

Systems Development: Object Oriented Programming

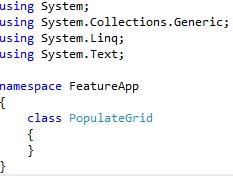
(H171 35)

WPF Application

Step 4 – draw the contents of the grid on the main application page

* Now let’s populate the grid that appears on the left hand side of the main application page with some elements. We will write the code for populating the grid in a different class, as you may want to be flexible and populate the grid differently depending on certain situations. We will add a new class called PopulateGrid. Be sure to update the “using” statements at the top of the file.

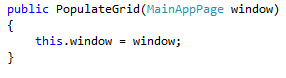




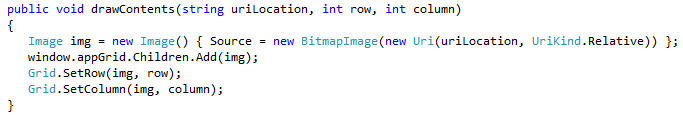
* In the PopulateGrid class add a member variable – a window containing a grid to be populated



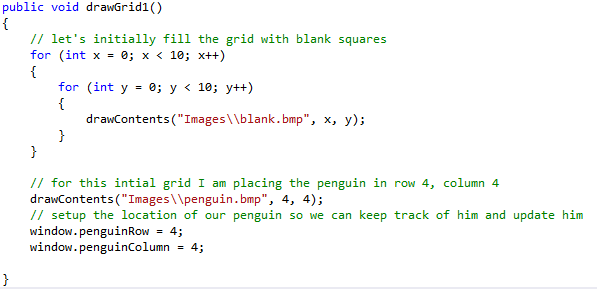
* Now create the constructor method for PopulateGrid which gets passed in a MainAppPage object (window containing the 10x10 grid) and sets the member variable window (use “this” to clarify that it is “window” belonging to this PopulateGrid class) to be this window passed in



* We are going to populate the grid with images, therefore the first thing we need to do is write a method that that demonstrates how to populate an individual cell within the grid with an image. This method will be called drawContents
  + You are essentially adding an item to the grid, which by default will always be placed in the first row and first column. Therefore after you have added the item to the grid you then set the row and column position for the item
* We are going to add an image to the grid therefore we need to firstly create the image, then add it to the grid and then position it:



* We want this PopulateGrid class to be able to populate a grid in lots of different configurations/layouts, so we will create separate methods to describe each different grid layout we want to create. In this tutorial we will only create one grid layout, but as discussed methods can be added to describe different layouts.
* The grid layout for this tutorial will be created in a method called drawGrid1()
  + It will initially fill the 10x10 grid with blank box images. These images can be found in the solutions “Images” folder and the bitmap is called blank.bmp. You must also create an Images folder in your project and add this image. When you add the image you need to set 2 of the blank.bmp properties
    - Build Action should be set to Content
    - Copy to output directory should be set to Copy always
  + We are then going to add a single penguin image to row 4 column 4 of the grid, which we will later on learn how to move this penguin around the grid
  + The final step is that we need to store the location of the penguin within the grid so we can keep track of it when we start moving it around.
    - The row and column will be stored in the MainAppPage class in 2 new member variables – penguinRow, penguinColumn



* Remember to update the MainAppPage class in MainAppPage.cs to include the 2 new variables to store the position of the penguin:

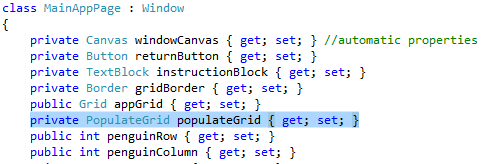




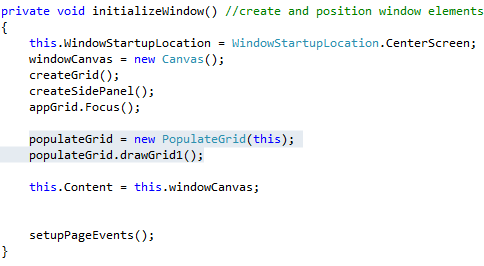
…..

…..

* We have now written the code that populates the grid, so we now need to call it when the main application page is launched i.e. call the code to populate the grid from the MainAppPage.cs intializeWindow() method
  + Add an instance of PopulateGrid to the MainAppPage class



* + In the MainAppPage.cs initializeWindow() method create an instance of PopulateGrid passing it the current(this) window we have created containing the 10x10 grid, and then call the method to draw our grid – drawGrid1()



RUN THE APP AND VERIFY THAT THE GRID ON THE MAIN APPLICATION PAGE IS DRAWN WITH BLANK SQUARES AND ONE PENGUIN

