CSCI 4448 HW 4

Josh Wepman and Kyle Poole

Fall 2011

The following are the design documents for a play-list management system called PlayMaster. This document includes three use cases, two activity diagrams, two sequence diagrams and an object diagram for the system.

**Use Case: Instructor provides a list of practice items for a student in a system**

|  |  |
| --- | --- |
| **Main Path**   1. Instructor inputs student name 2. Student exists in system (has been entered) 3. Instructor inputs song list and date for song list to be played (new date) 4. System adds songs to date list | **Alternate Paths**   * 1. Student doesn’t exist      1. Instructor re-executes the program to input student information      2. Instructor re-executes the program to input song list information   2. Date already exists in storage      1. Systems opens date to append to list |

**Activity Diagram**

Instructor enters new student info

[Student does not exist in system]

Instructor inputs name

[Student exists in system]

User inputs date

Program validates and stores song

User enters song

**Use Case: Student provides list of practice items for a given day**

|  |  |
| --- | --- |
| **Main Path**   1. Student enters name 2. Student enters new date for practice items 3. Student enters list of practice items for given date 4. System validates and stores information | **Alternate Paths**   * 1. Student Name Does Not Exist      1. User enters student information (new student) –or- re-enters name correctly   2.1 Date already exists in storage  2.1.1 System appends data to list of items for given date   * 1. Songs don't exist in storage (new song)      1. User enters additional song information and info is stored |

**Activity Diagram**

Student enters name

[append or create new date]

User enters additional song info

Program validates and stores song item

[existing song]

[new song]

User enters song item

User enters date for practice items

User enters new student information

[new user]

[user mistyped name]

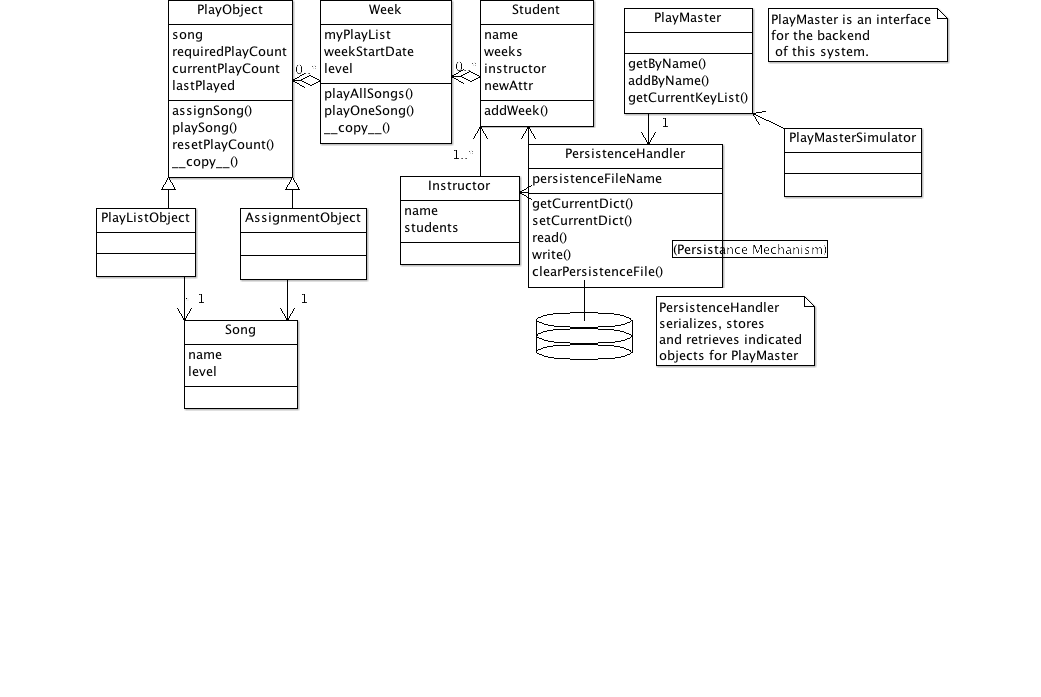
[invalid student]

[valid student]

**Use Case: Instructor requests monthly report for given student**

|  |  |
| --- | --- |
| **Main Path**   1. Instructor inputs name and requests report for given student and given month 2. System locates and builds report for existent student for given month | **Alternate Paths**   * 1. Student does not exist in storage      1. System returns error   2. Data doesn’t exist for given month/student      1. System returns error |

**Class Diagram**

****

Description of classes:

* **PlayMasterSimulator** - This object handles our textual user interface, interacting with the PlayMaster façade to handle all transactions between a user and the backend.
* **PlayMaster** – Provides a façade for the persistence and object backend of the system.
* **PersistenceHandler** – Interacts with DB backend – in this case, we store a JSON encoded dictionary of lists of serialized python objects (whew!), converts objects to serialized entities in DB and vice versa.
* **Instructor** – Handles instructor responsibilities, mapped with 1+ students to 1 instructor.
* **Student** – Handles student responsibilities, aggregates week list.
* **Week** – Aggregates playlist and added songs by instructor, keeps track of progress
* **PlayObject <- PlayListObject and AssignmentObject** –Handle aggregation of songs as appropriate for different expectations for number of plays. PlayObject is the parent object of both, which slightly differ in their implementation/requirements, but the polymorphism here is important…
* **Song –** Handle generic song responsibilities and store attributes such as name, creator, level, etc.

**A note on persistence**

I elected to go with a simple persistence route and store serialized python objects encoded with the pickle module (which ships with python natively, I believe). This approach is not secure (persistence file can be edited/altered to do bad things), but nice and convenient for ease of OOP design. The PersistenceHandler class implicitly capitalized on python-native polymorphism (every class definition implicitly subclasses the ‘object’ superclass to be able to serialize/deserialize nicely…

**Sequence Diagrams (for Use Cases 1 and 2)**

These are scanned and hand-drawn, for simplicity…

{to do}