Data Modeling and Test-Driven Development

Individual/Pair Project
CS205 Fall 2020
15% of course grade
Part I: due Sunday, Sept. 27th, 11:59 pm
Part II: due Sunday, Nov. 8th, 11:59 pm

1 Description

You will work individually or with a partner to design and implement an object-oriented model of data of your choice. You will then implement a set of basic operations on the model, as class methods, and write unit tests for those operations.

1.1 Description of data

Think of some kind of data that you want to model. There must be two different kinds of entities in your model, and there must be some kind of one-to-many or many-to-many relationship between the entities. You must also have a container to hold instances of one of the entities.

For example, I chose to model a library. The entities are books, patrons, and checkouts; and the relationship between them is that a checkout contains a book and a patron–this embodies a many-to-many relationship between books and patrons. And my library object holds both books and patrons.

```
Book
 title: string
 author: string
 catalog_number: string

Patron
 name: string
 card_number: integer
 checkouts: set<Checkout>

Checkout
 patron: Patron
 book: Book

Library
 books: set<Book>
 patron: set<Patron>
 checkouts: set<Checkout>
```

Other possibilities: courses, instructors; buildings, rooms, courses; restaurants, people, reservations; songs, artists; etc.

You must choose data that is different from the data that your warm-up project team is using.

1.2 Object-oriented design

You'll create the class design to represent your data. Some thoughts:

- **Entities** are the things you're modeling; these become your classes.
- **Attributes** describe the current state of the object; these become member variables.
- **Behavior** describes the things an object can do; these become member functions.

In my library example, the boos and patrons are entities. Title, author, and catalog number are attributes of a book object. And "checkout book" or "return book" are behaviors that a patron can do.

A class is a like a blueprint for the definition of an entity. The definition says what the attributes and behaviors of the object are.

Two key concepts:

- **Abstraction**: focus on what's important, and ignore what's not important.
- **Encapsulation**: hide the details.

Think of an object called "Lamp". It has an attribute isOn, which we could model with a boolean. We don't want the outside world to be able to change this attribute directly—only through methods `turnOn()` and `turnOff()`.

Or in my system, I should not be able to be able to modify the set of checkouts that the library has—this can only happen through the explicit actions "checkout book" and "return book"

For the simple set of data (classes) you'll use in this project, it shouldn't be necessary to use inheritance.

A general recommendation for design object-oriented systems: hide as much as possible.

1.3 Operations

Define a set of operations on the entities in your model. For my library system, this includes the following:

```
Book
 getters, initializer

Patron
 getters, initializer
 do_checkout(Book b)
 list<Checkout> get_checkouts()

Checkout
 getters, initializer

Library
 add_patron(Patron p)
 list<Patron> get_patrons()
 Patron find_patron(string card_number)
 list<Patron> get_patrons()
 add_book(Book b)
 list<Book> find_book(string title)
 do_checkout(Patron p, Book b)
 is_checked_out(Book b)
 list<Checkout> get_checkouts(Patron p)
 return_book(Patron p, Book b)
```

You should define a rich set of operations that will at least query all the attributes of the entities in your model.

Note: this is not a database. You don't have to create a database for this project. You merely need to provide interfaces to create, manage, and delete the objects that you are modeling.

2 Test-driven Development

Create a set of unit tests to test your model objects and the operations on your objects. Use PyUnit or JUnit.

3 Choice of Language

You can use Python or Java. I have implemented my Book/Patron/Checkout/Library model, with unit tests, in Python and Java. You can see my implementation in the class gitlab site: `https://gitlab.uvm.edu/Jason.Hibbeler/forstudents_f20/-/tree/master/CS205/TDD-examples`.

4 Deliverables

4.1 Deliverable I

By Sunday, Sept. 27th, 11:59 pm: description of your data and your initial description of your classes and class methods. I'll read these and give you feedback and possibly ask you to make changes. If you are working with a partner, tell me the name of your partner.

4.2 Deliverable II

By Sunday, Nov. 8th, 11:59 pm: completed implementation and unit tests. Include also a short report showing a diagram of your classes (an informal sketch is OK). Describe the tests that you created–specifically, what behaviors you are testing and what constitutes success for the tests. Also, create a test that fails—because the implementation is incorrect—and then fix the implementation and show that the test passes; and describe the fix you made to the implementation.

5 Graduate students

Use a code-coverage tool as well (coverage, for Python, or JCov for Java). Grad students must work individually.

6 Evaluation

I'll look at how thoroughly you've modeled the entities in your data and how thorough your unit tests are.