CS 371M: Homework 2 Peck

Overview. Once again you are developing a game. The idea for the game is that the user clicks a button and a sequence of English words appears in the area under the button. Also, each word appears individually (with a light grey background) in the game area, splayed out pretty randomly. You have to click each word in the sequence order in the time allotted. If you click a word out of order, the word flashes red.

- This assignment stresses string manipulation and it is an opportunity to dig a bit deeper into how to write Kotlin programs. There are many ways to solve this programming assignment, so I encourage you to find your own way. If you find the Buddha on the path, slay him or her. The path is for you.
- Your app has a single activity. The code as given has build errors.
- I'm going to go over the requirements for the homework, progressively adding detail. You get credit for functionality, so even if you have corner cases that aren't handled correctly in your code, you will still get partial credit.

Algorithms. There are two major algorithms in this assignment: (1) picking the words and (2) laying them out in the play area. I looked into AI to generate funny English sentences, but I couldn't find anything that I really liked. Instead, I chose part of a novel that is public domain and whose text I could get from Project Gutenberg. That novel is Pride and Prejudice by Jane Austin, which you should check out at some point. She is a wonderful author. The movie Mansfield Park (1999) made me into a fan and turned me on to her books.

Picking words. The API you are given is

```
fun pickWords(start:  Int, numWords:  Int) :  List<String>
```

You must implement this function, which takes `start`, a character offset into the Pride and Prejudice text and returns a list with `numWords` entries. Sounds pretty easy, right? The basics are pretty easy, but we have some details to consider.

For the first detail, let's look at how our text begins.

```
const val PrideAndPrejudice = """
PRIDE AND PREJUDICE

By Jane Austen
```

There is a newline at PrideAndPrejudice[0] and the letter P at PrideAndPrejudice[1]. What should you do if I give you `start = 3`? I don't want my first word to be "IDE" because that isn't a word.

Therefore, you must follow this algorithm: find the first whitespace character on or after your start position. In the case where `start = 3` that means you look at the `I`, the `D`, the `E` and finally you see a space character, which is white space (white space also include tabs and newlines). Once you see white space, continue to scan forward in the string and when you find the first non-white space character after the white space, then you start your first word. So `findWords(0, 1)` returns a list containing "PRIDE", but `findWords(1,1)` and `findWords(2,1)` return a list with "AND".

- **Words only.** We want words, not punctuation and not white space, so please remove all punctuation and white space specified in the punctSpaceStr in Words.kt. Do not change this string to add or remove punctuation or white space. Kotlin strings to have an `isWhitespace` function and my solution uses that.

- **Duplicates.** Duplicate words provide an additional challenge. Because you have to click on words in order, it is annoying for your user if they get something like `he said that he`. The word "he" will appear in two different boxes forcing the user to guess which "he" is the first one and which is the second one. Therefore, we number multiples in a list, starting with the second occurance so `findWords` would return this list, `[‘‘he’’, ‘‘said’’, ‘‘that’’, ‘‘he (1)’’`. Your code should cope with any number of duplicates, don't set a maximum based on Pride and Prejudice.

  You might think it is more logical to return `[‘‘he (1)’’, ‘‘said’’, ‘‘that’’, ‘‘he (2)’’`, but I didn't think that looked as good.

- **Efficiency.** Part of being a good software engineer is to be aware of the "cost" of your code in time, space, and complexity. Please don't read all of the Pride and Prejudice text into a data structure. For example, please don't build a huge list including every word. That is a waste of memory and it is not necessary. We will deduct points if your solution is unreasonably resource intensive.

- On the other hand, fiddly code that looks at each character individually and tries to remember where it is in the string is difficult to get correct and difficult to maintain. So I encourage you to use convenience functions, but just be aware of their cost. Remember, in Kotlin as in Java, creating a large substring from a read-only string is very cheap—the string just contains a pointer to the underlying storage.

- Please consult with the instructor or the TA if you are unsure whether your approach is inefficient. Primarily we are concerned with you splitting the entire text into individual words.

- **Testing.** One thing that makes testing your code hard is the difference between an average case and an unusual case. For this project, a randomly chosen string from our text tends to be "simple." But in order to do a good job testing your code, you need to find parts of the text where (for example) there are repeated words (see Duplicates above). What other cases should you test? I highly suggest doing non-random testing on the word picking function to make sure you get back what makes sense. For example, if you ask for 5 words from just before the last two words of the text, you only get back those two words.

Placement. We need to place the word boxes randomly in the play area, but we don't want the words to overlap because that would be messy. I considered placing the words one by one and checking for overlap, but you have to check for overlap with every existing box, which makes layout an N^2 operation that isn't even guaranteed to terminate (though it will terminate with very high probability if placement is random).

Then I found a posting on Stack Overflow about doing a random partition of a rectangle using randomly sized rectangles. Once the partition is done, you can choose any of them to get non-overlapping rectangles. That is a cool idea that is linear in the size of the play area, but now I have different sized rectangles to deal with. I didn't want to have to measure word lengths and make sure I was picking a rectangle that would fit the text (or shrinking the font of the text to make it fit).

I tell you this whole story because a good programmer does enough work to make things right, but isn't afraid to change the problem statement a little to make his or her life easier.

- For the y position of your text, split your play area into a sequence of rows, each exactly high enough to display a TextView with font sized 18sp. If you have a partial row left over, don't use it (because if you did, your textview would be cut off). For each word in your sequence choose a random row and set the y position to that row. Make sure that no two words are in the same row. While that means we won't ask you to place more TextViews then there are rows in the game area, remember that your `pickWords` function must be written to accept any number for `numWords`.
- For the x position, pick a random number, but make sure your text box is at least 8dp from both the start and end border.
- This x,y position is not strictly random. But I think it looks good.

Files. Let's go over the files in the project. All locations where you need to write code are marked with `// XXX write me`. You can trust me, there won't be unmarked areas that you have to change.

- **MainActivity.kt** This is where your code first gets control from the Android framework which launched your app in response to the user clicking the icon. You can modify `durationMillis` during testing if you like.

  You can leave `onCreate` and `doScore`. In `newGame` create an instance of Timer and one of Words. Figure out how to initialize Words.

- **Timer.kt** Do nothing here. If you come up with a better visual effect than I did, post an animated gif to piazza and we will marvel at your creativity.

  We will not use your version of this file when evaluating your code so DO NOT modify it.

- **AndroidManifest.xml** Do nothing here.

- **content_main.xml** What is here is correct (and should not be changed), but you need to add four borders and the play area. Each border is 12dp thick, colored black and is visible at all times. Please bear in mind that the play area should NOT overlap with the frame. It consists of the area **inside** the frames only and it is a FrameLayout.

  I mark the area where you need to write XML like this <!- - Need borders and play area - -> because that is how to write comments in XML.

  Does this feel familiar? At least I changed the layout a little bit.

- **Words.kt** This class requires the most work. Let's start in `playRound`, which calls `pickWords` and then needs to display the words in the `sentenceTV` (with a single space between each word and no spaces on the ends). It then should place each word in a dynamically created TextView that is displayed in the play area according to the description above.

  You need to figure out how to detect that the TextViews are clicked in order. I found that to be the most challenging part of the lab, so don't start there. If the user successfully clicks all TextViews in order, you should call the `wordsDone` function that is passed to `playRound`. That function if given to you—see the pretty Kotlin lambda!

  If the user clicks a TextView out of order, you must call `outOfOrderPick`. You can marvel at that function's implementation–no need to modify it.

  `pickWords` should adhere to the description above. My solution has several helper functions.

`createTextView` should create a TextView with font size 18sp, and the text passed in as a parameter. It should have 8 pixels of padding, and `neutralBgColor` as its background color. Note, these can be 8 device pixels. There is a way to go from device independent pixels (the "dp" we use in our XML layouts) to device pixels (the actual pixel values of the phone on which your code is executing), but that is more complication then we need for this lab.

We provide (simple) layout parameters that you should use. Finally, for the first 6 TextView objects that you create, assign them ids like this.
`textView.id = ids.getResourceId(index, 0)`.

You can add class variables if you want.

- **PrideAndPrejudice.kt** This one needs nothing. Except maybe zombies.

Please test your code thoroughly. We give you valid stubs in ExampleInstrumentedTest.kt and WordTest.kt. We especially encourage you to use the latter to test your `pickWords` function. You can just write tests to validate all of the corner cases that this document describes. These tests will run even without an emulator!

No posting code. For this homework, PLEASE NO POSTING CODE. Don't post any code to piazza or any other forum. I like the discussions better when there is no code. If you are having a specific problem with your code, you can always send your code to me or the TA.

No posting tests. PLEASE NO POSTING TESTS. A lot of this homework's functionality involves what list of words you return when given certain start parameters. I want each student to grapple with how to write these tests. You all need to practice this skill—specification by testing. I don't want a small set of motivated students to write tests for everyone else. You can ask about specific situations, but just make up a string and talk about offsets and word lists like I do in the homework writeup.

Submission. All submissions should be done via git. Refer to the git setup and submission documents for the correct procedure. The root directory of your Android studio project should contain your README file.

There is NO code collaboration for homework. Each student must do their own coding and they must do all of their own coding. You can talk to other students about the problem, you can talk to the instructor or TA. If you discuss the homework deeply with someone, note that in your README.

Please do not discuss specific functions or APIs. Don't post code to public forums, though you can mail code to the instructor or TA. You can talk to each other about classes, but leave it at that level.

README file. Just modify the one we give you. It should be in plain text and named README (not README.txt). It should be in the root directory of your submitted files. It includes these items.

1. Your name.
2. Your eid.
3. Your email address.
4. How many hours you worked on this assignment.
5. Are you using any slip days.
6. The names of anyone you spoke with intensely for this assignment.
7. Any comments for the grader.