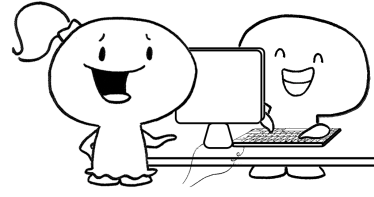
## VBugs

## Chapter 7

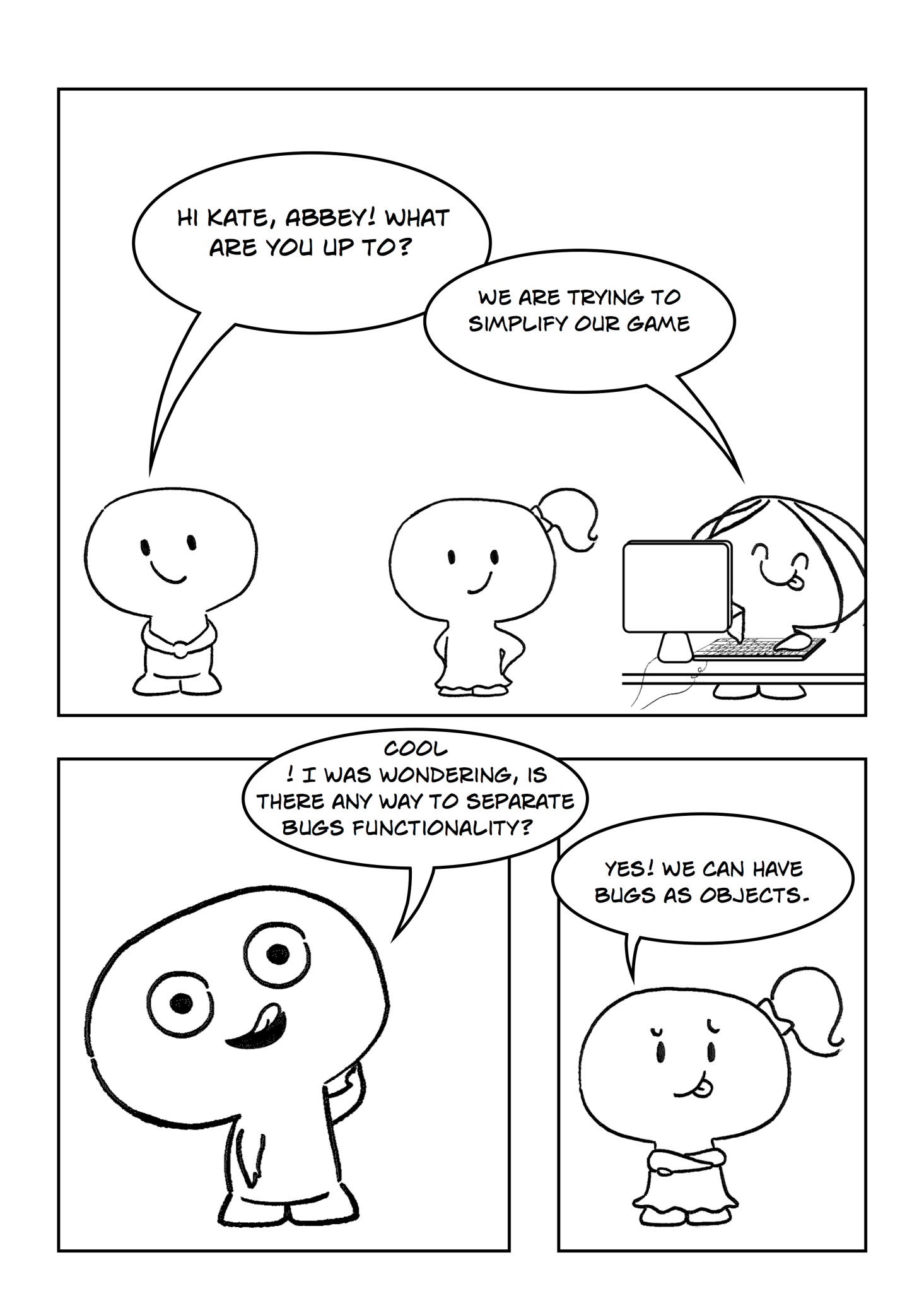
###### Objects and Classes

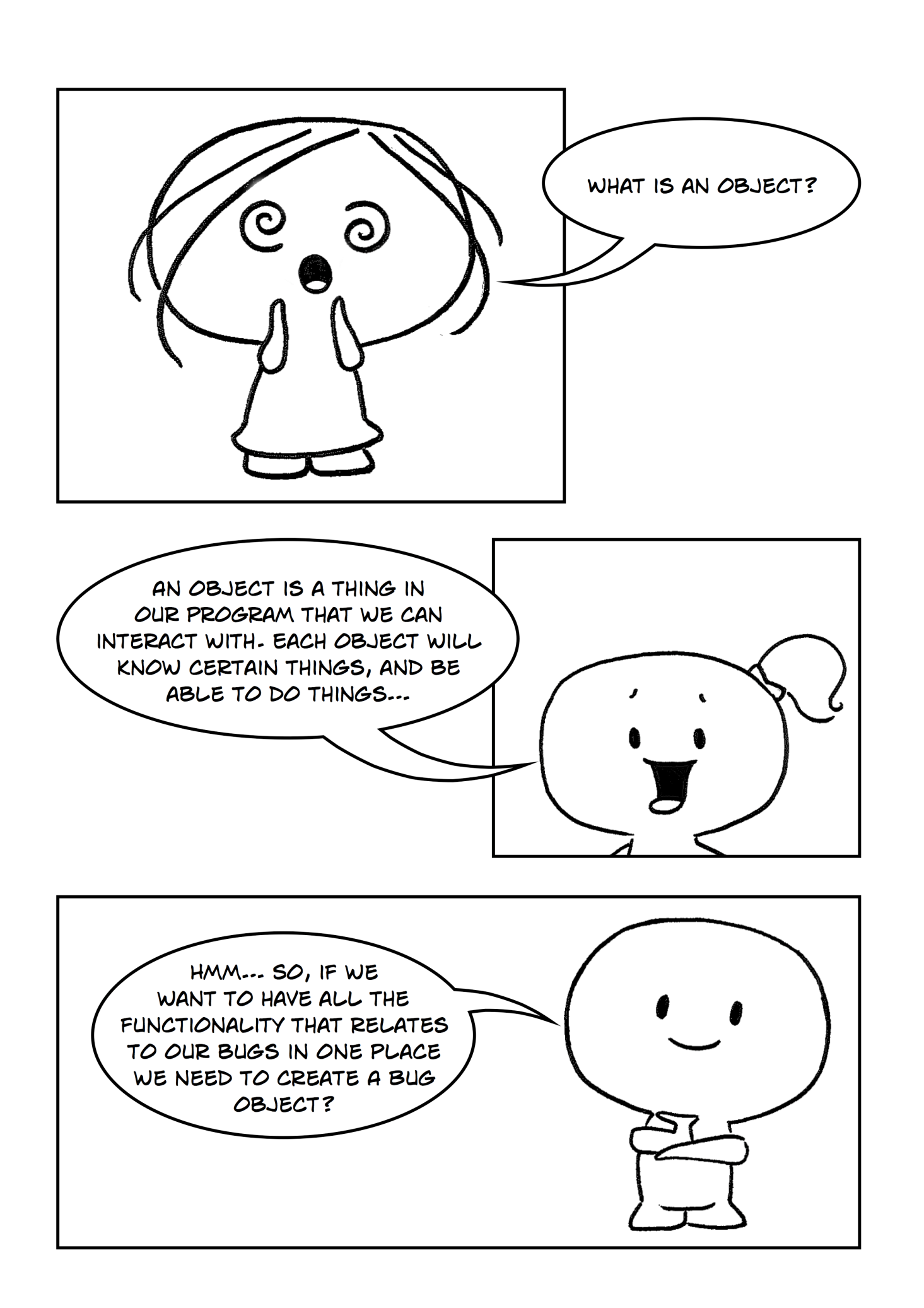


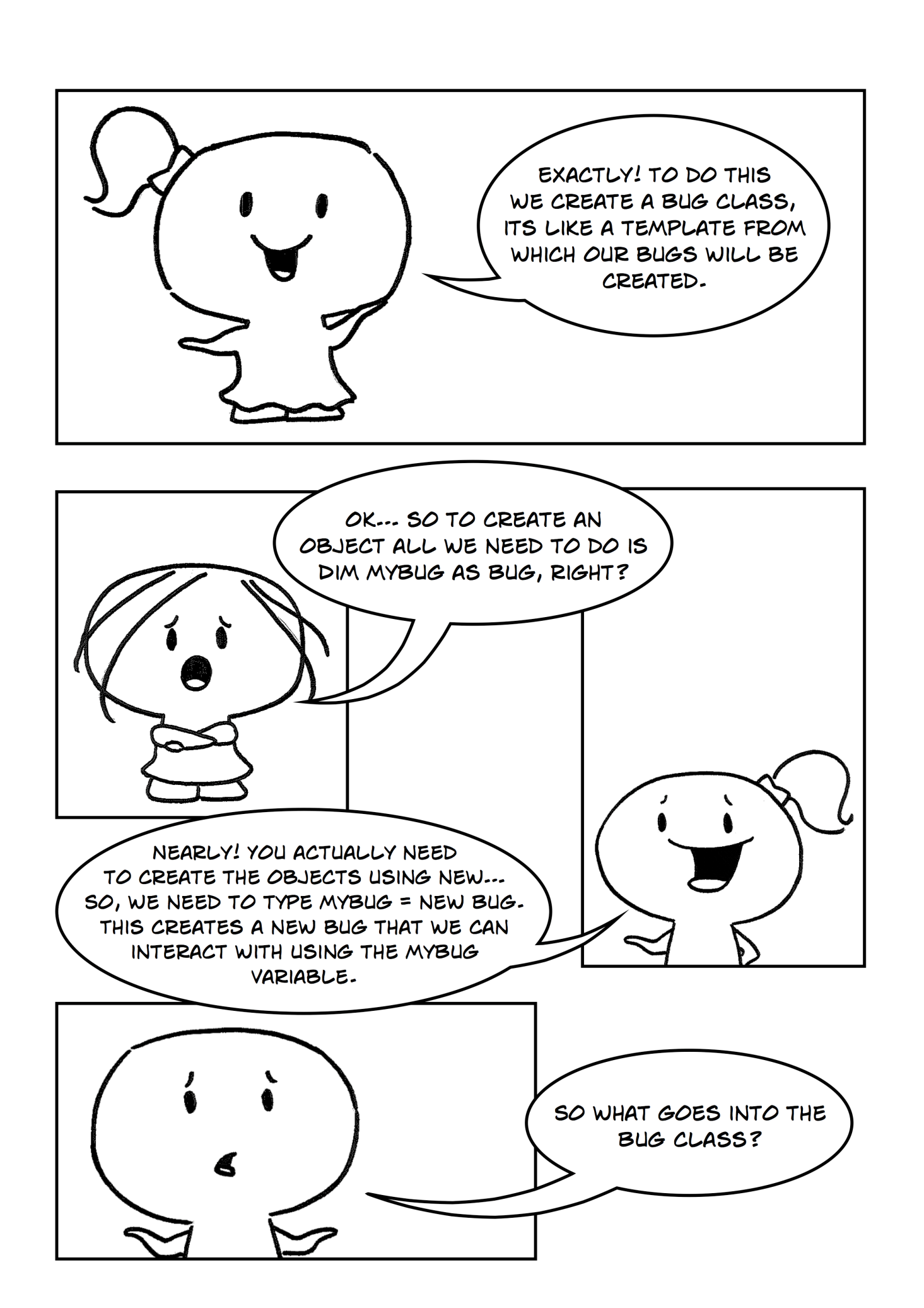


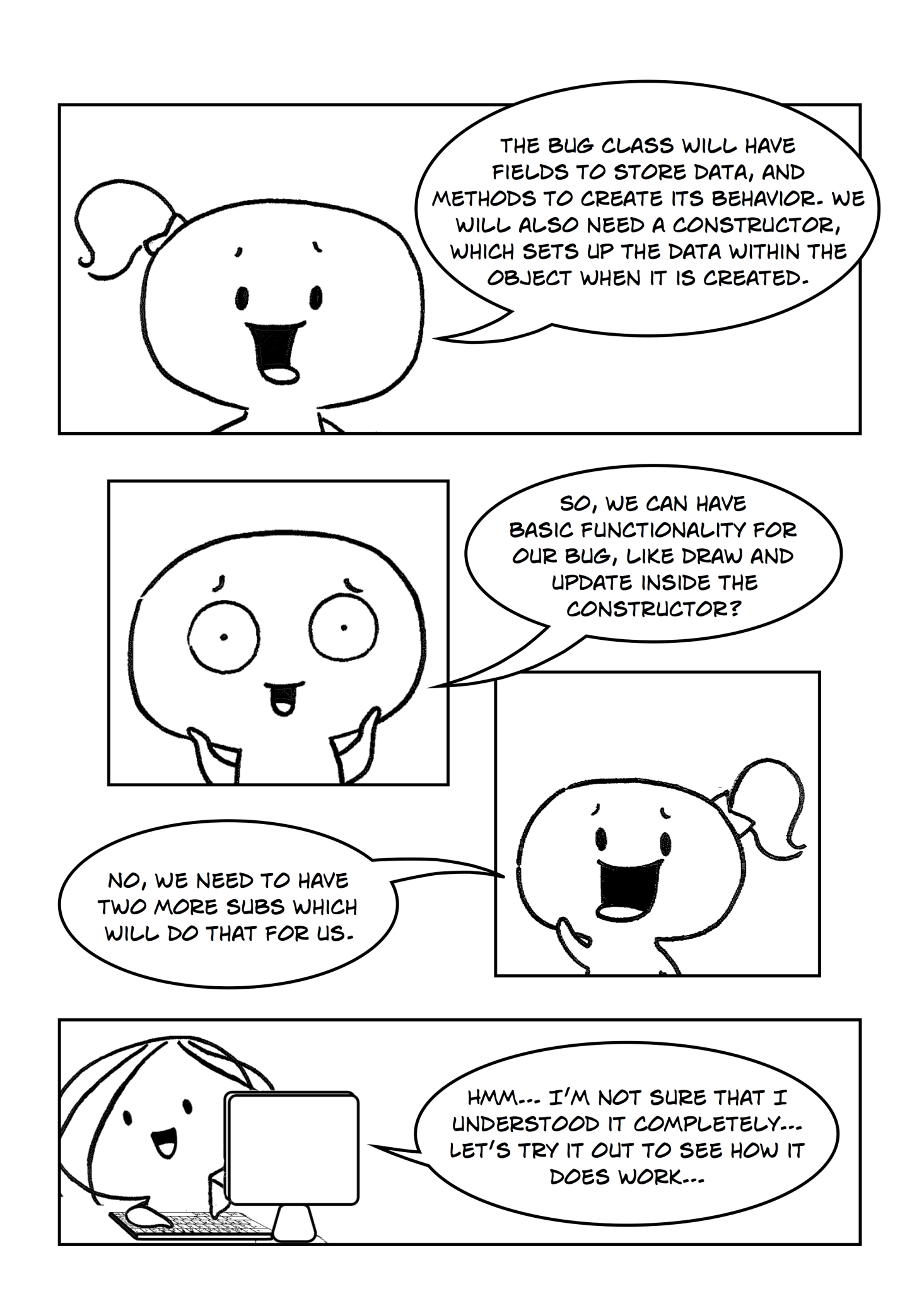
# Summary:

In this chapter we will build on the solution from the previous one. There are no additional materials required. You will learn about objects in VB and how to create objects and object templates. This will allow us to start turning our project in to a complex and enjoyable game.

****

****

****

****

## Objects, Classes and Constructors

Software objects are modeled after real-world objects in that they have “state” and a “behavior”. For example a washing machine has a state of “off” and “on” and has a behavior of washing clothes. A software object stores 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 more abstract concepts not related to the real world.

An **object** knows certain things and can do certain things. We want to create an object that knows everything about our bugs that can also create them for us on the screen when we call it.

In creating our new a new object we need to first create a **class**. A class is a template that defines the properties of an object and the methods used to control that object's behavior. So our class will contain **fields** (things that an object knows) and **methods** (things that an object can do). In our case we want to create a “Bug” class which will contain all behavior and all the data associated with our Bug object.

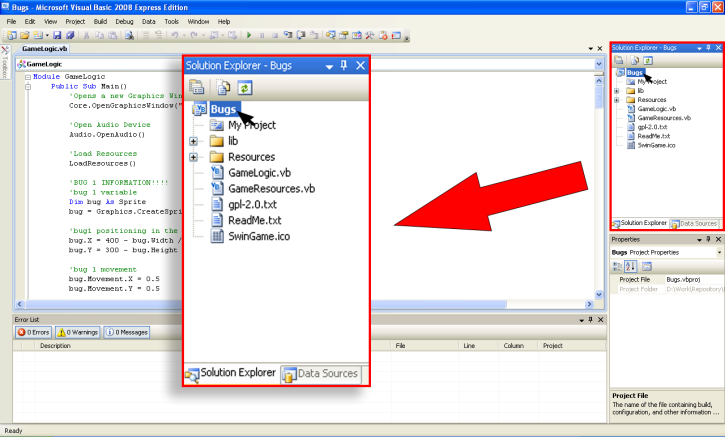
Before creating our class we will need a **constructor** whose job is to intialise the objects in the class. The constructor will initialise our sprites, declare their position on the screen and their movement. A constructor is a “Public” sub, which has to have the name “New”, it will be the first thing by defaul that is called when an object from this class is created.

To see our on the screen, we will need to have a Draw() and an Update() method as well.

*cha 2 - worksheet.pngQuestion1: Define the terms: object; class; constructor.*

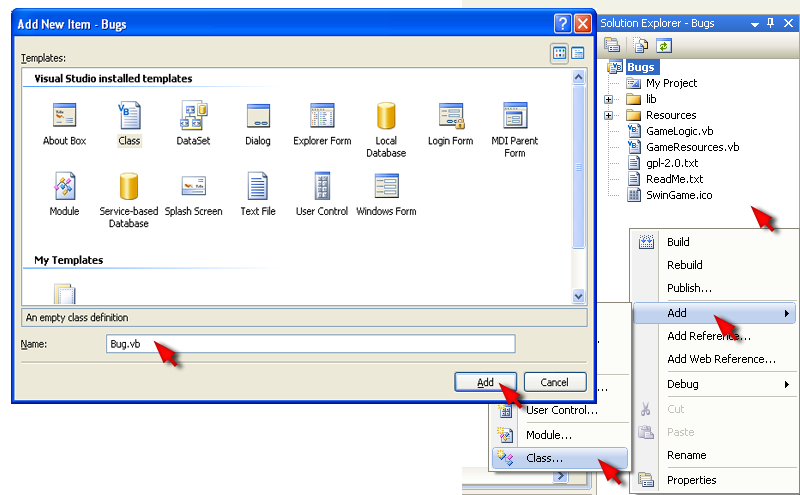
First add a class to your game follow the steps below:

1. In solution explorer, right click “Bugs” at the top of the list – this means that you are choosing the whole game (Figure 1):



Figure

1. Choose “Add” => “Class”, and type “Bug.vb” as the class name and click “Add” button (Figure 2):



Figure

Inside your bugs class we will need to create three fields to control the Sprites. The 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 an “Alive” field which will be Boolean (Remember “Boolean” is simply a variable type that stores either a true or false value - Private Alive As Boolean). This Boolean field will be “True” if the bug is alive and be changed to “False” if the Bug is dead.

## Properties

Notice that all three of these fields are set as “Private”, which means you can access them only inside this class (Bugs.vb). In order to make them accessible from the outside we need to create a property. A **property** is used in programming language to allow you to read or write to a “Private” field from somewhere outside the class. The following is an example of a property construction for IsAlive(), it allows us to be able to access whether our bug is alive or dead from outside the class:

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

*cha 2 - worksheet.pngQuestion2: Define the term: property.*

*Exercise 1: Creating fields and a property*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

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

## Creating the Constructor

The constructor when creating a new object from the Bugs class needs to perform the following actions:

1. Set the variable “Alive” to “True”, because of course we need to show a living Sprite when it is first created.
2. Create the “Alive” sprite in the code.
3. Set the position of the AliveSprite on the screen.
4. Set the movement and direction of the Alive sprite.
5. Create a DeadSprite, for later use when the living Sprite is clicked (killed).
6. So that the DeadSprite animation is only played once like before, we need to include DeadSprite.EndingAction = SpriteEndingAction.Stop. in the constructor as well.

To create a constructor you need to add this to your Bug class(remember constructors have to be called “New”):

|  |
| --- |
| Public Sub New()  …  End Sub |

1. Then in the constructor set the Alive variable to “True”
2. Now create the “Alive” sprite in the code using:

AliveSprite = Graphics.CreateSprite(GameImage("sprite"))

1. Now we need to declare the position of the sprite. It will be more interesting in the game if the bug is given a random starting position. To do this we will use the Rnd() function which generates random number between 0 and 1 (i.e. 0.002). So if we multiply this random number by the width of screen(800) and assign that value to the X position of the new sprite it should start the Sprite in random X position every time somewhere on the screen.

So for example a random number of 0.5 is generated by Rnd() and we multiply that by the screen width of 800 we get 400. Therefore the sprite will start at an X position of 400 (the middle of the SwinGame screen). If however it generates a random number of 1.0 we have a problem because 800 × 1.0 = 800. And if we set the X position of the bug to 800 then it will be off the screen (remembering the position is taken from the top left corner). So to get around this all we need to do is minus the width of the bug as is shown in the code below:

AliveSprite.X = Rnd() \* (800 - AliveSprite.Width)

AliveSprite.Y = Rnd() \* (600 - AliveSprite.Height)

1. The game would also be much more interesting each new bug moved in random directions at a random speed. We want the speed to vary between -1 and 1 and that is what the following code will do:

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

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

1. You will remember that the DeadSprite, which is an animated sprite, has to called when the AliveSprite has been clicked. At this point we just need to add the code to enable the DeadSprite so we can call it if we need to when it has been clicked, enable the DeadSprite with the following code:

DeadSprite = Graphics.CreateSprite(GameImage("deadBug"), 40, 10, 57, 43)

1. We need to tell the DeadSprite animation to stop after it plays once just like we did before:

DeadSprite.EndingAction = SpriteEndingAction.Stop.

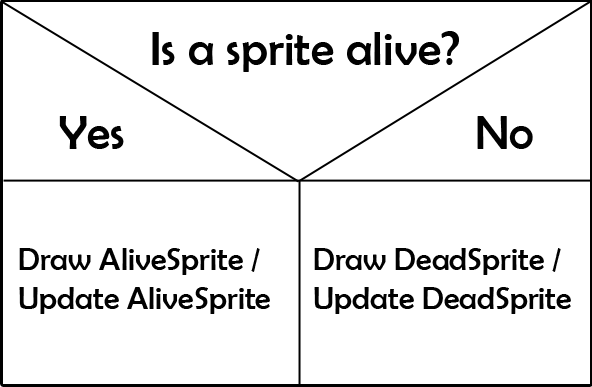
*Exercise 2: Creating the constructor*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

Create a costructor for your Bug class.

## Drawing and Updating our Sprite

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



Figure

Draw() will be a Public sub in the Bug class and you will need to use an If-Then-Else statement if the sprite is alive (If IsAlive Then...). If it is alive then draw the living sprite (Graphics.DrawSprite(AliveSprite)) otherwise draw the dead sprite (Graphics.DrawSprite(DeadSprite)) .

As for Update() method it will be the same as Draw() but instead of using: Graphics.DrawSprite(SpriteName) we should use Graphics.UpdateSprite(SpriteName).

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

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

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

## Modify you Code

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 below shows the all the code need GameLogic.vb, delete everyting except the sections indicated below:

|  |
| --- |
| 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)  'Hide the mouse pointer  Input.ShowMouse(False)    ‘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()  Music.Stop()  End Sub  End Module |

Figure 4

## Randomize

At some point we need to call the Randomize() method. This is a built-in method which comes up with a random numbers based on the clock in your computer and a mathamatical equation . These random numbers will be used when we randomise the location, speed and direction of our bugs. To call this method just put Randomize() right after LoadResources()in GameLogic.vb.

***Did you know?***

**** Computers cannot really generate “truly” random numbers. To be truly random a number must not be predictable at all. Computers will use a complex mathematical algorithm (equation) to make a list of numbers that look random, but if you ask it to do it many times it will come up with the same list of numbers as it has used the same algorithm and same starting point. Using the Randomise() feature means the number that goes into this algorithm (the starting point) is based on the computer clock, so if we ran it twice the time would be different so the list of random numbers based on this number would be different and seemingly more random….although not “truly” random.

Now to add a bug to our game based on our new Bug class. To do so we need to make a new object called myBug (Dim myBug As Bug). Then we need to associate this variable with the bug class (myBug = New Bug). We are creating this object only once, so it has to be made outside the Game Loop. But we still need to tell our object to draw and to update itself over and over again so this must be put into Game loop as shown below:

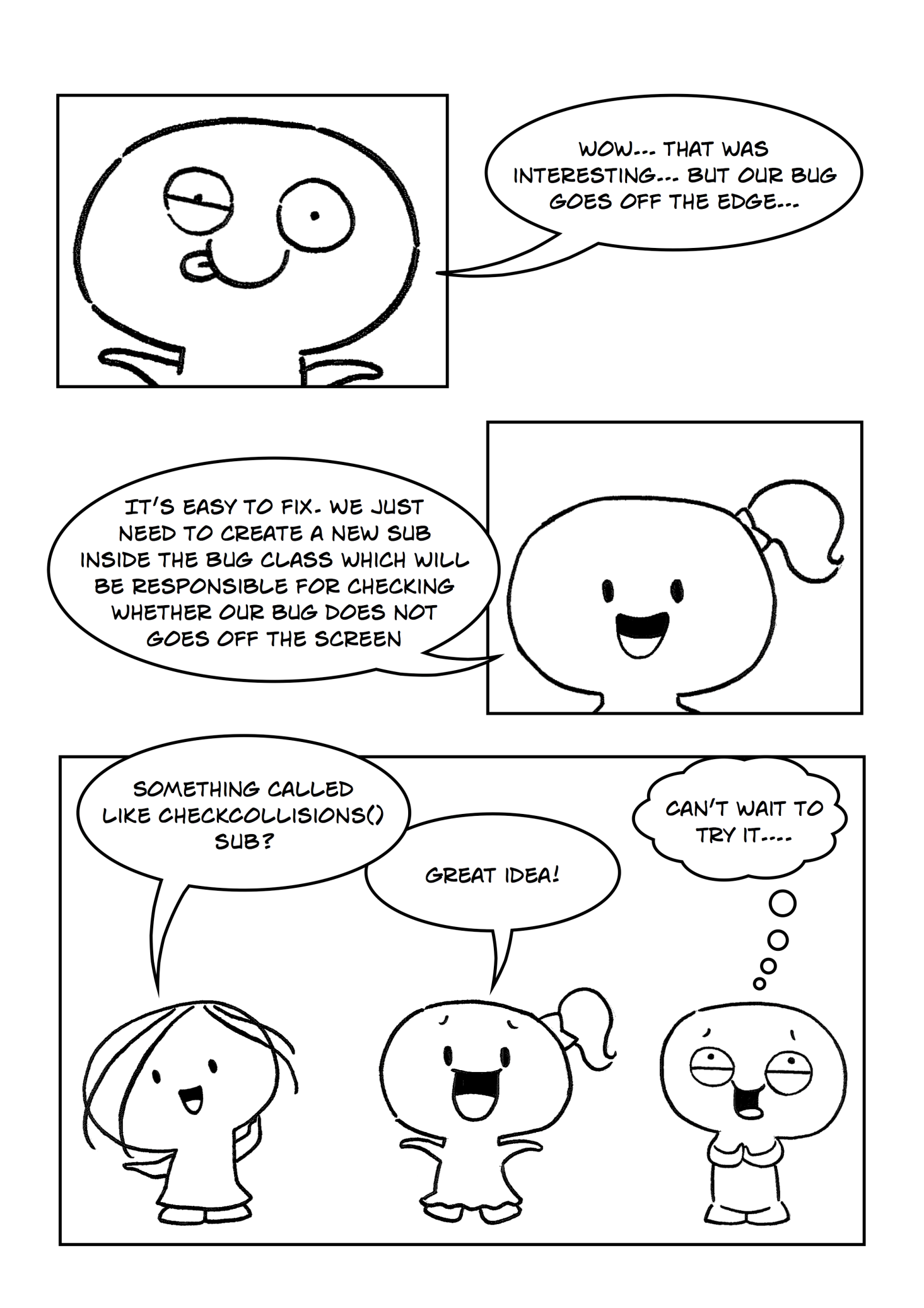
|  |
| --- |
| SwinGame.Graphics.ClearScreen(Color.White)  myBug.Draw()  myBug.Update() |

*Exercise 4: Creating an object*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

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

*cha 2 - worksheet.pngQuestion 3: Why are random numbers made by computers not “truly” random?*

****

## Checking Collisions

Now we need to have a sub inside the Bug class that will check if a bug has reached the edge of the screen and if it has reverse its direction play as sound when it does. To create this method inside the Bug class we will make Private Sub called CheckCollisions and put inside it almost the same code we used in Chapter 3 as below (notice this time we are using an “Or” in the If statements, this means instead of four if statements we only need two):

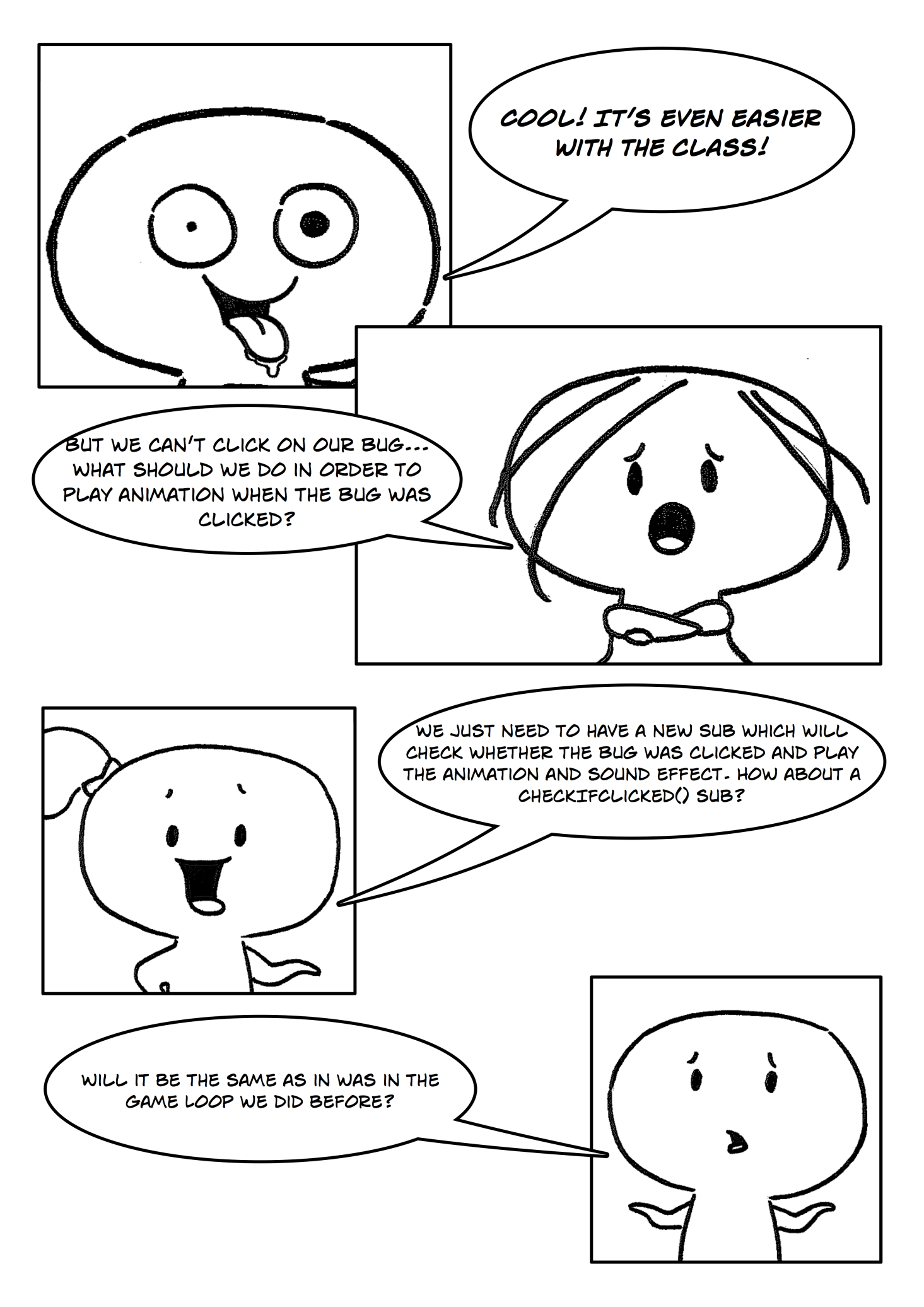
|  |
| --- |
| Private Sub CheckCollisions()  If AliveSprite.X + AliveSprite.Width >= Core.ScreenWidth Or AliveSprite.X <= 0 Then  AliveSprite.Movement.X = -AliveSprite.Movement.X  Audio.PlaySoundEffect(GameSound("hit"))  End If  If AliveSprite.Y + AliveSprite.Height >= Core.ScreenHeight Or AliveSprite.Y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

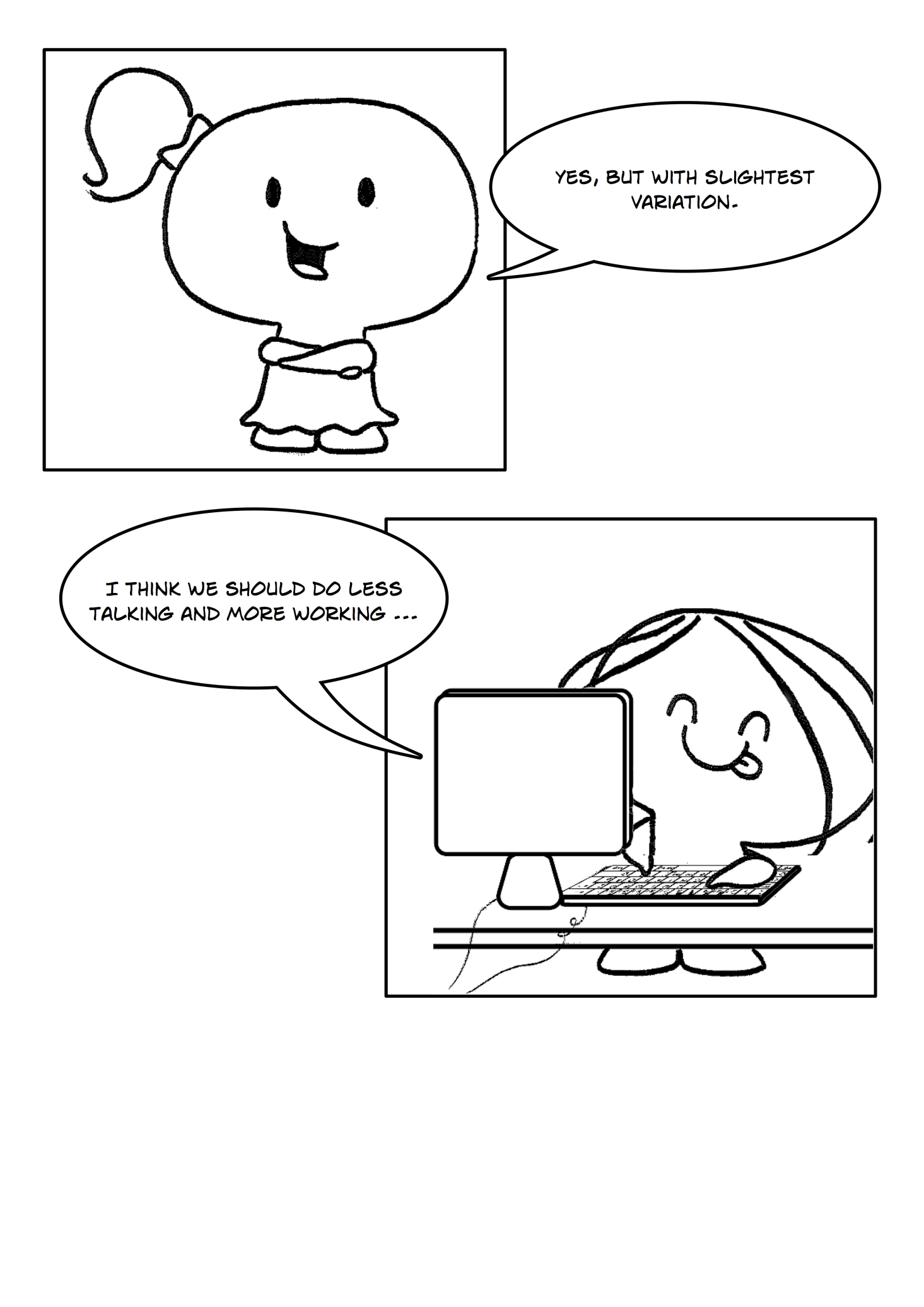
This sub is private as there is no need to call it from outside. All that we need now is to call CheckCollisions() after Graphics.UpdateSprite(AliveSprite) inside the Update() method in our Bugs class.

*Exercise 5: Creating CheckCollisions() method*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Create CheckCollisions() method inside the Bug Class (don’t forget to call it from the Update() method. Debug to see the result.

****

****

## Checking if Clicked

At this point our bug seems immortal – you cannot click on it to kill it and play the animation. So we need to create a new sub inside the Bug class that will check whether the bug was clicked.

To do so we need to create a new Public Sub called “CheckIfClicked()” and inside it first we have to declare mousePoint as a Point2D variable. Then we need to assign the current mouse position to the variable mousePoint (mousePoint = Input.GetMousePosition()).

Then we need to test if the sprite “IsAlive” and if sprite as at the same point as the mouse and the left button of the mouse was clicked (). If all of these things are true then the bug has been clicked and we can play the “splat” sound effect, set the Alive state to False and play the DeadSprite animation, set the DeadSprite animation to the same position as the Alive sprite and stop it after playing once.

The code here shows you how to do this; you will recognize most of the code from Chapter 6:

|  |
| --- |
| 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("splat"))  Alive = False  DeadSprite = Graphics.CreateSprite(GameImage("deadBug"), 40,10, 57, 43)  DeadSprite.X = AliveSprite.X  DeadSprite.Y = AliveSprite.Y  DeadSprite.EndingAction = SpriteEndingAction.Stop  End If  End If  End Sub |

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

*Exercise 6: Creating CheckIfClicked() method*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Create ChechkIfClicked() method inside the Bug class. Don’t forget to call the method from Update(). Debug to see the result.

saveicon.png Remember to save your project (File – Save All). Once you have finished then you can close Visual Studio or move on to the next chapter.