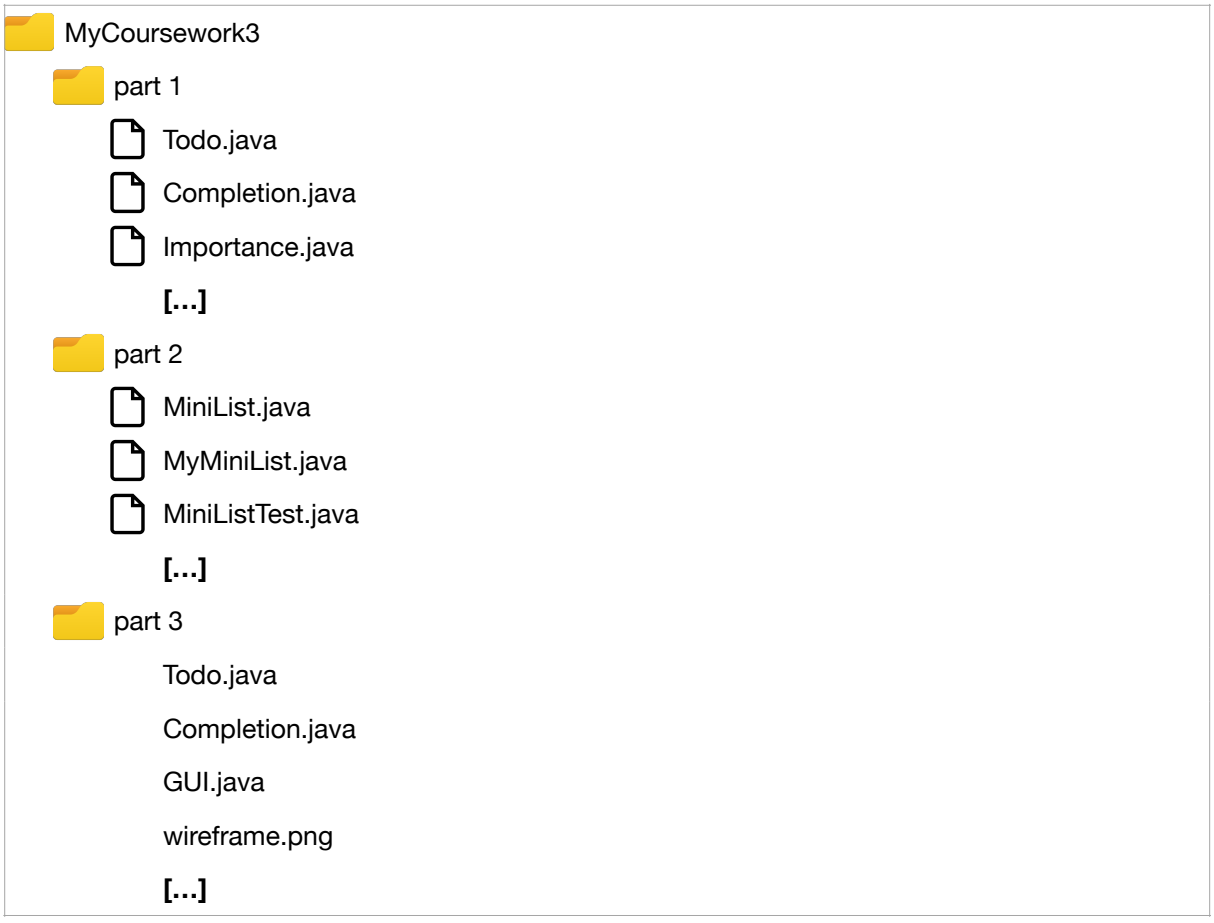
Principles and Applications of Programming: Coursework 3

This coursework is out of 60. There are 20 marks available for each part.

Please upload the source files (all the files ending .java) and any design work you have done. Don't include unnecessary folders or files including class files. Organise your submission so there is 1 folder for your todo application and 1 folder for the MiniList. For example:



Part 1: Todo List CLI [20 marks]

In this part you will write a simple todo list application. You will be able to interact with your application through the command line. See the coursework video to see a finished application in practice and the end of the . To complete the application you will need to write 3 classes and 3 enumerations as follows:

Name	Type	Marks
Todo	class	4

Name	Type	Marks
CLIMenu	class	7
Main	class	1
Category	enum	4
Status	enum	2
importance	enum	2

Status enum

The status enumeration includes the following enumeration constants:

- pending
- started
- partial
- completed

Importance enum

The status importance includes the following enumeration constants:

- low
- normal
- high

Category enum

The category enum should include a single private instance variable for each enumeration constant of type string that stores an ASCII colour value for displaying the TODO to the list. Call this variable colour You will therefore need a constructor to set the initial value of the colour and a getter method. Include the following enumerations and colour values:

- red: "\033[0;31m"
- white: "\033[0;37m"
- blue: "\033[0;34m"
- purple: "\033[0;35m"
- yellow: "\033[0;33m"
- green: "\033[0;32m"

Todo Class

The todo class represents the individual todos in your application. It should include the following member variables, a constructor to set their initial values, getter and setter methods for each and a toString method.

- private String text;
- private LocalDateTime due;
- private Category cat;
- private Importance importance;
- private Status completion;

Your toString method should return a string with details of the todo, instead of displaying the category as a string, use the colour value from the category to output in the correct colour. e.g.

```
return cat.getColour() + "Todo{" +
        "}\033[0m";
```

By including \033[0m at the end of the string you reset the colour back to default for any subsequent printing.

NOTE: The colours will not show on the standard Windows console. You should see them in IntelliJ regardless of platform.

CLIMenu Class

This class will handle the menu system for the application. It's constructor takes a single argument. An arrayList of type Todo to store the list of todos in. Assign the value of this argument to an instance variable of the same type. To understand the menu system further review the coursework guide video and the appendix at the back of this document, which includes the application working.

The main menu will have the following options.

1. List todos
2. Add todo
3. Update todo
4. Delete todo
5. Quit

List todos should list all the todos entered in so far. You should present the todos as a numbered, so when a user wishes to delete or update a todo they just enter the required number from the list.

Add todo should ask the user the following series of questions to be able to create a new todo:

- What is the title of the todo
- What date is the todo due
- What category does the todo belong to (list the enumerated values for category and ask the user to pick between them)
- What is the priority of the todo (list the enumerated values for priority and ask the user to pick between them)

Use the information gathered from the user to add a new todo to the arrayList. You can assume a new todo has the status pending.

Update todo should ask the user which todo they wish to update by entering its number from the list above. The menu then asks which field of the todo to update, such as its title, due date or category. The user is then prompted to enter new data for that particular field. The values are then updated for that todo in the ArrayList.

Delete todo asks the user which todo they wish to delete by entering its number from the above list. The todo is then removed from the list.

Quit exits the program.

Although there is quite a lot of parts to the menu system, carefully constructed code will can greatly reduce the code redundancy. Make sure you have limited code repeated and organise into functions where possible.

Main Class

The final class in the todo list application is a Main class that include a main method. This method should create an ArrayList of type Todo to store the Todos in. This should be passed as an argument to the constructor of a new instance of the CLIMenu class.

Part 2: MiniList collection [20 marks]

In this question you are going to produce a simplified collection type called `MyMiniList`. Use the template on learn, gold to get started. MyMiniList will work in a similar way to ArrayList. Internally, it will use an array, you will have to reorganise and grow the array to as the MiniList is updated through use of its methods.

Inside the template is an interface called MiniList. MyMiniList must implement this interface and be implemented with a single generic type.

Also included in the template is a simple set of tests for you to verify your implementation against. You will receive the following marks for correctly implementing each of the interfaces methods and a constructor.

Your class will need two instance variables, 1 for the current size of the list (an int) and an array of generic type T to store the elements of the list call this objectStore.

public MyMiniList() [2 marks]

initialise the array of elements, this is a little tricky as Java Generics doesn't support arrays. There are a couple of approaches you can use. This is the easiest and is moderately type safe, however, IntelliJ will give you some warnings:

```
objectStore = (T[]) new Object[10];
```

Initially, set the size of this array to be 10. Also set your size variable to be 0.

public void add(T element) [3 marks]

Elements are added to the end of the objectStore array if the array has space. If the array doesn't have space you will need to create a new array twice the size now required, copy in the existing elements and replace the existing objectStore with this new array. System.arrayCopy is useful to achieve this but there are other methods you could use. You can then add in the new element and update the size variable.

public T get(int index) [2 marks]

Check that the index given is with the required bounds. You can use the built in method Objects.checkIndex to do this (no imports required). If the index is valid return the item at the index.

public int getIndex(T element) [2 marks]

Search through the list for the first occurrence of the element given as an argument. If found return its index. Otherwise return -1.

public void set(int index, T element); [2 marks]

Check if the index is valid. If so set the value at the index to be the element given in the second parameter.

public int size(); [1 mark]

Return the size of the list. (Make sure you return the size of list not the size of the underlying array)

public T remove(int index); [4 marks]

Check if a valid index has been given. If so remove the element and return it. You will need to move down the end of the array after the removal point so there is no gap and adjust your size property accordingly.

public boolean remove(T element); [4 marks]

Search the list for the first instance of the element and remove it. Again you will need to shift the array down to fill the gap left. TIP: you might be able to use the previous method to help you.

public void clear(); [1 mark]

Empty the list and reset the size to 0.

Part 3: TodoList extension and GUI [20 marks]

Expand on the TodoList application from part 1. We would like you to start by creating a GUI for the application. The format that the GUI takes on is up to you and marks will be award for an initial design and the realisation of that concept. Further to this, extend the application by including a file saving and loading mechanism so that todos persist between application executions.

Todo List GUI [10 marks]

- **3 marks** for design and realisation of it. Include a wireframe sketch of your design with your submission.
- **3 marks** for the application of Swing components and containers.
- **4 marks** for use of event handling

Todo List file persistence [10 marks]

You can approach this in any number of ways. An optimal solution might allow the user to select the file that they want to load in each time the program is launched. Alternatively you might wish to store the todos in a database (file based only, such as SQLite.)

- **5 marks** technical difficulty of the solution
- **5 marks** correctness of implementation

Appendix: Todo menu system output

User input is preceded by a ➜

```
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 2
Enter a title for the todo
➜ Take out the rubbish
Enter a due date for the todo in the format YYYY-MM-DDTHH:MM
➜ 2021-03-09T16:30
Select a category for the todo
Select an item between 1 and 6 and press enter
1. red
2. white
3. blue
4. purple
5. yellow
6. green

➜ 3
Select an importance for the todo
Select an item between 1 and 3 and press enter
1. LOW
2. NORMAL
3. HIGH

➜ 2
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 1
1. Todo{text='Take out the rubbish', due=2021-03-09T16:30, importance=NORMAL,
completion=pending}
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit
```

➜ 2
Enter a title for the todo
➜ Another test todo
Enter a due date for the todo in the format YYYY-MM-DDTHH:MM
➜ 2021-03-09T17:30
Select a category for the todo
Select an item between 1 and 6 and press enter
1. red
2. white
3. blue
4. purple
5. yellow
6. green

➜ 4
Select an importance for the todo
Select an item between 1 and 3 and press enter
1. LOW
2. NORMAL
3. HIGH

➜ 3
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 1
1. Todo{text='Take out the rubbish', due=2021-03-09T16:30, importance=NORMAL, completion=pending}
2. Todo{text='Another test todo', due=2021-03-09T17:30, importance=HIGH, completion=pending}
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 3
Which todo do you want to update?
➜ 1
Select an item between 1 and 5 and press enter
1. title
2. due date
3. category
4. importance
5. completion

➜ 5

Select a status for the todo
Select an item between 1 and 4 and press enter
1. pending
2. started
3. partial
4. completed

➜ 4
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 1
1. Todo{text='Take out the rubbish', due=2021-03-09T16:30, importance=NORMAL, completion=completed}
2. Todo{text='Another test todo', due=2021-03-09T17:30, importance=HIGH, completion=pending}
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 4
Which number todo do you wish to delete?
➜ 1
Select an item between 1 and 5 and press enter
1. List todo
2. Add todo
3. Update todo
4. delete todo
5. Quit

➜ 5