Midterm 3 Solution

CS 1323, Fall 2017

Name (printed legibly): _____

Student number: _____

Integrity Pledge

On my honor, I affirm that I have neither given nor received inappropriate aid in the completion of this exercise.

Signature: _____

Do not write anything on the back of any page. If you need extra space use the blank page at the back of the examination. If you pull the test apart, please put all of the pages in order when you hand your exam in. M

Answer all programming questions in Java. Show your work to receive partial credit.

Pay attention to whether the program requires you to use an ArrayList or an array.

Pay careful attention to what is requested: a code fragment (a few lines of code), a method, or a program.

Since you do not have the whole API available during the examination, it is acceptable to guess at method names, parameters and return values. If your guesses are reasonable—even if not perfect—you will receive credit. If, however, you make up new methods that are not reasonable for the class or that magically solve problems the class cannot solve, you will not get credit.

You do not need to import packages or throw exceptions in any methods.

Relevant parts of the API are given on the back of previous pages and in a separate document.

From the String class:

replace

```
public String replace(char oldChar,
              char newChar)
```

Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

If the character `oldChar` does not occur in the character sequence represented by this `String` object, then a reference to this `String` object is returned. Otherwise, a new `String` object is created that represents a character sequence identical to the character sequence represented by this `String` object, except that every occurrence of `oldChar` is replaced by an occurrence of `newChar`.

Examples:

```
"mesquite in your cellar".replace('e', 'o')
        returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
        returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
        returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```

Parameters:

`oldChar` - the old character.

`newChar` - the new character.

Returns:

a string derived from this string by replacing every occurrence of `oldChar` with `newChar`.

From the StringBuilder class:

reverse

```
public StringBuilder reverse()
```

Causes this character sequence to be replaced by the reverse of the sequence. If there are any surrogate pairs included in the sequence, these are treated as single characters for the reverse operation. Thus, the order of the high-low surrogates is never reversed. Let *n* be the character length of this character sequence (not the length in `char` values) just prior to execution of the `reverse` method. Then the character at index *k* in the new character sequence is equal to the character at index *n-k-1* in the old character sequence.

Note that the reverse operation may result in producing surrogate pairs that were unpaired low-surrogates and high-surrogates before the operation. For example, reversing "\uDC00 \uD800" produces "\uD800\uDC00" which is a valid surrogate pair.

Returns:

a reference to this object.

1. (15 points; 3 points each for a) and b); 9 points for c))

a) What does the String object greeting contain after the following code is executed?

```
String greeting = new String("Happy");
greeting = greeting.replace('H', 'h'); // see API on back of previous page
```

happy

b) What does the StringBuilder object description contain after the following code is executed?

```
StringBuilder description = new StringBuilder("Long");
description.reverse(); // see API on back of previous page
```

gnoL

c) Write a method that changes all of the String objects stored in an oversize array to lower case. The signature is below:

The API for a useful method is given on the back of this page.

```
public static void toLowerCase(String[] data, int dataSize)
{
      for (int index = 0; index < dataSize; ++index)
      {
            String element = data[index];
            element = element.toLowerCase();
            data[index] = element;
      }

}
```

From the String class:

toLowerCase

```
public String toLowerCase()
```

Converts all of the characters in this `String` to lower case using the rules of the default locale. This is equivalent to calling `toLowerCase(Locale.getDefault())`.

Note: This method is locale sensitive, and may produce unexpected results if used for strings that are intended to be interpreted locale independently. Examples are programming language identifiers, protocol keys, and HTML tags. For instance, `"TITLE".toLowerCase()` in a Turkish locale returns `"t\u0131tle"`, where '\u0131' is the LATIN SMALL LETTER DOTLESS I character. To obtain correct results for locale insensitive strings, use `toLowerCase(Locale.ENGLISH)`.

Returns:

the `String`, converted to lowercase.

See Also:

`toLowerCase(Locale)`

2. (15 points; 5 points for algorithm, 10 for sorting) Sort the values below using either insertion sort or selection sort. You must indicate which algorithm you used and show the process exactly as it was in class***, with each individual swap or move of data shown on a separate line***. **Only data that are moved or changed should be shown.**

I used (circle one): selection sort insertion sort

Selection sort

3	1	4	2	9	8	Aux
1	3					
	2		3			
		3	4			
			4			
				8	9	

Insertion sort

3	1	4	2	9	8	Aux
						1
	3					
1						
						4
		4				
						2
			4			
		3				
	2					
						9
				9		
						8
					9	
				8		

Do not write anything on the back of any page

3. (15 points) Trace the code fragment below in the table below. Only show changes to the data.

```
int[] heights = {7, 1, 6, 3, 2, 5};
for(int start = 0; start < heights.length-1; ++start)
{
      for (int swapMe = start; swapMe < heights.length-1; ++swapMe)
      {
            if (heights[swapMe] > heights[swapMe+1]) // swap
            {
                  int temp = heights[swapMe];
                  heights[swapMe] = heights[swapMe+1];
                  heights[swapMe+1] = temp;
            } // end if
      } // end for swapME
} // end for start
```

start	swapMe	data[0]	data[1]	data[2]	data[3]	data[4]	data[5]
0	0	1	7				
	1		6	7			
	2			3	7		
	3				2	7	
	4					5	7
	5						
1	1		3	6			
	2			2	6		
	3				5	6	
	4						
	5						
2	2						
	3						
	4						
	5						
3	3						
	4						
	5						
4	4						
	5						
5	5						

4. (15 points; 1 point for a), 2 points for b), 3 points each for c) and d) 6 points for e)) Use the documentation for the PageFormat class (given in a separate document) to answer the following questions and perform the following tasks.

   In parts c) and d), if there are no methods of the required type, say "None."

   a) Write the import statement that would be needed to use this class.

   import java.awt.print.PageFormat;

   b) Construct a PageFormat object. Store the reference in a variable.
   PageFormat pf = new PageFormat();

   c) List two mutator methods in the PageFormat class.

   setOrientation, setPaper

   d) List two accessor methods in the PageFormat class.

   Choose two from: getHeight, getImageableHeight, getImageableWidth, getPaper, getWidth, getImageableX, getImageableY, getMatrix, getOrientation

   c) Given a PageFormat object with reference pictures, write a code fragment that changes the PageFormat to landscape orientation if the object is not already in that orientation. If the PageFormat object is already in landscape orientation, it should be left alone.

```
if (pictures.getOrientation() != PageFormat.LANDSCAPE)
{
    pictures.setOrientation(PageFormat.LANDSCAPE);
}
```

5. (40 points; 15 points for a), 15 points for b), 10 points for c)) Write a program that creates the Stow application, which helps you pack for trips. It takes the type of trip, the number of days of the trip and gives you a list of things you need to pack. Stow also keeps a list of things that are essentials for you personally (e.g. glasses, granola bars, water bottle) and includes them on the list of things that need to be packed.

The trick to making this all work is having lists of necessary items stored in files. These files come with the app. For each type of trip there are two files: one of things that only need to be packed once, and another of things that need daily copies. The file names are the type of the trip. There also is a file of essential items that must be included on every trip.

The software will add items to a list of unpacked items. The items in Trip-Type.txt will be added to the list once, the items in Trip-Type Daily.txt will be added once for each day of the trip, and the items in Essentials.txt will be added once. This is your starting list of unpacked items.

The two files for a Business Trip, and Essentials are shown below:

Business Trip.txt	**Business Trip Daily.txt**	**Essentials.txt**
Dress shoes	Underwear	Cell phone charger
Nightgown	Outfit	Glasses
	Socks	Toothbrush

A two day business trip using the lists above would pack the following items: Dress shoes, Nightgown, Underwear, Outfit, Socks, Underwear, Outfit, Socks, Cell phone charger, Glasses, Toothbrush.

When you pack, you move items from the unpacked list to the packed list. When you are finished print out the packed and unpacked lists

The software interface is shown below. Bold entries were made by the user. As always we assume that our user doesn't make any mistakes.

```
What kind of trip will you take?
Business Trip
How many days will you be gone?
2
Enter the item you packed or Quit to exit
Socks
Enter the item you packed or Quit to exit
Socks
Enter the item you packed or Quit to exit
Glasses
Enter the item you packed or Quit to exit
quit
You have packed: [Socks, Socks, Glasses]
You have not packed: [Dress shoes, Nightgown, Underwear, Outfit, Underwear,
Outfit, Toothbrush, Phone charger]
```

a) Write the createList method. This method should ask the user for the kind of trip and the number of days. Then the method should add items in the two files that match that trip type into an ArrayList<String>, and add items in the essentials file into the ArrayList<String>.

Your method must use the method below, that takes an ArrayList<String> and a file name and adds each line in the file to the end of an ArrayList<String>. **Do not write the addToList method.**

public static void addToList(ArrayList<String> list, String fileName) throws FileNotFoundException

The signature for the method you will write is below.

```
public static ArrayList<String> createList(Scanner keyboard) throws
FileNotFoundException
{
	ArrayList<String> unpacked = new ArrayList<String>();

	// First put in the unique items
	System.out.println("What kind of trip will you take?");
	String tripFileName = keyboard.nextLine();
	addToList(unpacked, tripFileName + ".txt");

	//Add in the daily items once for each day of the trip
	System.out.println("How many days will you be gone?");
	int days = keyboard.nextInt();
	keyboard.nextLine();

	for(int count = 0; count < days; ++count)
	{
		addToList(unpacked, tripFileName + " Daily.txt");
	}

	// Add in the personal essentials
	addToList(unpacked, "Essentials.txt");
	return unpacked;
}
```

b) Write the pack method. This method takes an ArrayList<String> that contains the unpacked items and a second ArrayList<String> that contains the packed items. This method should allow the user to repeatedly enter items that have been packed. Each entered item should move from the unpacked list to the packed list. The method signature is below.

```
public static void pack(ArrayList<String> unpacked, ArrayList<String> packed,
Scanner keyboard)
{
	// Priming read
	System.out.println("Enter the item you packed or Quit to exit");
	String item = keyboard.nextLine();

	while(!item.equalsIgnoreCase("Quit"))
	{
		// Remove the item from unpacked
		unpacked.remove(item);

		// Add the item to packed
		packed.add(item);

		//Priming read
		System.out.println("Enter the item you packed or Quit to exit");
		item = keyboard.nextLine();
	}

}
```

c) Complete the main program. You must call the methods you wrote in a) and b). The signature of these methods is given below.

public static ArrayList<String> createList(Scanner keyboard) throws FileNotFoundException
public static void pack(ArrayList<String> unpacked, ArrayList<String> packed, Scanner keyboard)

```
public static void main(String[] args) throws FileNotFoundException
{
	Scanner keyboard = new Scanner(System.in);
	ArrayList<String> packed = new ArrayList<String>();
	ArrayList<String> unpacked = createList(keyboard);

	pack(unpacked, packed, keyboard);

	System.out.println("You have packed: " + packed);
	System.out.println("You have not packed: " + unpacked);
	keyboard.close();
}
```