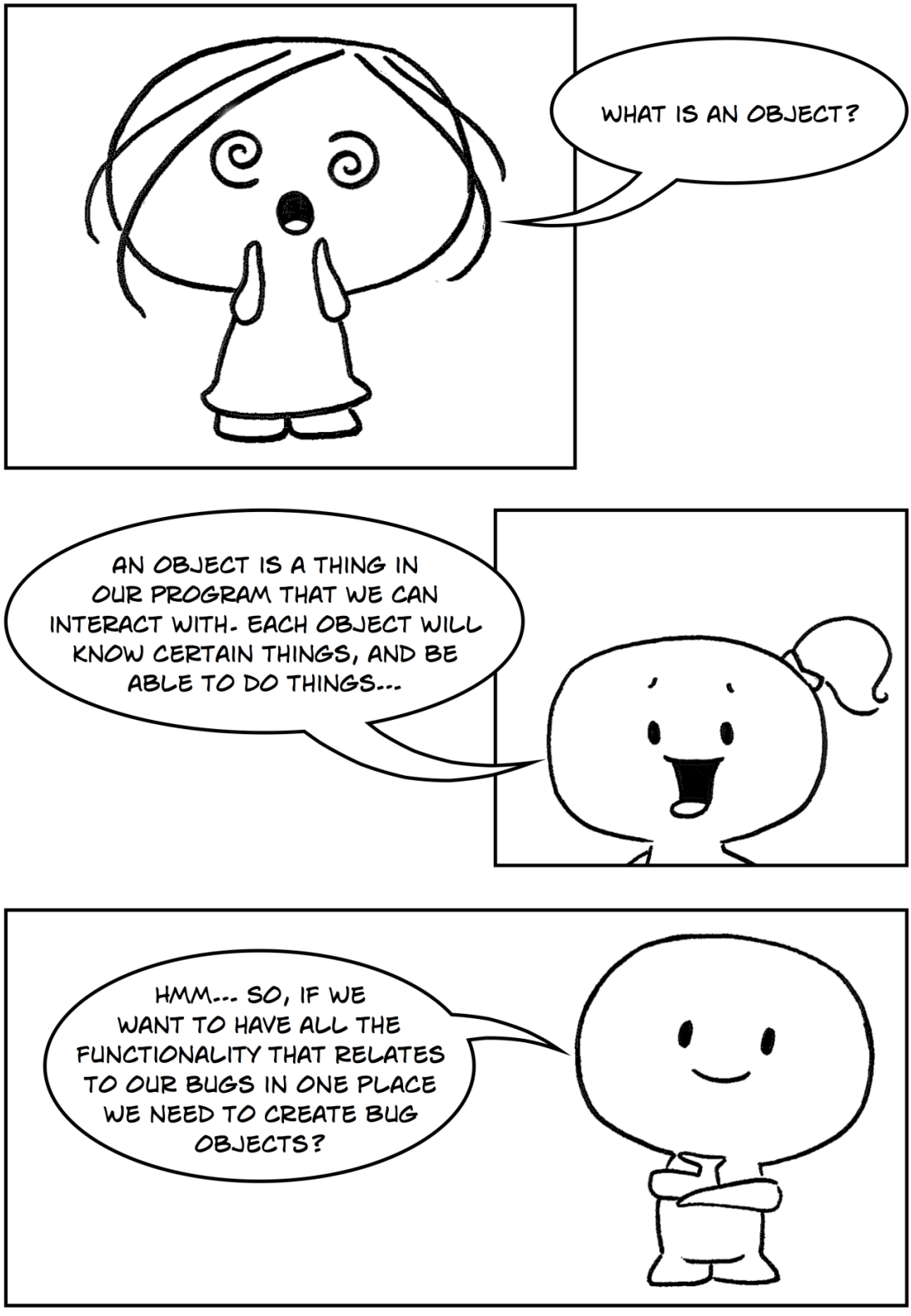
Chapter 7

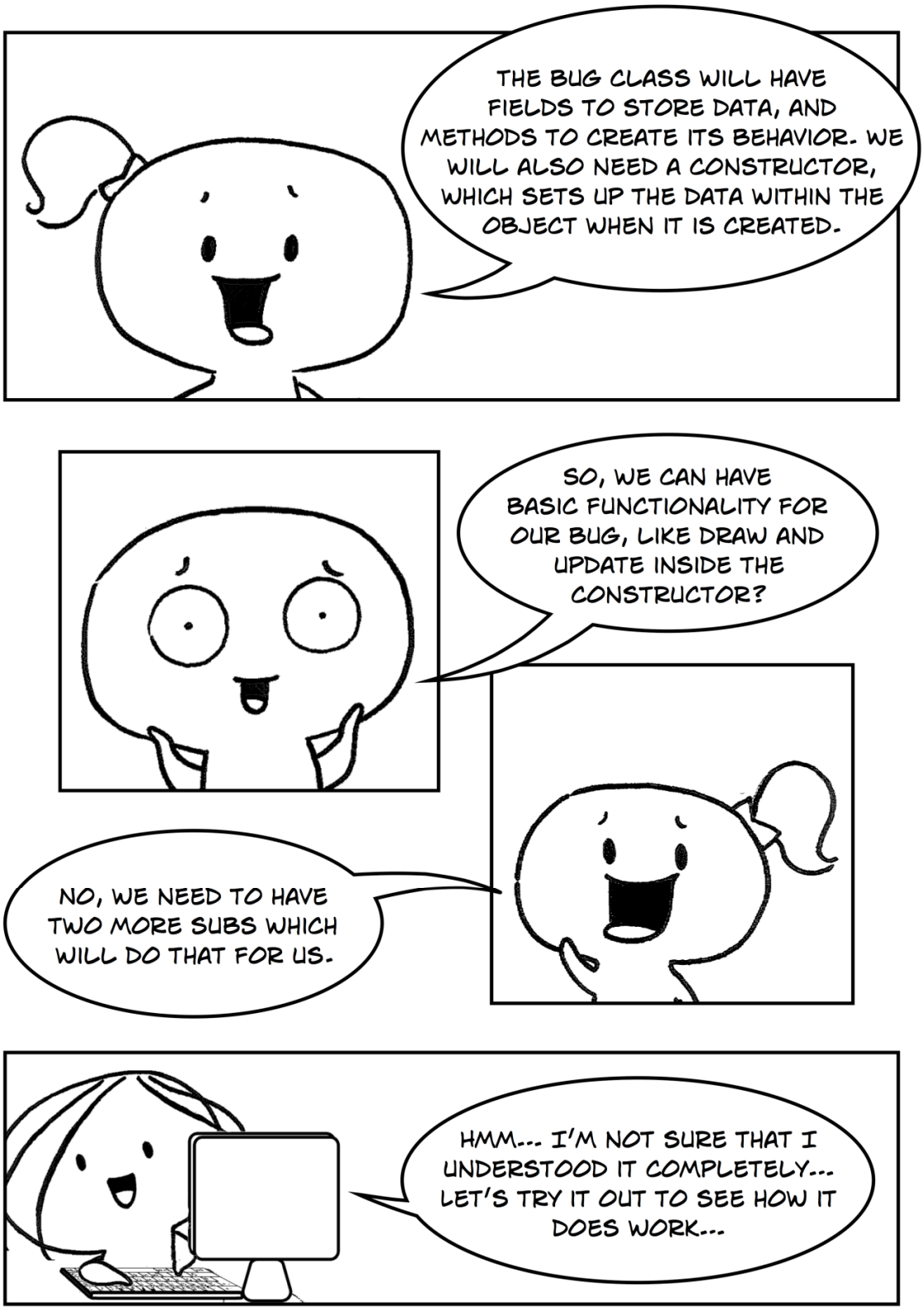
"Objects and classes"

Summary:

To continue with this chapter you will need to use the solution from the previous one. There are no additional materials required. You will learn about objects in VB, how to create objects and object templates.







Part 1

Software objects are modeled after real-world objects in that they have state and behavior as the real ones. A software object maintains its state in variables and implements its behavior with methods. You can represent real-world objects using software objects. However, you can also use software objects to model some abstract concepts.

Object knows and can do certain things. To create a new object we need to create a class, which will be a template for all objects made from this class. This class can contain fields (things that an object knows) and methods (things that an object can do).

In our case we need to create a Bug class which will contain all behavior and all data associated with our object. To do so, we will need a constructor, which will set up the data that we need for our object such as creating a sprite, declare a position of the sprite, declare sprite movement. A constructor is a public sub, which has to have a name "New" and it will be called firstly when an object from this class is created.

Also, in order to create an object and see any result on the screen, we will need to have a Draw() and Update() methods.

Firstly, we need to add a class to your game. To do so, follow the steps below:

1) In solution explorer, choose "Bugs" at the top of the list – this means that you are choosing the whole game (Figure 1):

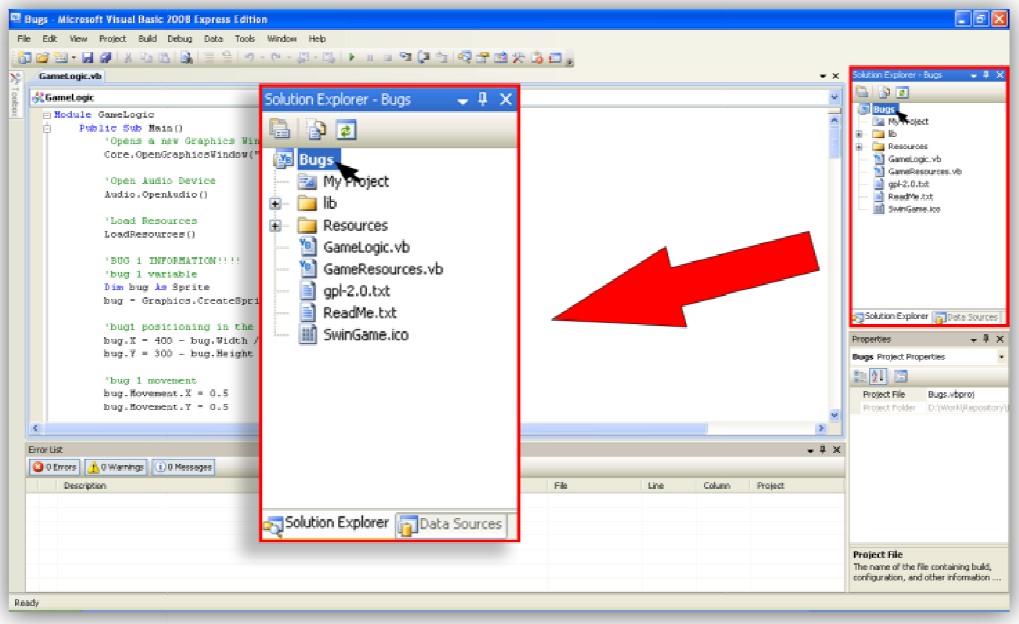


Figure 1

2) Right click inside the solution explorer, choose "Add" => "Class", and type "Bug.vb" as class name and click "Add" button (Figure 2):

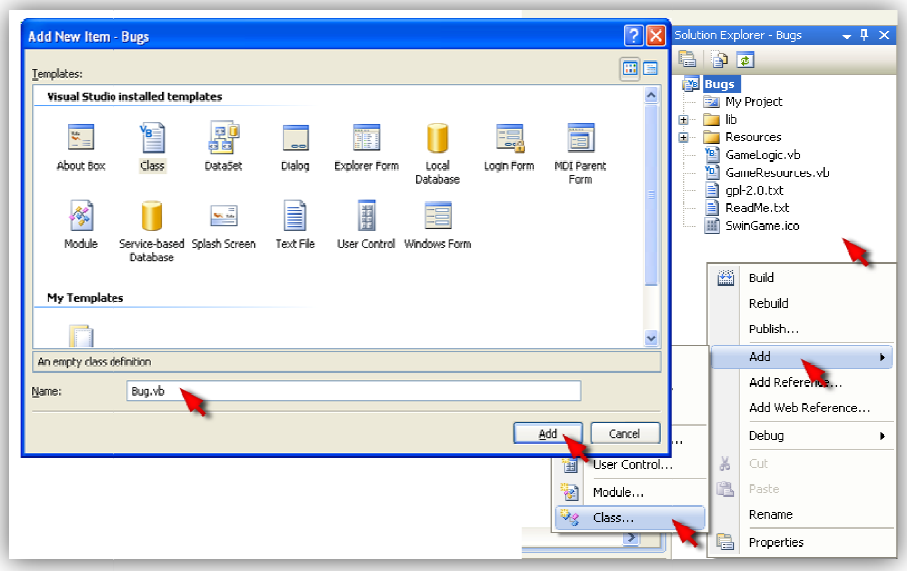


Figure 2

We will need to create three fields inside Bug class. First one will represent an alive sprite – the sprite that you see at the start of the game (`Private AliveSprite As Sprite`), second one will represent dead sprite (`Private DeadSprite As Sprite`) this is the animated sprite which will be played when the sprite was clicked, and the third one will be Alive field which will be Boolean (Boolean is simply true or false value of the variable - `Private Alive As Boolean`). Notice – all of them are private, which means you can access them only inside this class. In order to make them accessible from the outside we need to create a property. A property is a construction in programming language, that allows you to read or write a private field from the outside the class. To create a property for Alive variable you need to use the following construction:

```
Public Property IsAlive() As Boolean
        Get
            Return Alive 'allows to read the value
        End Get
        Set(ByVal value As Boolean)
            Alive = value 'allows to assign a value
        End Set
End Property
```

Exercise 1: Creating fields and a property

Make the following changes in your program and write your solutions to the worksheet:

1. In you Bug class, create AliveSprite, DeadSprite and Alive fields and a property for Alive field.

When we are creating a new object from this class, this object must have Alive = True, because we need to show alive Sprite at the beginning, it must know position of the AliveSprite and the movement. An object must also create a DeadSprite, in order to use it later, and for playing DeadSprite animation only ones, we need to put `DeadSprite.EndingAction = SpriteEndingAction.Stop`. All of this will be enclosed within a constructor.

To create a constructor you need to type `Public Sub New()` into your Bug class and press Enter. Assign Alive variable to true. Type `AliveSprite = Graphics.CreateSprite(GameImage("sprite"))` to create alive sprite. Now we need to declare the position of the sprite. It will be more interesting if we would have each time random starting position for our bug. To do so type

```
AliveSprite.X = Rnd() * (800 - AliveSprite.Width)
AliveSprite.Y = Rnd() * (600 - AliveSprite.Height).
```

Rnd() function generates random number between 0 and 1 (i.e. 0.002). We need to consider width and height of our sprite in order to find right position (so the sprite will be always inside the game window).
Also it will be much more interesting in our bug will move in different directions with different speed. It is easy to have with Rnd() function. All that we need is to put the following code into our constructor:

```
AliveSprite.Movement.X = Rnd() * 2 - 1
AliveSprite.Movement.Y = Rnd() * 2 - 1.
```

Also, do not forget to create the DeadSprite, which will be an animated sprite, in order to call it when AliveSprite have been clicked. To do so, we need to type the following code:

```
DeadSprite = Graphics.CreateSprite(GameImage("deadBug"), 20, 10, 57, 43)
DeadSprite.EndingAction = SpriteEndingAction.Stop.
```

Exercse 2: Creating a constructor


Make the following changes in your program and write your solutions to the worksheet:

1. Ceate a costructor for your Bug class.

Now, in order to see our changes working, we need to have Draw() and Update() methods. The logic for both of these methods is shown in Figure 3 below:

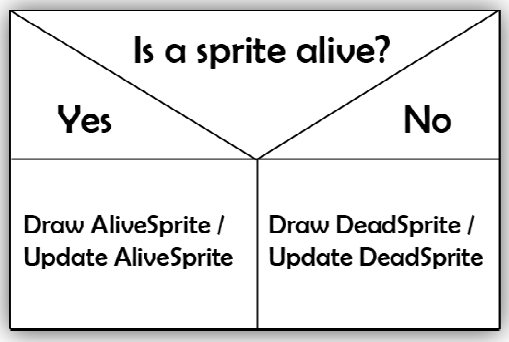


Figure 3

So, for example Draw() method will be as it shown in the area below:

```
Public Sub Draw()
        If IsAlive Then
            Graphics.DrawSprite(AliveSprite)
        Else
            Graphics.DrawSprite(DeadSprite)
        End If
End Sub
```

As for Update() method it will be absolutely the same but instead of using `Graphics.DrawSprite(SpriteName)`, we should use `Graphics.UpdateSprite(SpriteName)`.

Exercise 3: Creating Draw() and Update() methods


Make the following changes in your program and write your solutions to the worksheet:

1. Create Draw() and Update() methods inside the Bug class.

Now we need to make a few changes in GameLogic.vb file in order to see how our program works. Firstly wee need to delete all code that we don't need anymore. Figure 4 shws the code that you have to leave in GameLogic.vb:

```
Module GameLogic

    Public Sub ControlMusic()
       'leave this sub untouched
    End Sub

    Public Sub ChangeVolume()
        'leave this sub untouched
    End Sub

    Public Sub DrawMouse()
        'leave this sub untouched
    End Sub

    Public Sub Main()
        'Opens a new Graphics Window
        Core.OpenGraphicsWindow("Game", 800, 600)

        'Open Audio Device
        Audio.OpenAudio()

        'Load Resources
        LoadResources()

        'Playing music in the loop of infinity
        Audio.PlayMusic(GameMusic("lion"), -1)
```

```
        'Game Loop
        Do
            'Clears the Screen to White (customized color)
            SwinGame.Graphics.ClearScreen(Color.White)

            DrawMouse()
            ControlMusic()
            ChangeVolume()

            'Refreshes the Screen and Processes Input Events
            Core.RefreshScreen()
            Core.ProcessEvents()

        Loop Until SwinGame.Core.WindowCloseRequested() = True

        'Free Resources and Close Audio, to end the program.
        FreeResources()
        Audio.CloseAudio()

    End Sub

End Module
```

Figure 4

Firstly, right after LoadResources() procedure we need to call build-in Randomize() method. This is build-in method which changes the algorithm of choosing the random number in our game. To call it you need to put `Randomize()` right after `LoadResources()`.

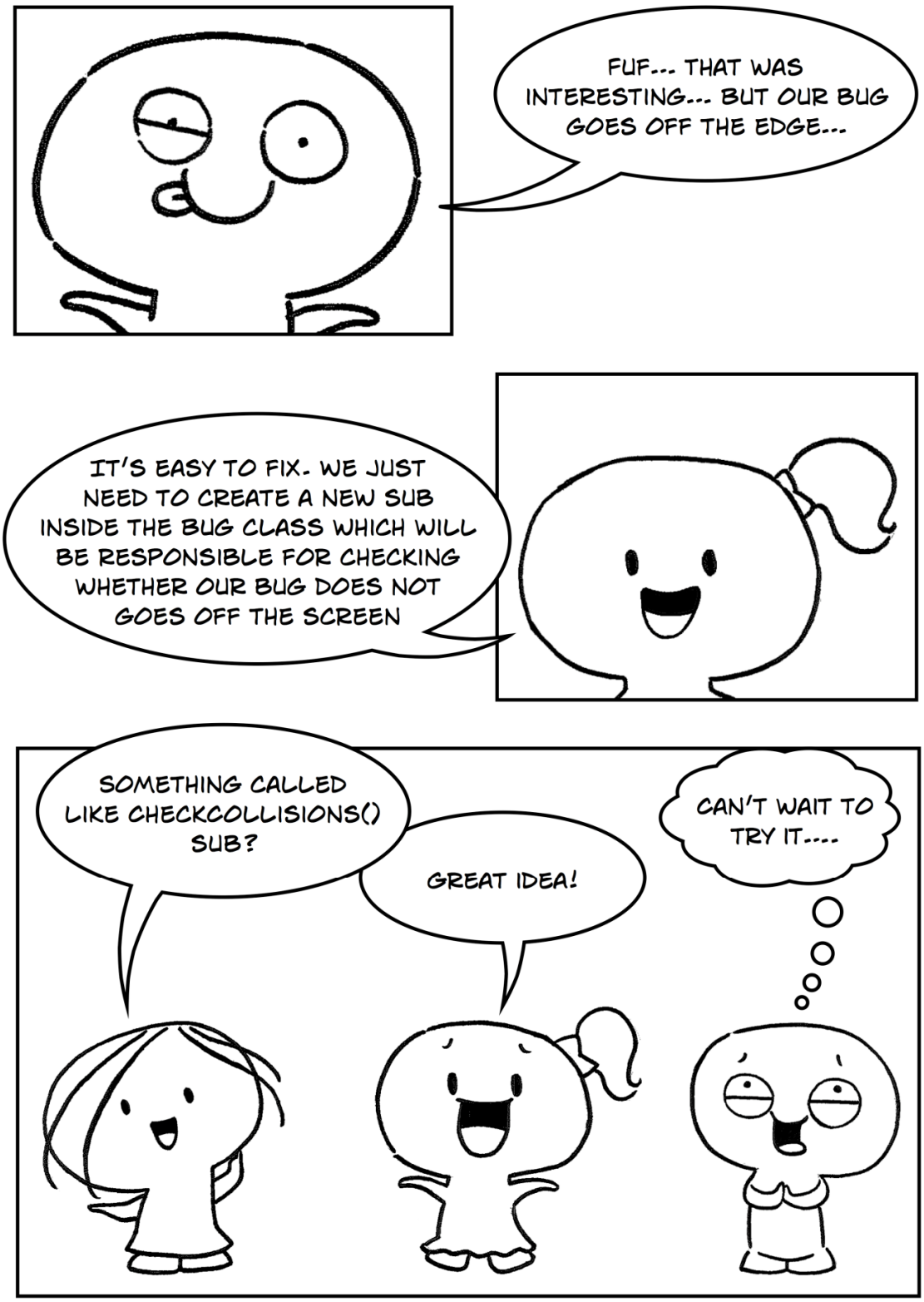
We also need to create a new object caled myBug from the Bug class. We need to have a variable – myBug as Bug (`Dim myBug As Bug`) and we need to associate this variable with the bug class (`myBug = New Bug`). We are creating an object only once, so it have to be made outside the Game Loop. But we still need to tell our object to draw and to update itself. It must be made insude the Game loop and the code for this is shown below:

```
    SwinGame.Graphics.ClearScreen(Color.White)

    myBug.Draw()
    myBug.Update()
```

Exercise 4: Creating an object

Make the following changes in your program and write your solutions to the worksheet:

1. Create myBug object and make it to draw and to update itself (do not forget to add Randomize() method). Debug to see the result.



Part 2

Now we need to have a sub inside the Bug class that will check whether a bug goes off the screen and play the sound when the bug hits the edge. To create this method we need to type the following inside the Bug class:

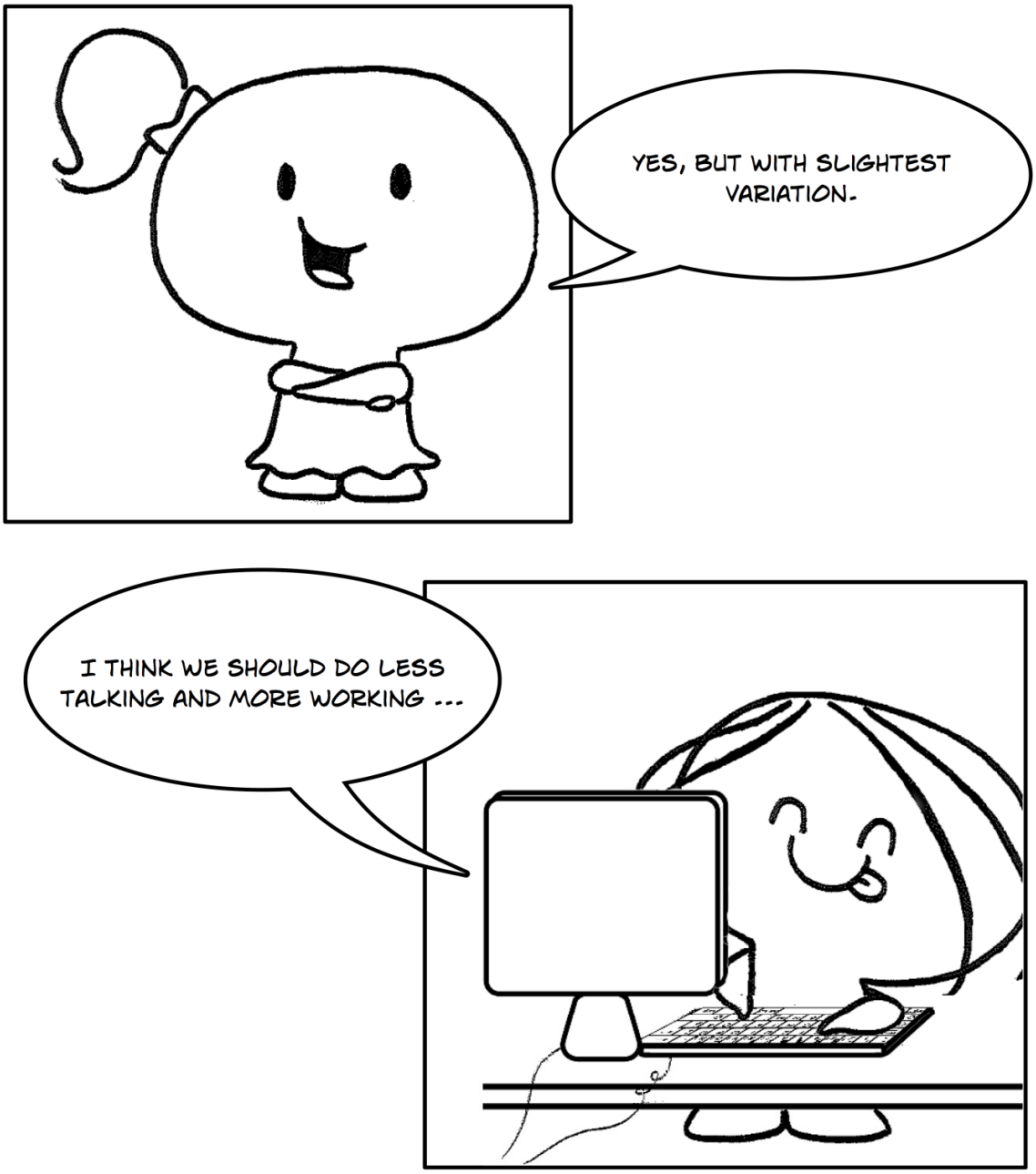
```
Private Sub CheckCollisions()
        If AliveSprite.X + AliveSprite.Width >= Core.ScreenWidth Or
AliveSprite.X <= 0 Then
            AliveSprite.Movement.X = -AliveSprite.Movement.X
            Audio.PlaySoundEffect(GameSound("hit"))
        End If

        If AliveSprite.Y + AliveSprite.Height >= Core.ScreenHeight
Or AliveSprite.Y <= 0 Then
            AliveSprite.Movement.Y = -AliveSprite.Movement.Y
            Audio.PlaySoundEffect(GameSound("hit"))
        End If
End Sub
```

This sub is private because there is no need to call it from the outside. All that we need now is to put `CheckCollisions()` inside after `Graphics.UpdateSprite(AliveSprite)` inside the `Update()` method.

Exercise 1: Creating CheckCollisions() method

Make the following changes in your program and write your solutions to the worksheet:

1. Create CheckCollisions() method inside the Bug Class. Debug to see the result.



Part 3

At this point our bug seems immortal – you cannot click on it to kill it and play animation. So we need to create a new sub inside the Bug class that will check whether the bug was clicked. To do so, put the following code after CheckCollisions() sub:

```
Private Sub CheckIfClicked()
   Dim mousePoint As Point2D
   mousePoint = Input.GetMousePosition()

   If IsAlive And Physics.IsSpriteOnScreenAt(AliveSprite,mousePoint.X,
mousePoint.Y) Then
        If Input.MouseWasClicked(MouseButton.LeftButton) Then
           Audio.PlaySoundEffect(GameSound("hit1"))
           Alive = False

           DeadSprite =  Graphics.CreateSprite(GameImage("deadBug"),
20, 10, 57, 43)

           DeadSprite.X = AliveSprite.X
           DeadSprite.Y = AliveSprite.Y
        End If
   End If
End Sub
```

This sub is checking whether the sprite is at mouse position and whether it was clicked. If so, it will play sound effect, changing Alive to false, do the animated DeadSprite will be played and updated, and it puts the DeadSpirte at position where the AliveSprite was clicked.

We need to call this method right under the CheckCollisions() call inside the Update() method.

Exercise 1: Creating CheckIfClicked() method


Make the following changes in your program and write your solutions to the worksheet:

1. Create ChechkIfClicked() method inside the Bug class. Debug to see the result.