The Library Management System for ACM Class 2010

09 ACM

Xiao Jia

Dec. 3rd, 2010

Outline

- Overview of the Architecture
- The Library Interface
- Data Access Objects
- Implementation Hints
- Advanced Features
- Testing and Grading

Overview of the Architecture

- Understanding Requirements
- The Whole Infrastructure
- View of Participant Classes (VOPC)

Understanding Requirements

- A User’s Perspective
  - Reader
    - Student
    - Teacher
  - Administrator
- A Resource’s Perspective
  - Book Kind (A Kind of Book)
  - Book

A User’s Perspective

- As a reader (a student or a teacher)
  - Change password (derived from User)
  - Borrow
  - Return
  - Renew
  - Reserve
  - List all borrowed books
  - List all reserved books
  - Get my penalty

A User’s Perspective

- **3 differences** between students and teachers
  - The reader type (STUDENT or TEACHER)
    - This influences how you store the reader’s information
  - The number of books that can be borrowed
  - Only teachers can reserve books

A User’s Perspective

- As an administrator
  - Change password (derived from User)
  - Create/update/remove a reader
  - Create/update/remove an administrator
  - Create/update/remove a kind of book
  - Create/update/remove a book
  - List all readers
  - List all administrators

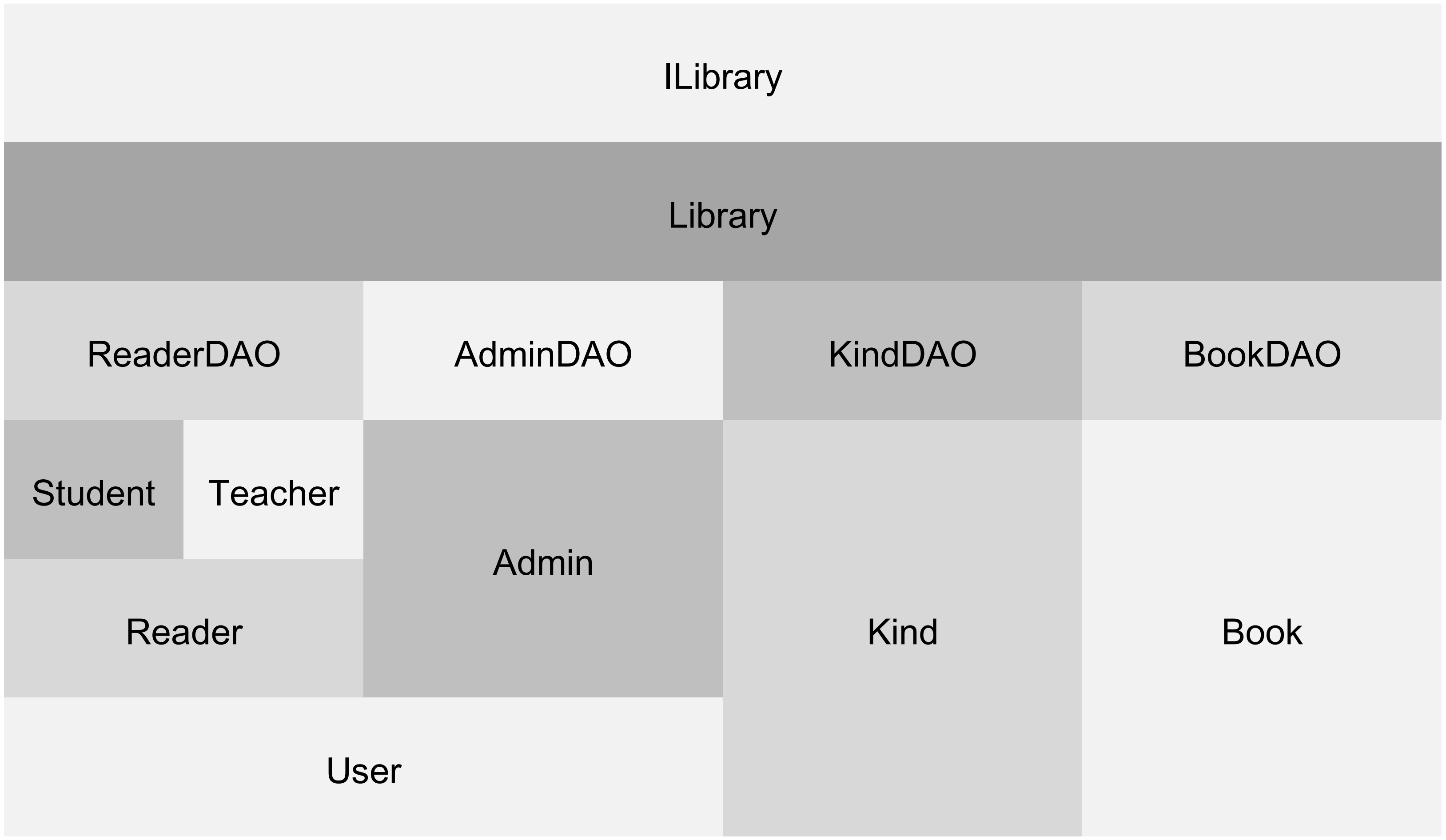
A Resource’s Perspective

- Book Kind means “a kind of book”
- A Kind has the following attributes
  - ISBN
  - Name
  - Authors
  - Index
- A book kind is an **aggregation** of many books

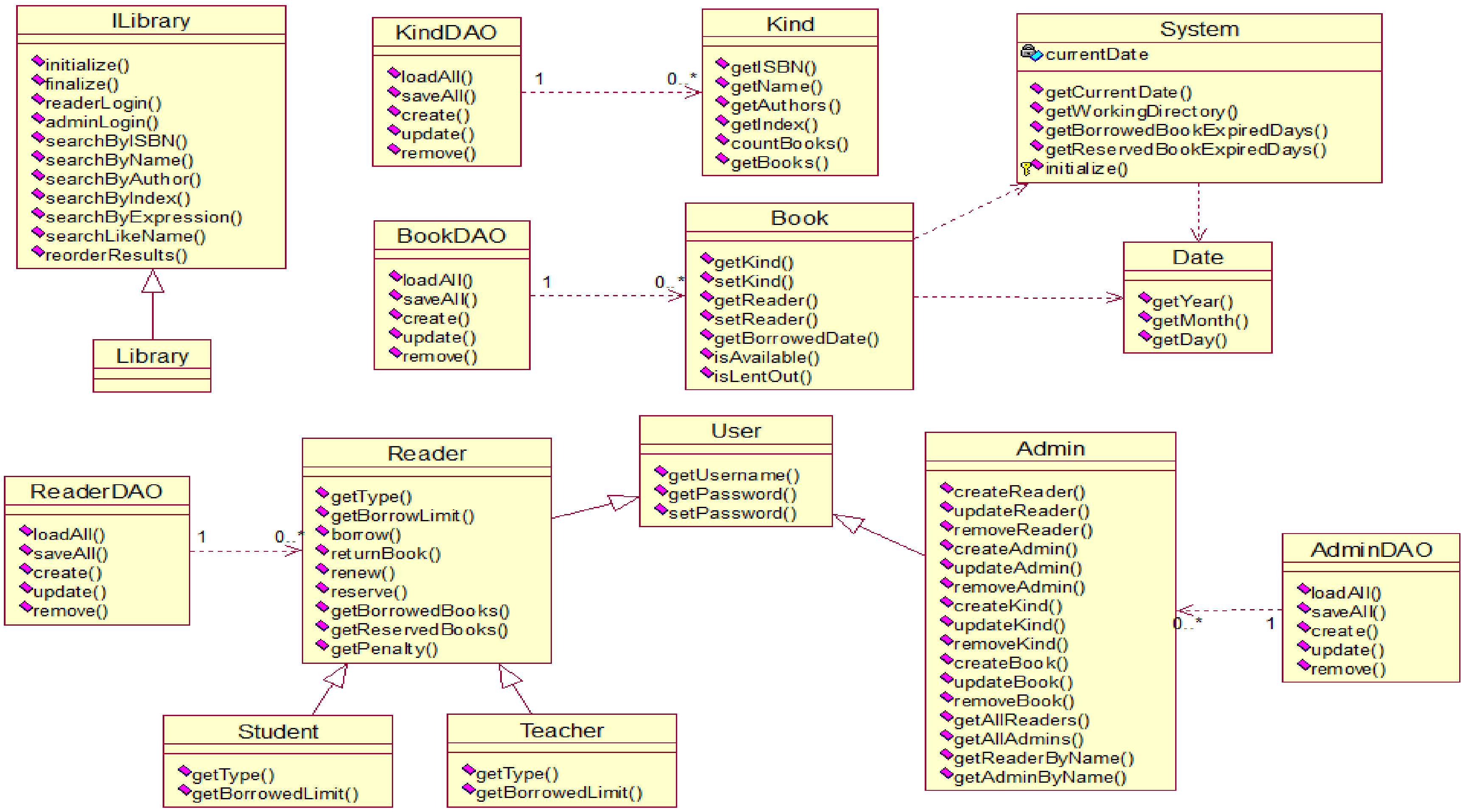
A Resource's Perspective

- A book is an instance of a book kind.
- A book kind consists of many instances which share the same attributes
  - Each of them is a book
- We should be able to check whether a book is available to borrow
- We should be able to know when a book was borrowed

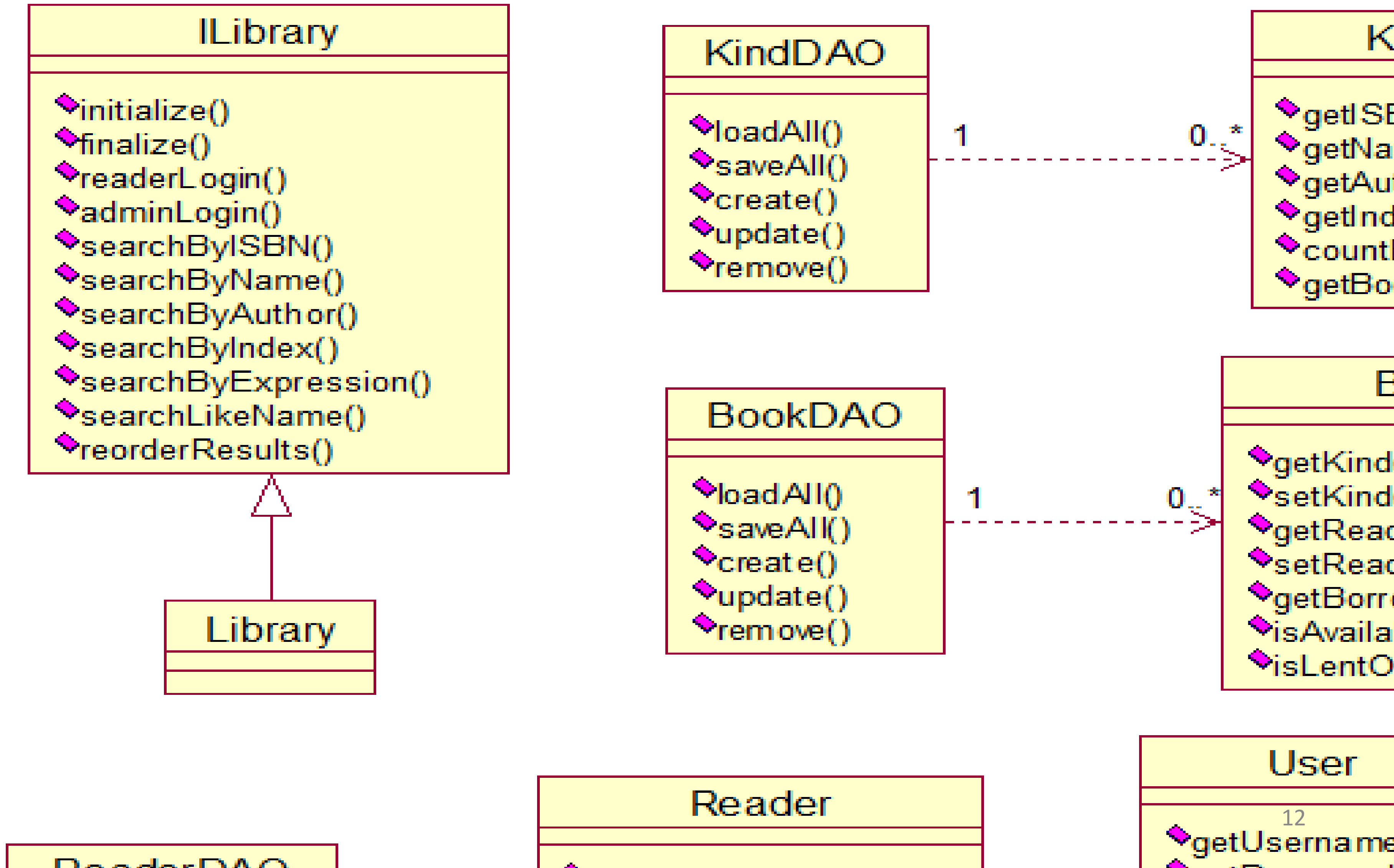
The Whole Infrastructure



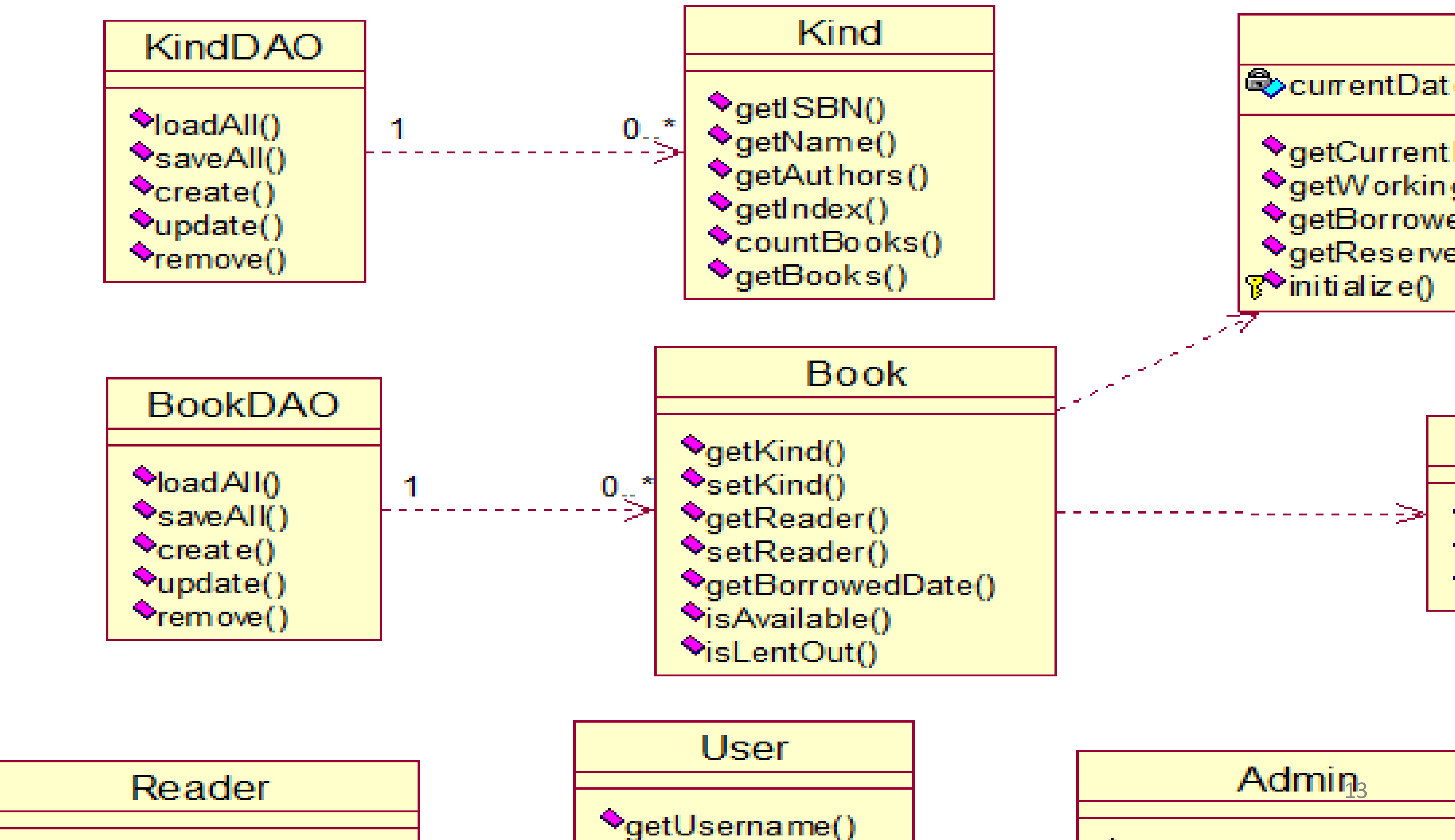
View of Participant Classes



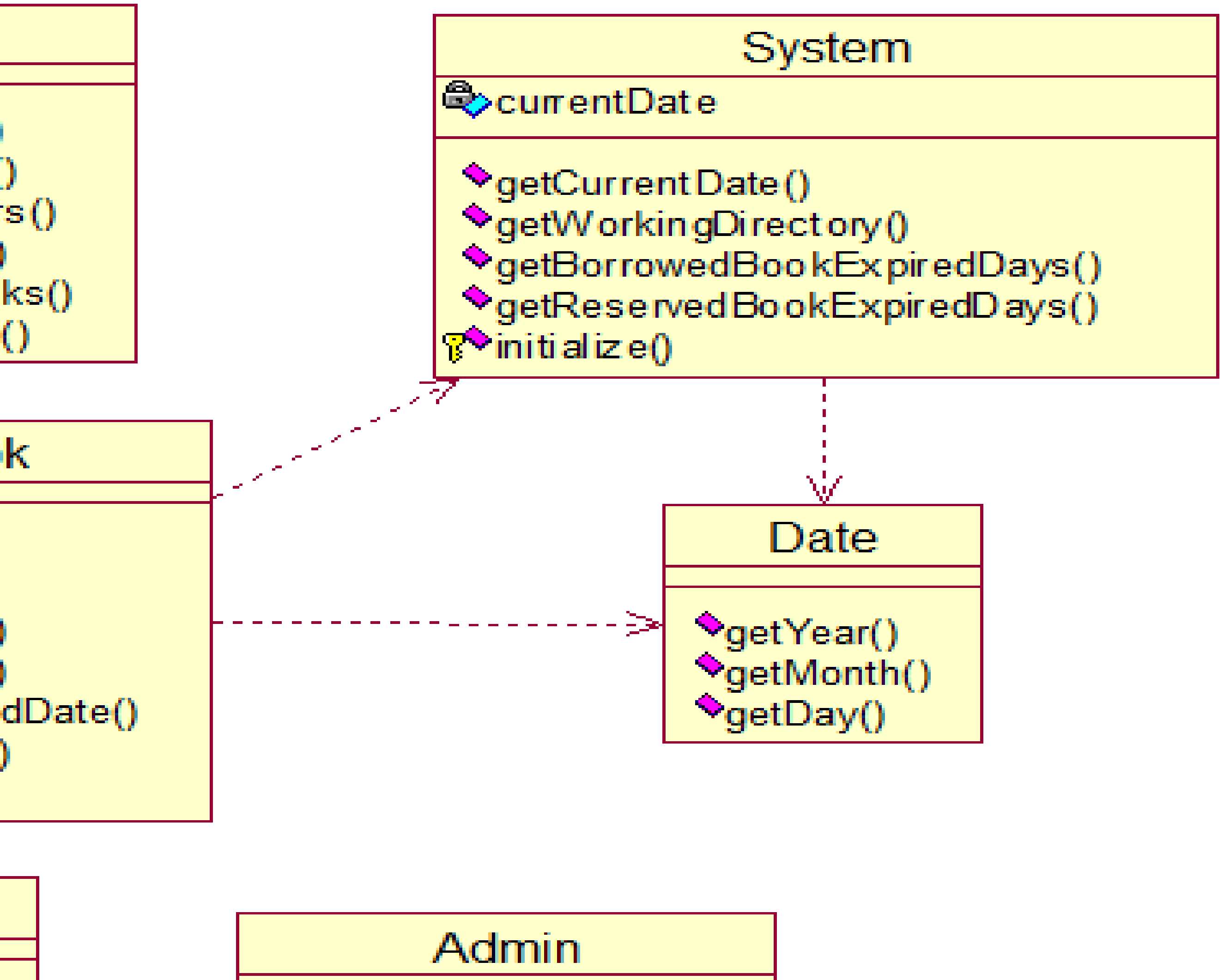
View of Participant Classes



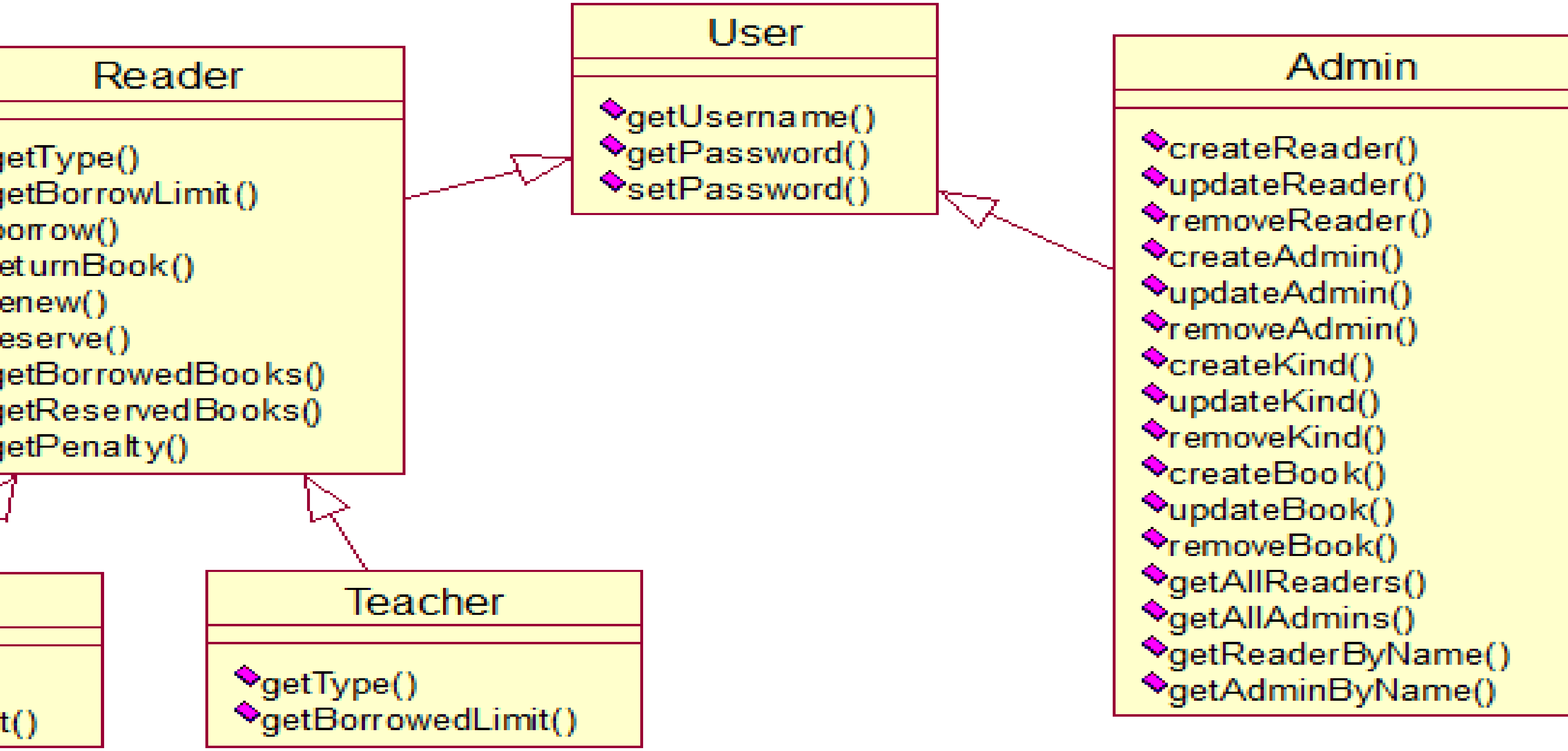
View of Participant Classes



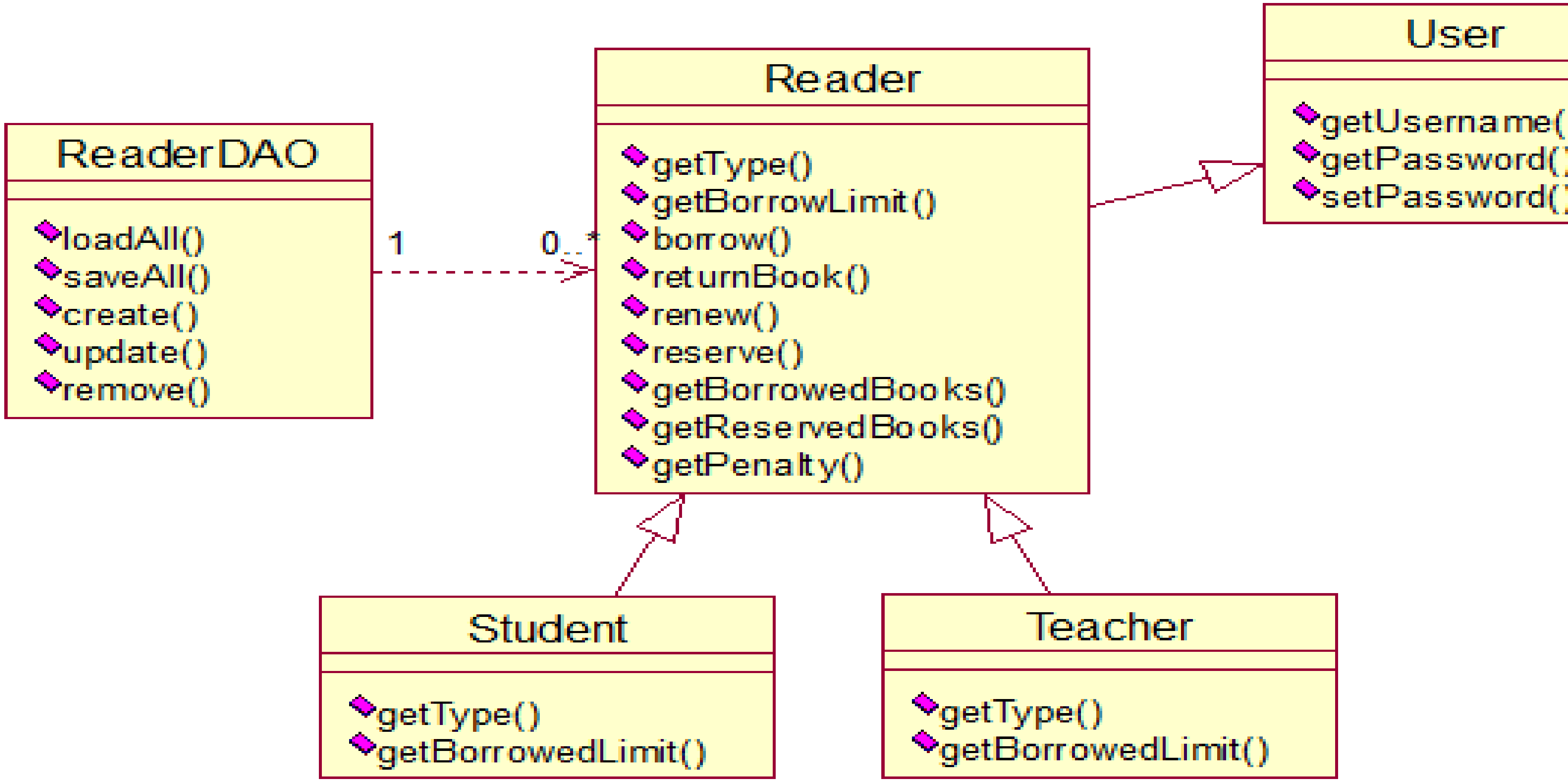
View of Participant Classes



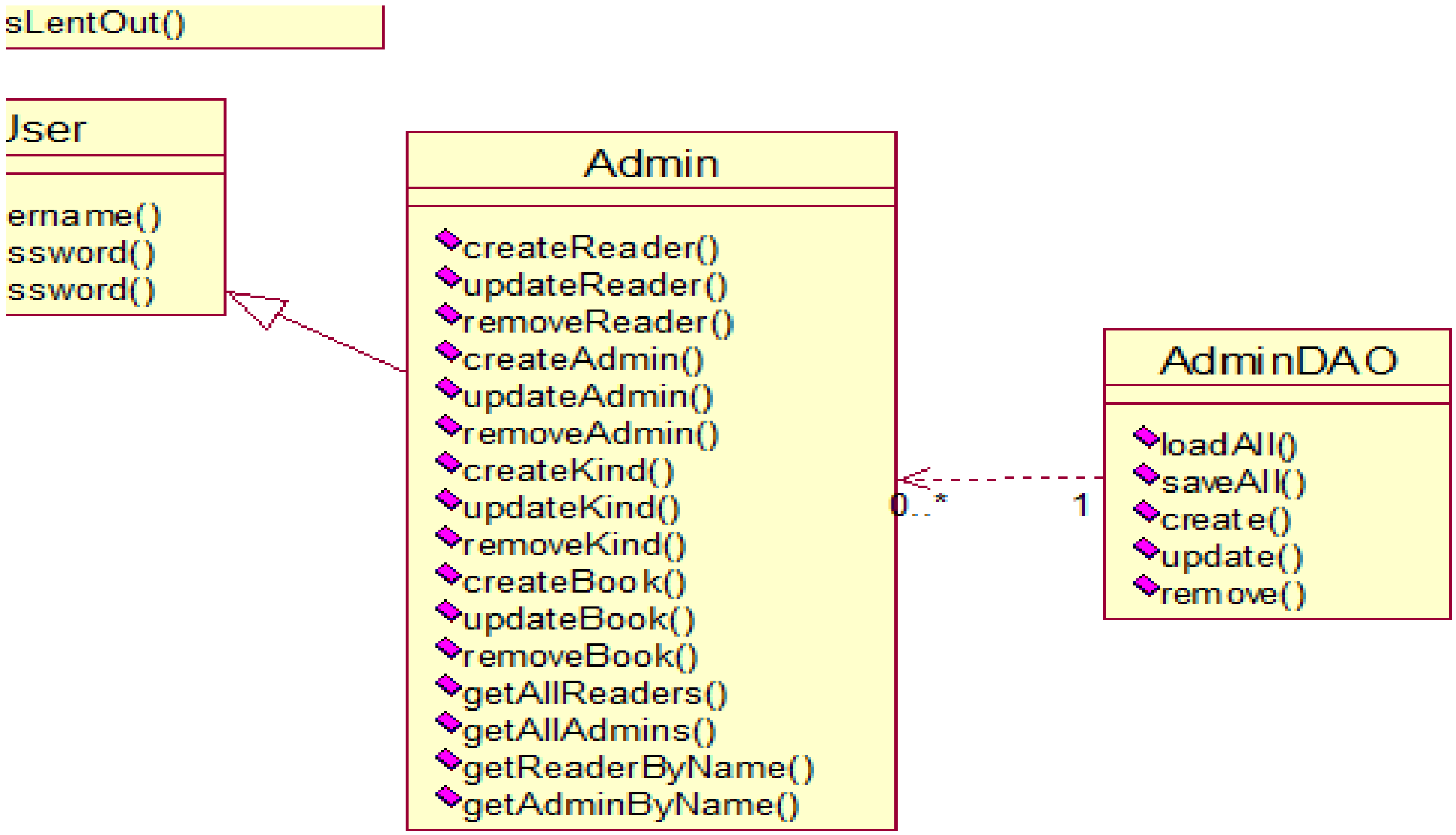
View of Participant Classes



View of Participant Classes



View of Participant Classes



The Library Interface

- The library interface exposes the functionalities that everybody can access
  - Log in as some role (reader or administrator)
  - Search books (by various fields or methods)
  - Advanced features such as reordering results
- ILibrary is pure virtual as an interface
- Library extends ILibrary as a subclass
  - You have to implement the class named Library

What is a login? And why?

- “login” is a noun and “log in” is a verb phrase
- A login is an access to your library system
- An external participant get a login using a matching pair of username and password
- A login identifies who is accessing the system
- Different logins have different access rights
  - A reader can only borrow, return, …
  - An administrator can only create, update, …

Data Access Objects

- Motivation
  - Each model needs the functionality of …
    - Creation
    - Updating
    - Removal
  - Access the underlying data directly is not clean
    - The system is built as a layered application
    - It violates the basic rule (or principle) of layering that business routines have to break through several layers to get the data needed

Data Access Objects

- The pattern of *Data Access Object* (DAO) helps with manipulating data
  - Provide a unified interface for object creation, updating, removal, etc.
- In this library management system, DAOs also hold a universal copy of corresponding data objects
  - DAO::loadAll – called by ILibrary::initialize
  - DAO::saveAll – called by ILibrary::finalize

Data Access Objects

- A DAO class should keep a list of pointers to dynamically allocated objects
  - The objects should be read from the hard disk at the beginning of system execution (initialization)
  - The objects should be dynamically created (i.e. the memory is dynamically allocated using `new`)
  - The objects should be written to the hard disk at the end of system execution (finalization)

Implementation Hints

- References vs. Pointers
- std::list
  - *http://www.cppreference.com/wiki/container/list/start*
- Wide-character set
  - wchar_t
  - std::wstring
  - Conversion between std::string and std::wstring
- File Operations
  - Check file existence
  - Read from a file
  - Write to a file

References vs. Pointers

- A pointer can point to many different objects
- A reference can refer to only one object
- A pointer may be NULL
  - Means that there may be a …
- A reference is always valid
  - Means that there is a …
- **Use references whenever you can**
- Use pointers in STL containers since references are **not assignable**

Conversion between string and wstring

- You should find materials about character sets and encodings by yourself
- You should not use the codes directly without understanding them first
- The following codes are supposed to be compiled under Windows (Win32 Platform)
  - #include <windows.h>

```
inline string wtos(const wstring &w)
{
  int len = WideCharToMultiByte(
    GetACP(), 0, w.c_str(), -1,
    NULL, 0, NULL, NULL
  );
  char *buf = new char[len];
  WideCharToMultiByte(
    GetACP(), 0, w.c_str(), -1,
    buf, len, NULL, NULL
  );
  string s(buf);
  delete[] buf;
  return s;
}
```

```
inline wstring stow(const string &s)
{
  int len = MultiByteToWideChar(
    GetACP(), 0, s.c_str(), -1,
    NULL, 0
  );
  wchar_t *buf = new wchar_t[len];
  MultiByteToWideChar(
    GetACP(), 0, s.c_str(), -1,
    buf, len
  );
  wstring w(buf);
  delete[] buf;
  return w;
}
```

Check file existence

```
bool file_exists(string const &path)
{
  fstream f(path.c_str());
  bool exists = f.is_open();
  f.close();
  return exists;
}
```

Advanced Features

- reorderResults
- searchLikeName
  - Be sure to use wide-character set functionalities
- searchByExpression
  - Simple cases
  - Powerful cases
    - Theory: set operations (union, intersection, difference)
    - Practice: (1) tokenizing (2) expression evaluation

Testing and Grading

- All programs will be tested automatically
  - *http://code.google.com/p/mycpptest/*
  - You don't have to write a function named "main"
    - main() will use some test suites to test your program
    - A test suite consists of many test cases
  - You should not create your own classes
  - You should implement all the methods specified
    - You should ensure that your code can compile
    - Your implementation can be wrong but you have to write it
  - You are allowed to add methods to existing classes

Testing and Grading

- Grading policy
  - Basic functionalities (65%)
  - Documentation (15%)
  - Coding style (20%)
  - Bonus for advanced features (5% for each)
  - Final score will not exceed 100%
- Basic functionalities and advanced features are tested automatically
  - Your score is strictly related to the percentage of test cases that your program passes

Always Challenge Miracles

THANK YOU FOR LISTENING