Command Oriented Personal Information Manager

This assignment involves the creation of simple Personal Information Management system that can deal with 4 kinds of items: todo items, notes, appointments and contacts. Each of these kinds of items is described in more detail below. The assignment requires that you create a class for each item type, and that each class extends an abstract base class provided for you. In addition to creating the four classes, you need to create a manager class that supports some simple text-based commands for creating and managing items.

Each of your 4 item type classes will be derived from the following abstract class:

```
public abstract class PIMEntity {
    String Priority; // every kind of item has a priority

    // default constructor sets priority to "normal"
    PIMEntity() {
        Priority = "normal";
    }

    // priority can be established via this constructor.
    PIMEntity(String priority) {
        Priority =  priority;
    }

    // accessor method for getting the priority string
    public String getPriority() {
        return Priority;
    }
    // method that changes the priority string
    public void setPriority(String p) {
        Priority = p;
    }

    // Each PIMEntity needs to be able to set all state information
    // (fields) from a single text string.
    abstract public void fromString(String s);

    // This is actually already defined by the super class
    // Object, but redefined here as abstract to make sure
    // that derived classes actually implement it
    abstract public String toString();
}
```

PIMTodo

Todo items must be PIMEntites defined in a class named **`PIMTodo`**. Each todo item must have a priority (a string), a date and a string that contains the actual text of the todo item.

PIMNote

Note items must be PIMEntites defined in a class named **`PIMNote`**. Each note item must have a priority (a string), and a string that contains the actual text of the note.

PIMAppointment

Appointment items must be PIMEntites defined in a class named **`PIMAppointment`**. Each appointment must have a priority (a string), a date and a description (a string).

PIMContact

Contact items must be PIMEntites defined in a class named `PIMContact`. Each contact item must have a priority (a string), and strings for each of the following: first name, last name, email address.

There is one additional requirement on the implementation of the 4 item classes listed above, the 2 classes that involve a date must share an *interface* that you define. You must formally create this interface and have both PIMAppointment and PIMTodo implement this interface.

PIMManager

You must also create a class named `PIMManager` that includes a `main` and provides some way of creating and managing items (from the terminal). You must support the following commands (functionality):

- **List:** *print a list of all PIM items*
- **Create:** *add a new item*
- **Save:** *save the entire list of items*
- **Load:** *read a list of items from a storage*

When creating a new item it is expected that the user must response to a sequence of prompts to enter the appropriate information (and even to indicate what kind of item is being created). Do this any way you want, just make sure that your system provides enough information (instructions) so that we can use your systems!

There is no required format for the user interface, anything that allows users to create, list, save and load is fine. Here is what it might look like (user input shown in red):

```
java PIMManager

Welcome to PIM.
---Enter a command (suported commands are List Create Save Load Quit)---
List
There are 0 items.
---Enter a command (suported commands are List Create Save Load Quit)---
Create
Enter an item type ( todo, note, contact or appointment )
todo
Enter date for todo item:
04/26/2020
Enter todo text:
Submit java homework.
Enter todo priority:
urgent
---Enter a command (suported commands are List Create Save Load Quit)---
List
There are 1 items.
Item 1: TODO urgent 04/26/2020 Submit Java homework.
---Enter a command (suported commands are List Create Save Load Quit)---
Save
Items have been saved.
---Enter a command (suported commands are List Create Save Load Quit)---
Quit
```

Alternative PIMEntity

Reimpletement `PIMEntity` as `interface`, add change the rest of all code to use it.

Storage options (seperate projects)

1. **array version:** a simple array with size 100.

   Note that there is not a required Delete command. Feel free to use any data structure you want to hold a list of items, you are allowed to use a simple array with size 100 (you are not required to support lists of more

than 100 items). Handling array bound exception is required.

2. **collection version:PIMCollection class**

This option involves the creation of a custom collection object that can be used to manage PIMEntities (from array version). You must create a class named PIMCollection that implements the interface Collection. In addition to being a Collection, your class must implement the following methods:

getNotes

```
public Collection getNotes();
```

`getNotes()` returns a Collection that holds all the PIMNote items currently in the PIMCollection. If there are no PIMNote items in the PIMCollection, this method should return an empty Collection. Note that it is not specified exactly what implementation of a Collection is returned (that's up to you), just that whatever is returned implements the Collection interface.

getTodos

```
public Collection getTodos();
```

`getNotes()` returns a Collection that holds all the PIMTodo items currently in the PIMCollection. If there are no PIMTodo items in the PIMCollection, this method should return an empty Collection. Note that it is not specified exactly what implementation of a Collection is returned (that's up to you), just that whatever is returned implements the Collection interface.

getAppointments

```
public Collection getAppointments();
```

`getAppointments()` returns a Collection that holds all the PIMAppointment items currently in the PIMCollection. If there are no PIMAppointment items in the PIMCollection, this method should return an empty Collection. Note that it is not specified exactly what implementation of a Collection is returned (that's up to you), just that whatever is returned implements the Collection interface.

getContacts

```
public Collection getContact();
```

`getContacts()` returns a Collection that holds all the PIMContact items currently in the PIMCollection. If there are no PIMContact items in the PIMCollection, this method should return an empty Collection. Note that it is not specified exactly what implementation of a Collection is returned (that's up to you), just that whatever is returned implements the Collection interface.

getItemsForDate

```
public Collection getItemsForDate(Date d);
```

`getItemsForDate` returns a Collection that holds all the PIMEntities in the PIMCollection that have a date that matches the date d. If there are no items that match the date, this method should return an empty Collection. Note that it is not specified exactly what implementation of a Collection is returned (that's up to you), just that whatever is returned implements the Collection interface.

Note: It is expected that all PIMEntities in the returned collection implement the interface you created during array version. (There should only be PIMTodo and PIMAppointment items, since notes and contacts are not associated with dates).

Testing PIMCollection

Test your `PIMCollection` program using your other classes (everything from previous array version). Although your PIMCollection class should not depend on any of your other classes (except for possibly the name of the "dateable" interface you created), we will use your classes when testing a PIMCollection, so make sure you include them in your submission. Feel free to make any changes you want to your previous array version classes for this collection version.

You do not need to provide us with a main() that tests your IMCollection class, but feel free to do so if you want. We can write our own main that creates a PIMCollection object and puts some PIMEntities into it to test it. (Although since we don't have standard interfaces to PIMNote, PIMTodo, etc we won't be putting anythings except priorities in the PIMNotes, PIMTodos, PIMAppointments, etc).

IMPORTANT NOTE

Your PIMCollection class must be a Collection! All methods defined of any Collection object must be supported. Although this sounds difficult, all you need to do is to extend an existing Collection class (like an ArrayList or HashSet). Use whatever makes sense to you as your base class, but it isn't necessary (or wise) to implement the entire Collection interface from scratch (just inherit it!).