**The Data package**

***The "gameString" class***

Summary: This class holds text that will be displayed on the screen.

Description: When you write console apps (non-graphical, command-line apps like we focused on all semester), you only have to worry about the string you want to print and then using System.out.println to display it.

Once you set your Java application into full screen graphical mode (known as GUI mode), this all changes. You gain the ability to intricately control how and where the text is displayed but the downside is complexity. It is far more difficult to set up and display text strings in GUI mode than in console mode.

I wrote the "gameString" class to simplify the process of displaying text to the screen. When you wish to display text to the screen in GUI mode, you need to know the following:

1. Color to display the string in
2. x, y coordinates of where to place the string
3. The string that needs to be displayed
4. The font object that defines the look, size, etc of the text

***The "Sprite" class***

Summary: This class holds a single image that needs to be displayed on screen at some point

Description: Sprites are 2D raster images that contain color (RGBA) information and positioning information that forms a graphical image. The following is needed for the sprite object:

1. x, y coordinates of where to display the sprite (coordinates are top-left of image)
2. The path and filename of the image file on disk that we load the "picture" from

Once the main Sprite buffer loads all images from disk at the program startup, we no longer load from disk as that would be slow and unnecessary. We then reference the sprites loaded into memory by a "sprite number" that works like indexes into a Sprite ArrayList to reference the specific sprites we want. The Sprite class is a class that deals with a SINGLE image. We will later make a robust class that will handle two databases of Sprites: one for loading all of the game's images into at startup and the other for being a temporary Sprite buffer for sending to the renderer every frame to display the current game frame.

# The Graphics package

**The "Graphic" class**

Summary: This is the class that gets your program into GUI mode in Java.

Description: This class is probably the lowest level of the whole project. It consists of largely boilerplate Java API method calls to get the program into GUI mode. It consists of setting full screen hardware flags, giving the desired resolution, checking to make sure the desired graphics mode is available, allowing the programmer to hide the default mouse cursor, and giving the programmer a way to attach keyboard and mouse action listeners to the graphics handle object.

You can learn a lot from delving into it but most of it is boilerplate for all GUI full screen programs. It is quite Java specific and doesn't have much Computer Science value other than just doing it to get the mode set that we want.

**The "Sprites" class**

Summary: This handles a database of Sprite objects.

Description: The purpose of this is to group together a collection of Sprite objects. This makes it easy to pass the reference to the renderer for displaying.

Since we keep the List object private, we pass an Iterator to the Sprite objects instead of the List itself.

We use two instances of this class in the program: one to hold all of the Sprites in the game (load from file into memory), and another to hold just the Sprites for the current frame. This is the object that is passed to the renderer, displayed, and erased for the next frame to start all over again.

# The Input package

**The "Keyb" class**

Summary: This handles the keyboard input (real time).

Description: Unlike console (non-graphical) apps that use in.readLine to retrieve entered text, GUI mode apps can utilize "Key Listeners" that can provide real-time feedback every time a key is pressed or released. This is essential to responsive feedback needed for games.

This class implements the Java API interface "KeyListener" which requires the class to override a few essential unimplemented methods. These methods are like hardware interrupts that are called by the Virtual Machine as soon as keyboard input is detected.

We can also "poll" our class every frame to check for newly processed keyboard events and respond accordingly. That is what this class deals with.

**Note: In my personal version of the game engine, I also have a class that manages mouse input. I didn't include it here as it adds more complexity and is too much to talk about in a single module. I hope to teach a game development course here one day that can go over a more robust game engine that includes mouse input processing.**

# The time package

**The "stopWatch" class**

Summary: I made a mistake in the naming conventions and my class names are reversed from their intended purpose. Oh well! This class handles timing set intervals between events.

Description: Using the System timer behind the scenes, this class takes a duration and then counts down from that. You can continue to poll the object to see if the time is up (e.g. isTimeUp method). Once the time is up, you can respond to that by adding in your next action.

I use this class to time animation frames. Animation, at its core, is just a series of images and a timer. Once an image is displayed for x time, we switch the image (and sometimes position) and time that. This class works to help me accomplish animation timing.

**The "Timer" class**

Summary: This is used for coarser granularity timing like a game's "gameplay" clock. I usually use this object to track how long a user has been playing.

Description: Not much to say about this class other than the summary. You can look at the methods that are part of this class and figure out what is going on with it.

# The Main package

**The "LoadTest" class**

This is just a tester class I created to test the load rates of multiple different image file formats. One of the bottlenecks of a game engine is in the initial loading time of the assets. Therefore, I wanted to test how long it takes to load 1000 relatively large images of a given file type.

My tests concluded that bmp files are the fastest, jpgs are the next fastest, and png are the slowest. I only tested those three file formats but you are welcome to run your own tests.

This particular class may not benefit you but I left it in because it was a neat example of running your own tests to determine efficiency over just relying on rules of thumb or third party information.

**The "Main" class**

This is the class that puts everything together to form our little demo. The code is commented as detailed as I could think to. This shows how to set the GUI mode (1280 x 720 resolution for this example), set up a key listener, set up sprite and text buffers, set up a game loop, and render both sprites and text to the screen at the desired positions.

I encourage you to run this program but also play around with (or "tweak") the parameters to make things change. You can learn by modifying code that is already there and seeing how it changes the end result. It is also fun to get a feel for things when incremental changes are noticeable.

Have fun!