Final Project Report

Name: Sara Hyo Park
Title: Birthday Tracker

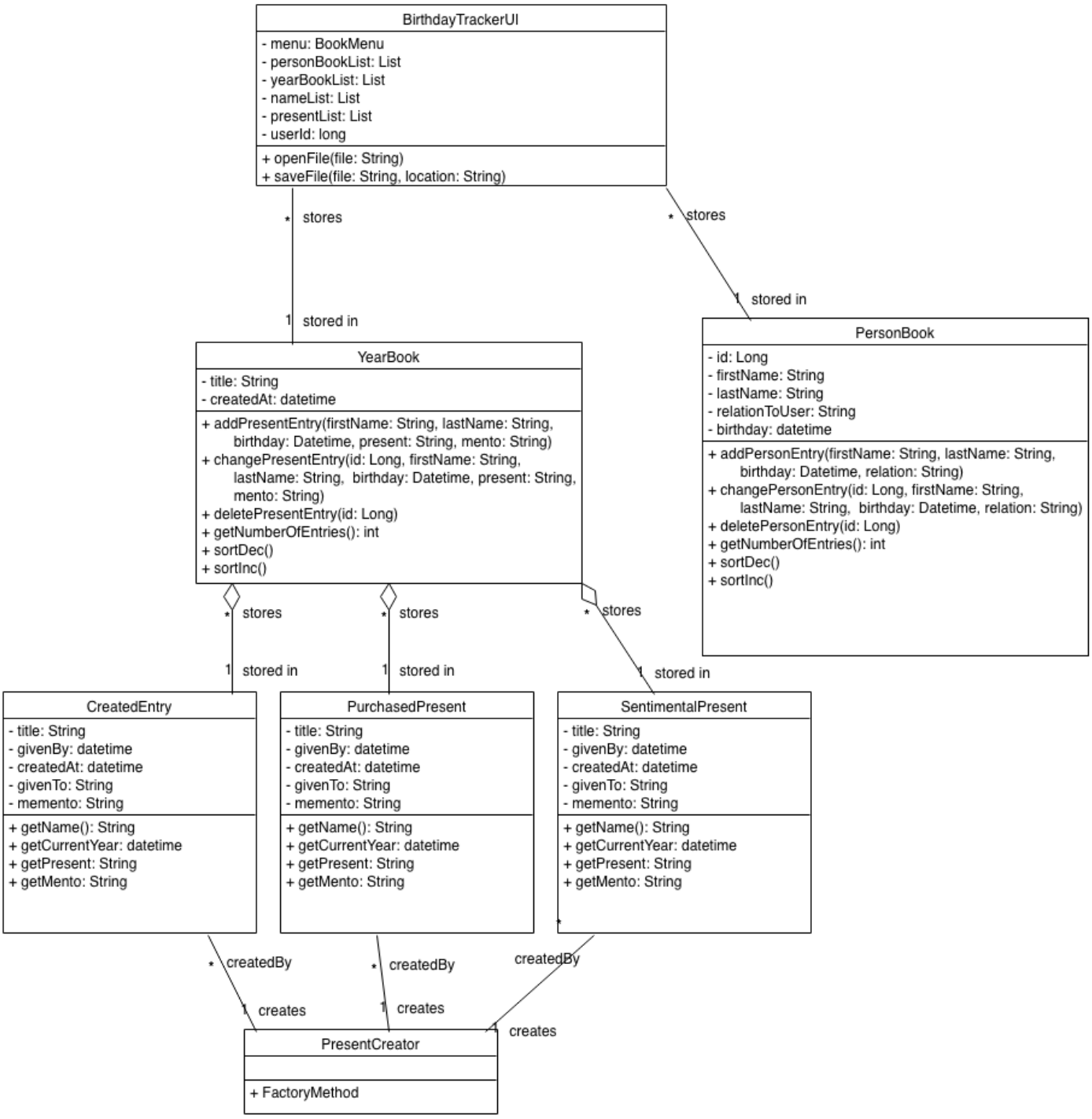
Summary: Birthday Tracker will be an application that always users to store the birthdays of their friends and presents that they have given them. The application will have year books in which past presents or future presents can be stored. The application will also have person books in which the user can store friends and family with their birthdays and relations.

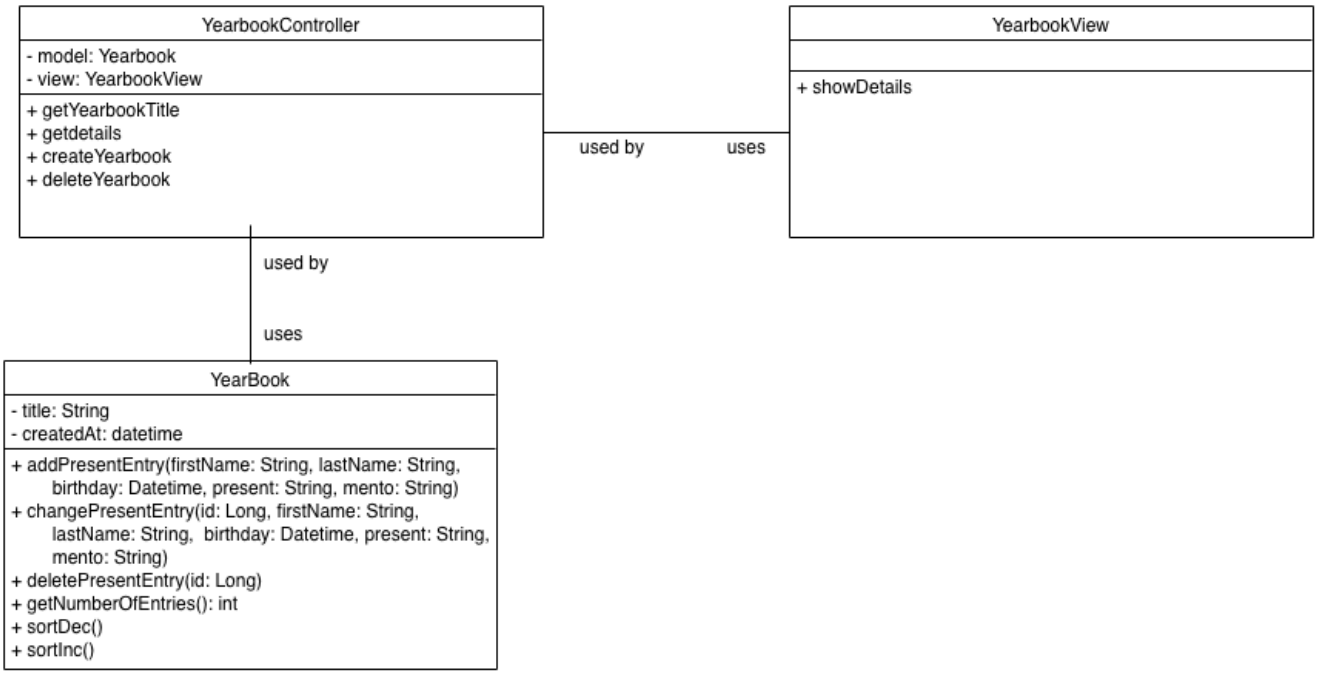
Implemented Features:

1	User can select to open Year Books, Person Book, and Admin site
2	User can add and delete a person's birthday information in Year Books or Person books
3	User can edit birthday information: name, date of birth
4	User can add past presents information in Year Books
5	User can add people to person books with birthday and relation
6	User can add presents to a certain year book, whether past, current, or future
7	User delete year books or person books
8	User can login as admin and change any model

Unimplemented Features

9	User can see user name at top of application
10	User can change user info
11	User can search for a present, person book, or yearbook


BirthdayTrackerUI
- menu: BookMenu
- personBookList: List
- yearBookList: List
- nameList: List
- presentList: List
- userId: long
+ openFile(file: String)
+ saveFile(file: String, location: String)
* stores
1 stored in
* stores
1 stored in
YearBook
- title: String
- createdAt: datetime
+ addPresentEntry(firstName: String, lastName: String, birthday: Datetime, present: String, mento: String)
+ changePresentEntry(id: Long, firstName: String, lastName: String, birthday: Datetime, present: String, mento: String)
+ deletePresentEntry(id: Long)
+ getNumberOfEntries(): int
+ sortDec()
+ sortInc()
PersonBook
- id: Long
- firstName: String
- lastName: String
- relationToUser: String
- birthday: datetime
+ addPersonEntry(firstName: String, lastName: String, birthday: Datetime, relation: String)
+ changePersonEntry(id: Long, firstName: String, lastName: String, birthday: Datetime, relation: String)
+ deletePersonEntry(id: Long)
+ getNumberOfEntries(): int
+ sortDec()
+ sortInc()
* stores
1 stored in
* stores
1 stored in
* stores
1 stored in
CreatedEntry
- title: String
- givenBy: datetime
- createdAt: datetime
- givenTo: String
- memento: String
+ getName(): String
+ getCurrentYear: datetime
+ getPresent: String
+ getMento: String
PurchasedPresent
- title: String
- givenBy: datetime
- createdAt: datetime
- givenTo: String
- memento: String
+ getName(): String
+ getCurrentYear: datetime
+ getPresent: String
+ getMento: String
SentimentalPresent
- title: String
- givenBy: datetime
- createdAt: datetime
- givenTo: String
- memento: String
+ getName(): String
+ getCurrentYear: datetime
+ getPresent: String
+ getMento: String
* createdBy
1 creates
* createdBy
1 creates
createdBy
1 creates
PresentCreator
+ FactoryMethod



Class Diagrams:

Items that were changed:

1. Model-View-Controller implementation was created for Yearbooks, Personbooks, and Presents
2. Present Factory was added to create different types of presents.

There were not many things changed because of the fact that a design was created early on prior to coding.

Design Patterns:

The design pattern I decided to implement was the Factory Design Pattern. Present Factory was added to create different types of presents: Sentimental, Purchased, and Created. This was added to allow the user to create different present objects types that are color-coded. It also allows for the user to view just one type of present at a time.

I implemented Present Factory by using Django's model.Manager, which allows for creating objects at initiation. In the manager, I used a method "makePresent" which created different presents for each type of present that was created by the user. I believed that the Factory design pattern would be the most useful in creating different objects depending on the type of the object because it is used to replace class constructors and could create the objects at run-time.

What I learned:

Designing object-oriented models is absolutely crucial to the success and organization of a project. Having a model to constantly look back on was incredibly useful. Another thing that I learned is to be appreciative of the ability of Django as a framework to provide MVC implementation as well as many design patterns.