Bell Schedule App
Advanced App Development 2015

High Level Code View

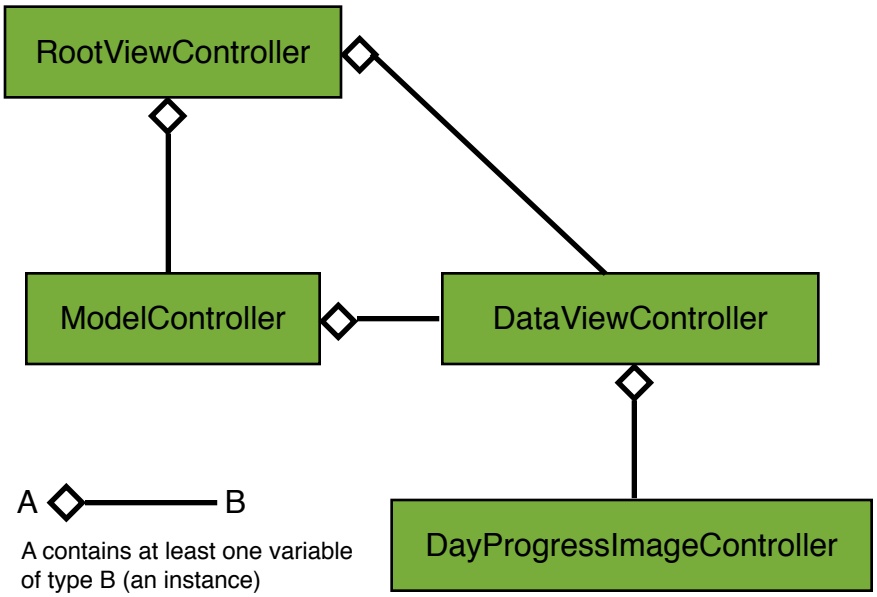
Day

Period

Schedule

These three class represent types tat are used throughout the project. A **Day** object represents a day on the calendar (day, month, year). A **Period** object represents a section of time where students need to be somewhere (class, passing period, early bird, etc). A **Schedule** is a list of periods.

The four classes below control the main display of the bell schedule app. The **RootViewController** is the base view controller. It contains a UIPageViewController, which allows us to have the page turning effect when switching between days. For a UIPageViewController to work, it needs both a **ModelController**, which controls which page is currently being displayed, and a **DataViewController**, which controls the data that is displayed on that page. While displaying a bell schedule, the progress through the schedule is displayed using a green background image, which is created by the **DayProgressImageController**.

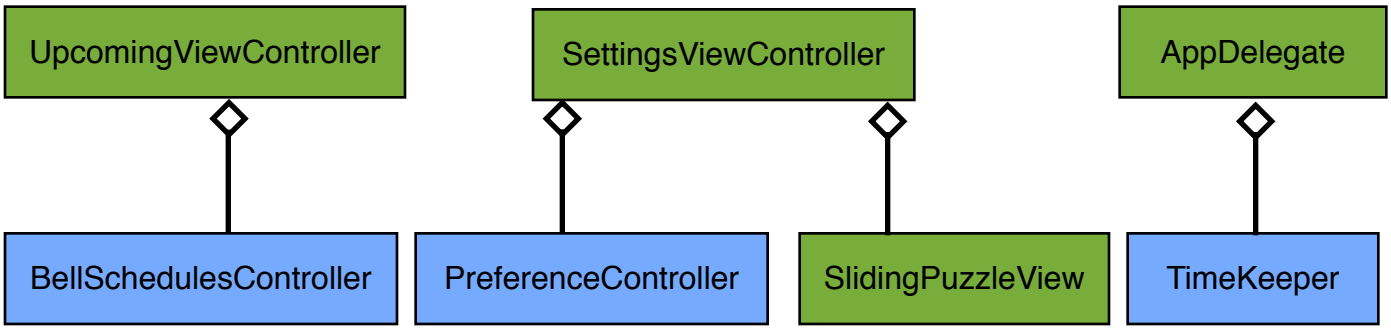


Below are seven other classes. The light blue class (**BellSchedulesController**, **PreferenceController**, and **TimeKeeper**) are singleton classes - meaning that there is only one instance of that class in the program (in fact, the constructors are private!). The advantage of singleton classes is that they are easy to access from every other object in the program. In fact, the singleton classes are used by almost every other object in the program (so the connections below are not exhaustive by any means).

The **BellSchedulesController** manages all the **Schedule** objects in the program. It is tasked with downloaded information about bell schedules (what periods are on those schedules, what days have those schedules, etc) and providing outside classes with the ability to grab those Schedules by name or Day.

The **PreferenceController** manages all configurable settings about the app. This includes what Early Bird schedule should be used (there are three) and whether the user should be notified a minute before class is supposed to start.

The **TimeKeeper** is the heart beat of the app. It tics every second and updates the displayed schedule (what period we are in, should we move to the next schedule, etc).



The **UpcomingViewController** displays a list of all the upcoming special schedules in a table. Selecting a special schedule teleports the user to that Day on the calendar so that the schedule can be looked at in more detail.

The **SettingsViewController** displays the settings that the user currently has enabled, as well as a fun sliding tile game in the **SlidingPuzzleView**.

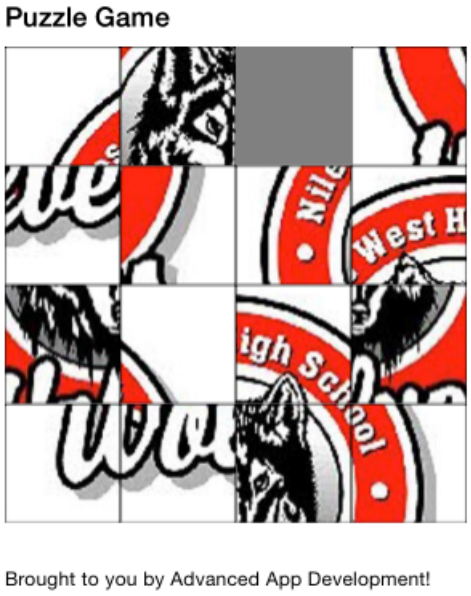
The **AppDelegate** receives all application wide events, such as opening, coming to the foreground, going to the background, etc. Among other things, the AppDelegate tells the TimeKeeper to stop ticking when the app goes into the background and to start ticking again when it comes back.

For non-singleton classes to communicate with each other, we use the NSNotifcationCenter, a class that allows one object to post a notification of an event that is then delivered to other objects that register with the notification center to listen for those events. Below is a table of these events:

Class	Listens For...	Posts...	Data Dictionary
RootViewController	SchedulesLoaded DateChanged		
SettingsViewController		EarlyBirdChanged	
UpcomingViewController		DateChanged	key: date value: "MM/dd/yyyy" (String)
BellSchedulesController		SchedulesLoaded	
DataViewController	EarlyBirdChanged SchedulesLoaded	DateChanged	
DayProgressImageController	TimePassed		
TimeKeeper		TimePassed	key: "percent" value: 0.0 to 1.0 (Double)

Classes and Methods to be Finished

SlidingPuzzleView [3 methods] - you can see this in the Settings Window



drawRect()

This function should loop over every image in the **images** 2D array instance variable and draw the images at the correct spots. Each image will have a width and height that is exactly one-fourth the width and height of the view. The command to draw an image at a point is:

img.drawAtPoint(p)

Where p is a CGPoint object and img is a UIImage. If you do this correctly, you should see the Niles West wolf logo in unscrambled form.

randomize()

The randomize function randomizes the positions of pictures in a 4x4 grid of **images**. One of these images will be the **grayImage**, at **grayRow**, **grayCol** in the images 2D array. The purpose of this function is to prepare the classic tile sliding game.

For a 4x4 grid, doing about 100 swaps is enough to give you a good randomization. The problem with just swapping any two tiles is that it could create a board that is not solvable. The solution is to only swap tiles that are adjacent to the **grayImage** (make sure to update the **grayRow** and **grayCol** as well!).

touchesBegan()

This function is called whenever the user touches the sliding puzzle game. If the user has selected an image that is adjacent to the **grayImage**, then the **grayImage** and the selected image should swap (updating **grayRow** and grayCol as well). To determine which image the user has clicked on, it will be helpful to determine which row and column they have selected, as the given point will be the pixel location. Knowing that each image has a width and height that is one-fourth the width and height of the view will also be helpful.

Period [10 methods]

init(String, Int, Int, Int, Bool, Bool)

The Period class has two constructors (the first is already done). Your job is to finish the second constructor. Like all constructors, your job is to initialize the instance variables to the given parameters. The problem is that you are not given the endHour and endMin for the Period. Instead, you are given the startHour, startMin, and the length of the period. Here are some examples for you to consider in your code:

	Ex 1	Ex 2	Ex 3
startHour	10	10	12
startMin	30	30	30
length	20	40	50
endHour	10	11	1
endMin	50	10	20

You should notice that we are doing times as AM and PM (not military). We are going to assume for this project that any hour from 6 to 11 is AM, and any hour at 12, and from 1 to 5 is PM

zeroPad(Int)

This function takes an Int that may either be two digits or one digit. This function should return a String that represents the same Int, but with a 0 added to the front if the Int is single digit. So zeroPad(3) would return "03" and zeroPad(45) would return "45"

toString()

This function returns a string representation of the Period in the following format:

periodName startHour:startMin - endHour:endMin

You will want to use the zeroPad function as well to make sure that both startMin and endMin are two digits.

convertToMinutes(Int, Int)

This class function (notice that it is static), take two Int parameters (hour and minute). The method should return the number of minutes this time is from 12:00 AM. What makes this function interesting is that we will assume that any hour between 6 and 11 represents an AM time, and 12, 1 through 5 represents a PM time. Here is a table of examples for you to consider

	Ex 1	Ex 2	Ex 3	Ex 4
hour	6	11	12	2
minute	30	50	20	10
minutesPassed	390	710	740	850

timeDistance(Int, Int, Int, Int)

This class method returns the number of minutes between to the four time parameters (hourA, minuteA) and (hourB, minuteB). It will be very useful to convert the two pairs of times into minutes first...

distanceBetween(Period)

This instance method returns the number of minutes between this (called **self**) Period's end time and the parameter Period's (**other**) start time. It will be useful to use the timeDistance() method...

percentThroughDay(Int, Int, Int, Int, Int, Int)

This method takes 6 Int parameters - (startHour, startMin) for the day, (endHour, endMin) for the day, and (currentHour, currentMin). This method should return a number between 0.0 and 1.0 that represents how far the current time is through the day. The timeDistance() method will be useful...

containsTime(Int, Int)

This method takes two parameters - an hour and a minute. The method should return if the time represented by the parameters falls within this period (between startHour, startMin and endHour, endMin).

isAfterTime(Int, Int)

This method takes two parameters - an hour and a minute. The method should return if this Period's (startHour, startMin) comes after the parameter time.

length()

This method returns the number of minutes in this period (from (startHour, startMin) to (endHour, endMin))

Schedule [2 methods]

setEarlyBird(String)

This function is called to update the Periods that are in the current Schedule. The **allPeriods** instance variable has all the periods that COULD be in the schedule. There are three early bird periods (A, B, and C), but only one will be shown on the current schedule based on the user's preference (they can also choose to show no early birds). To make this work, there is another instance variable named **periods**, which contains all the active periods (ones the user actually wants to see), and also includes passing periods (these are not in the **allPeriods** instance variable). When this function is called, it takes one parameter - **name** - the name of the Early Bird that the user wants.

To finish this method, please do the following:
a) remove all the objects from the **periods** NSMutableArray object
b) loop over every Period object in the **allPeriods** array
    i) if the next period is an early bird period, only add it to the **periods** array if its **periodName** property is equal to the **name** parameter.
    ii) if the period is not an early bird period and it does not overlap with the previous period (if any) in the **periods** array (the Period class has some functions that will help with this), then add the period to the **periods** array.
c) Now you need to add passing periods. For every pair of consecutive periods in the **periods** array, find the distance between the end of one period to the beginning of the next, and add a new Period between them with the name of "Passing", a start hour and minute the same as the end hour and minute of the one period, and a length equal to the distance between the two periods.

lengthOfDay()

This method returns the total length of the current Schedule in minutes. This can be found be determining the first and last period of the day and finding the time distance in minutes between the beginning of the first period and the end of the last period (The Period class has some functions that can aid with this...)

BellScheduleController [5 methods]

readInDays()

this function reads in all the days that have a special schedule from an online google form.

	A	B	C	D
1	Timestamp	Select the date of the schedule	Select the schedule	Username
2	6/9/2015 17:47:16	8/11/2015	Opening Day	matfah@d219.org
3	6/9/2015 17:47:41	9/4/2015	Late Start	matfah@d219.org
4	6/9/2015 17:48:32	9/7/2015	Non Attendance Day	matfah@d219.org

You can see that the initial column is the time that the form was submitted (we will not use this). Column one contains the day that there will be a special schedule in the format MM/dd/yyy. The second column contains the name of the special schedule (Opening Day, etc). The third column contains the name of the d219 employee who submitted the schedule change. This is important as we want to restrict schedule changes to a few people (like the principal, assistant principal, etc).

This method already has a lot of code written for downloading the data from the spreadsheet. Your job will be to parse out the relevant pieces of data and store them in the **dates** instance variable (of type NSMutableDIctionary). [Note - in this section of code, you will have to write **self.dates** instead of just **dates** - it has to do with the fact that I've technically written a function within a function - the same goes for all other instance variables]

You will begin in the else case - you should already see a local variable named **strData** of type NSString. This will contain the data from the google form in a comma separated values (CSV) format. For example:

> Timestamp,Select the date of the schedule,Select the schedule,Username\n
> 6/9/2015 17:47:16,8/11/2015,Opening Day,matfah@d219.org\n
> 6/9/2015 17:47:41,9/4/2015,Late Start,matfah@d219.org\n

This would be the format that the google form will be encoded as in a String - notice that there is a new line character (\n) between lines, and commas (,) between data elements. Also notice that the initial row is useless and you will want to disregard.

The **strData** variable supports a command called **componentsSeparatedByString(s)** - this command allows you to take an NSString and "split" it around a separator s. The result of this command is an NSArray with all the components. You could split the entire strData into the individual rows by the "\n" separator, and then for each row, split around the "," separator to get individual columns.

For each row (except the initial row), you will need to extract the date, schedule name, and username. If the username is not in the **legalEmails** instance variable (it's an array), then this row should be ignored. Otherwise, the **dates** dictionary should have the date associated with a the schedule name [use **dates.setObject()**, where the schedule name is the object, and the date is the key).

Additionally, the program needs to know what the earliest scheduled day and the latest schedule day is. There are already two instance variables, **firstDate** and **lastDate**, that try to track that information. They are currently set incorrectly (the **firstDate** is set to a date way in the future, and the **lastDate** way in the past). Your job is to see if any of the dates you have just found predate the **firstDate** variable, or postdate the **lastDate** variable, and if so, update those instance variables accordingly. To compare your date (which is a String) with **firstDate** and **lastDate** (which are Day objects), you will need to create a Day object from your date String, then use the **compareWith()** method of the Day class.

readInSchedules()

this function reads in all the period schedules from an online google form.

	A	B	C	D
1	**Schedule Name**	**Period Name**	**Period Start**	**Period Length**
2	Regular	Early Bird A	7:05	42
3		Early Bird B	7:20	42
4		Early Bird C	7:53	59
5		1st	8:10	42
6		2nd	8:57	42
7		3rd	9:44	42
8		Homeroom	10:31	10
9		4th	10:46	42
10		5th	11:33	42
11		6th	12:20	42
12		7th	1:07	42
13		8th	1:54	42
14		9th	2:41	42
15	Late Start	Early Bird A	8:45	32
16		Early Bird B	9:00	32
17		Early Bird C	9:23	51
18		1st	9:40	34

Your job is very similar to what you did in readInDays(). You will already have a **strData** local variable that contains the period schedule data in a CSV format. For each row that has a schedule name that is not the empty string, you will need to create a Schedule object with that name and add it to the **schedules** instance variable (NSMutableDictionary) using the **setObject()** command using the name of the schedule as the key.

For every row (except the initial, which you skip), you also need to extract information so you can create a Period object - you will need the name, start hour and minute (which you can split out of the time column by splitting around the ":" separator), the length, and whether the period is an early bird period (only true if the name begins with the word "Early"). With all that information, you can construct a Period object and add it to the Schedule that you are currently building by using the **addPeriod()** method.

dayIsWithinSchedule(Day)

This method has a Day parameter (**date**) - the method should return true if **date** is between (inclusive) the **firstDate** and the **lastDate** instance variables (the Day class has a method that can help with this...)

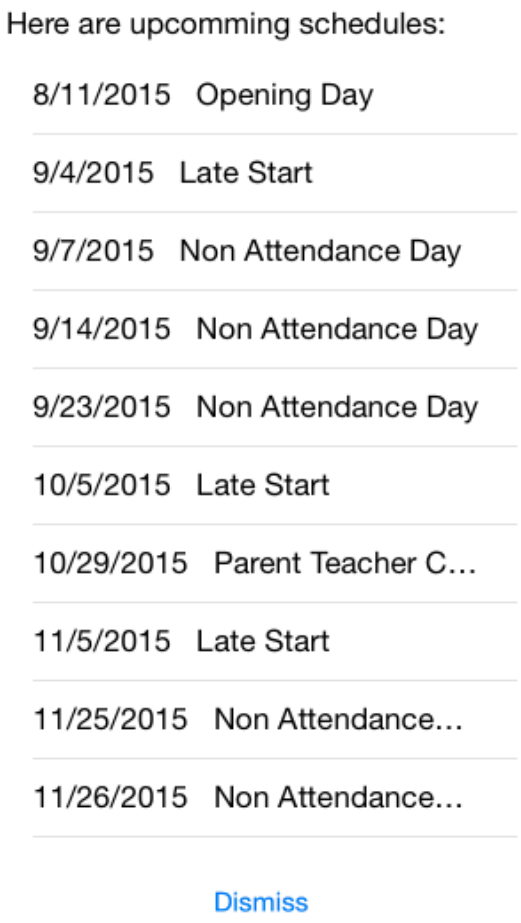
dayIsBeforeEndOfSchedule(Day)

This method has a Day parameter (**date**) - the method should return true if **date** is before or on the **lastDate** instance variables (the Day class has a method that can help with this...)

scheduleNameFor(Day)

This method has a Day parameter (**date**) - the method should find which Schedule (if any) in the **dates** dictionary. If there is a schedule, then the name of the Schedule is returned. If there is no schedule, then if the **date** is on the weekend (there is a method for this), then "Weekend" is returned. If it is not on the weekend but the **date** is within the schedule, then "Regular" should be returned. Otherwise, "Non Attendance Day" should be returned.

UpcomingViewController [1 method] - you can see this in the Settings Window



viewWillAppear()

This method needs to organize all the upcoming dates in date order. The local **keys** variable (of type NSArray) contains all the dates as String objects. Your job is to fill the **sortedArray** variable (of type NSMutableArray) with all the same date Strings, but in ascending order.

A couple of suggestions. First - you can use the **Day** class to construct **Day** objects of the given Strings. This is useful because Day objects have a **compareWith()** method that will tell you which Day object is smallest (earliest). Additionally, you can use the **toString()** method of a Day object to get the original String object back.

Second - you can sort the Days manually using one of the sorts you learned in AP Computer Science (Bubble, Selection, Insertion, etc... but please no Bogo!).

TimeKeeper [1 method]

update()

this method is called every second to update where we are in the current schedule. You should notice that there is an if case already written for you - all your code will go in here. The if case tried to get the current schedule (it could be nil) - inside the body of the if case, you can assume that we have a valid schedule to consider. The **currentSchedule** instance variable is then set, and some fancy code is written to access the current hour and minute, stored in the local variables **hour** and **min**.

Your job is to determine what Period (if any) we are currently in (and set the **currentPeriod** instance variable accordingly), and then to also determine what percent through the current day's schedule we are, and assign that value to the **percentDayPassed** local variable as a number between 0 and 1.

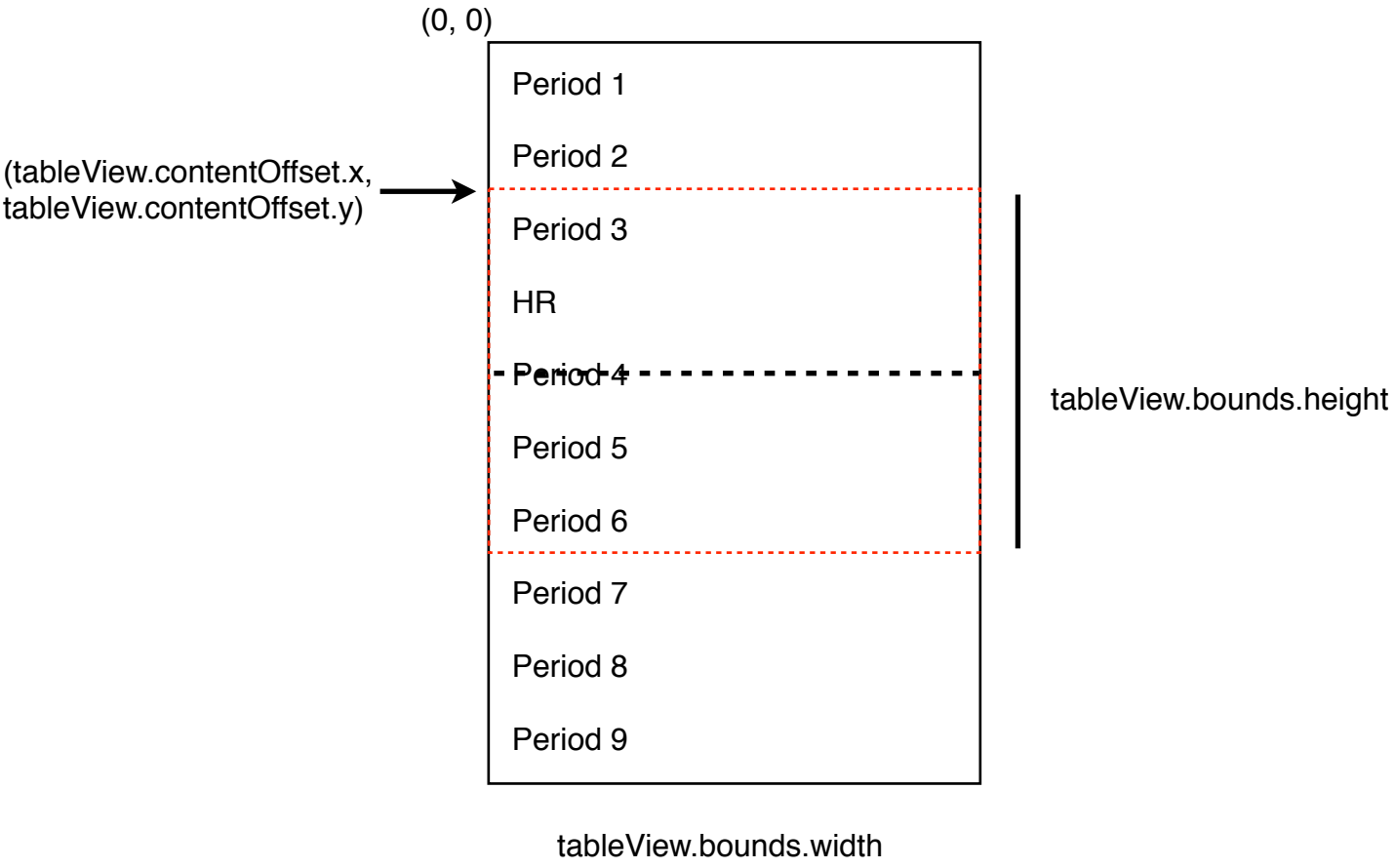
To accomplish this task, first check if the **currentSchedule** has any periods - if not, then currentPeriod should be set to nil, and percentDayPassed should be set to 0. If there is at least one period, then you need to determine if the current **hour** and **min** are before the first period or after the last period in the **currentSchedule**. In either case, the **currentPeriod** should be set to nil, but in the former case the **percentDayPassed** should be set to 0, while in the former it should be set to 1. If the current **hour** and **min** is within a period, you need to loop over all the periods int he **currentSchedule** and determine which one, assigning **currentPeriod** appropriately. Also, you will need to set the **percentDayPassed** - this can be accomplished by using the Period classes **percentThroughDay**() method.

DayProgressImageController [2 methods]

update()

This method is called to update our progress through the day as a **percent**. For example, as school usually starts at 8:10 and ends at 3:23, there are a total of 433 minutes in the day. At 10:00, 110 minutes have passed, and 110 / 433 is roughly 0.25 or 25%.

Below is a picture of the situation. We have a **tableView** (the red, dashed box) with a given width and height. It contains content that is larger than the **tableView** (the black box) - in fact, it will always be twice as tall as the **tableView**. The dashed black line represents where we are in the day (what **percent** through the day we are - 0% would have the line at the top, 100% at the bottom). The black dashed line should be centered in the red box.

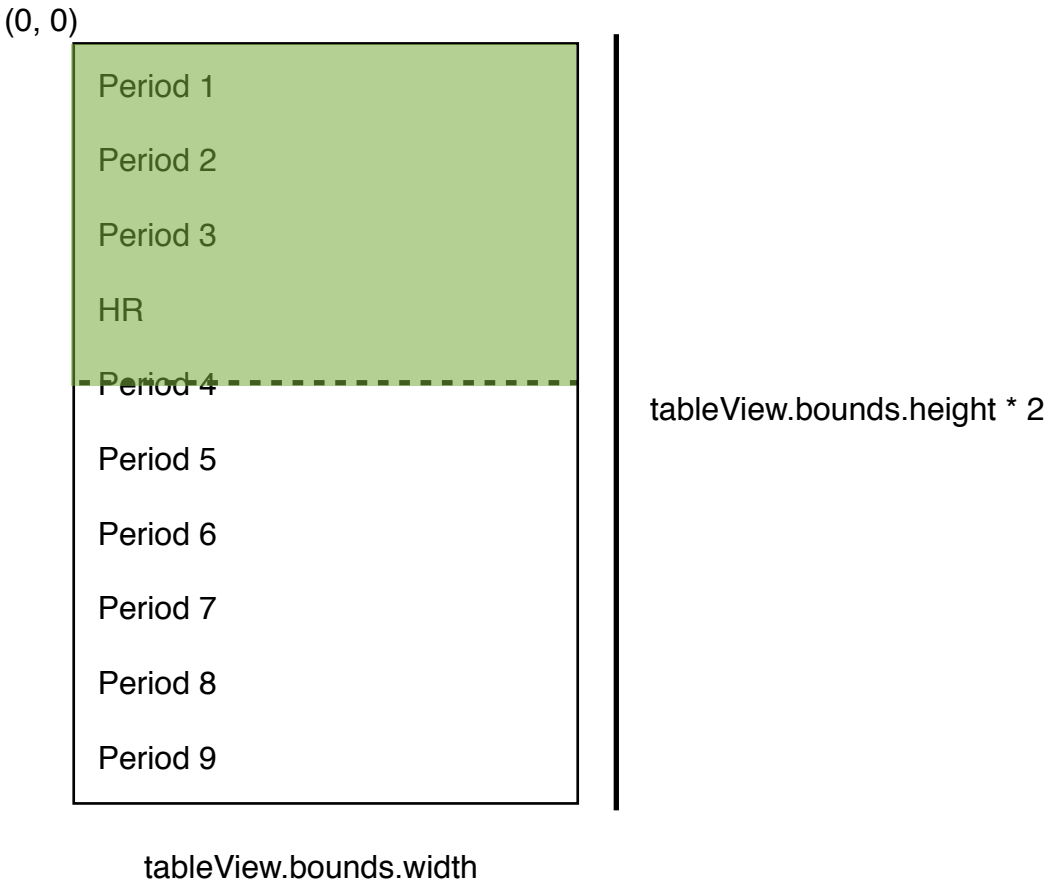


Your job is to assign the **contentOffset** (it is of type CGPoint) property of the **tableView** instance variable based on what **percent** (another instance variable) through the day we are. CGPoint's have both an x and y coordinate.

There are two special cases - the red box should not go above the top of the content area or below the bottom. For example, if the **percent** through the day was 0%, the **contentOffset** y coordinate should also be 0, not some negative value. This means that the red box should NOT be centered on the black dashed line when we are near the beginning or end of the day.

getImage()

This method generates a colored image to be drawn on the background of our table view. For example, if the black dashed line represents our **percent** progress through the day, then the green box at the top represents how much of the day has passed, and the white box at the bottom represents what is yet to come.



To fill a rectangle in swift, you will need to create two UIBezierPath objects that represent rectangles. As an example, here is code that creates a UIBezierPath that represents an arrow.

After creating the path, you need to set a color and fill it. The relevant commands are:

```
UIColor.greenColor().set()              //sets the drawing color to be green
nameOfBezierPathObject.fill()           //fills a UIBezierPath object with the color green
```