CO320 In-Class Assessment 5
Version 2018.11.14.01

The session time will be 80 minutes except where an individual's ILP allows for proportionately more time.
Assessment Conditions
Please read carefully

The session will start at 5 minutes past the hour and late arrival will only be permitted at the discretion of the session supervisors and the time of arrival will be recorded. The examiners reserve the right to award a mark of zero in the case of late arrival.

You will not be permitted to leave before the end of the session. If you anticipate finishing early it would be wise to bring a book to read to fill in the remaining time.

The assessment will be conducted under exam-like conditions and violations will be treated under the University's disciplinary procedures.

The primary condition is that you must not seek the help of anyone else in completing this assessment.

Please make sure you strictly follow these rules:

- Switch **off** your mobile phone and music players – don't just mute them.
- You may use a programming environment, such as BlueJ, Netbeans, Eclipse, or similar.
- You may use a web browser but **you must not** access Web sites **outside** **`kent.ac.uk`** with the *sole exception* of the Java API at **`docs.oracle.com`**:
  - You can access the Java API from the following URL: https://docs.oracle.com/javase/8/docs/api/
  - You may access the CO320 Moodle page.
  - You may access your local filestore and past assessment work.
  - **You may not** search the Web for answers or to look up compiler error messages, for instance.
  - **You may not** access files via dropbox.com, google.com, or any other file-sharing web site.
- You must not use any form of email or messaging during the session.

- You may use and access any of the course resources, such as PowerPoint slides, existing anonymous Q&A material, course-related projects and assignments.

- You may bring and use a single Java textbook, which may be annotated.

- You may use your own laptop provided that it is powered by battery and you allow class supervisors to view the screen at all times during the sessions. It is your responsibility to ensure that your laptop is adequately charged for the duration of the session.

If you are in doubt about what might be permitted, ask the session supervisor.

Once you have completed the assessment, you must create either a zipped copy of your project folder or a JAR file from within BlueJ and upload it to the Assessment area on the Moodle page.

Make sure that you know how to create a JAR file from with BlueJ or a zipped copy of a folder in advance of the session. To create a JAR file:

- Select *Project/Create JAR file*.
- Ignore the *Main class* dropdown.
- Ignore the checkboxes for *Include user libraries*.
- Tick both *Include source* and *Include BlueJ project files*.
- Select *Continue*.
- Save the file to your Desktop with the name assign5.jar
- Upload assign5.jar to Moodle.

There is a short, soundless video on how to create a JAR file from within BlueJ at https://www.cs.kent.ac.uk/~djb/co320/create-jar-file.mp4.

If you wish to check the contents of the jar file to make sure everything required is in it, *change the .jar suffix to .zip* and then uncompress the file. It is safe to ignore any operating system warning about changing the suffix. A JAR file is just another name for a ZIP file.

Any attempt to submit after the end of the session will result in a mark of zero being awarded and might result in disciplinary action being taken.

In your answers to these questions you must adhere to good and consistent style and you must include a descriptive comment for each field, constructor and method, using javadoc style where appropriate. Comments inside the bodies of methods and constructors are not required.

1. Write a class called `Idea` that represents details of an idea for something to do on someone's 'bucket list'. It has one field of type `String` called `activity` and one field of type `int` called `cost` and one field of type `boolean` called `done`.

In the `Idea` class write:

- One constructor that takes three parameters that are used as the initial values of its fields.

- An accessor method for each field.

- A method called `ticked` that sets the `done` field to `true`.

- A method called `getDetails` that takes no parameters and returns a `String`. It returns a `String` containing the values of the fields in the following format:

  ```
  "Rock climbing costs 150"
  ```

  where `"Rock climbing"` is an example of the value of the `activity` field and `150` is an example of the value of the `cost` field. If the `done` field is set to `true` then the `String` must also include `" (ticked)"` at the end; e.g.,

  ```
  "Rock climbing costs 150 (ticked)"
  ```

  Different `Idea` objects will have different values for these fields.

2. Write a class called `BucketList` that is able to store a `LinkedList` of `Idea` objects.

In the `BucketList` class write:

- A method called `addIdea` that takes an `Idea` object as a parameter and adds it to the list.

- A method called `getSize` that returns the number of `Idea` objects in the list.

- A method called `list` that prints out the details of all the `Idea` objects stored in the list, one per line. Use the `Idea` objects' `getDetails` method to format the details. The print out must **not** include the index of each `Idea` in the list.

- A method called `isInList` that takes a `String` as a parameter and searches the list for an `Idea` whose `activity` is an exact case-insensitive match for the parameter. It must return `true` if a match is found and `false` otherwise. The `activity` `"Rock climbing"` would give a case-insensitive match for parameter `"ROCK CLIMBING"` but not for either `"Rock"` or `"climbing"`.

  Note: You are **not** permitted to use either a `break` or `return` statement **inside** any loop control structure in this method.

3. In the `BucketList` class write:

- A method called `removeExpensiveItems`. This method takes a single `int` parameter. It finds all `Idea` objects in the list whose `cost` is greater-than the parameter's value and removes them from the list. The method must store all the removed `Idea` objects in a new `LinkedList` that is then returned from the method. No error message is to be printed if no `Idea` objects are removed from the list but an empty list must still be returned.