Assignment 1

Creating a Design Document

Author: Jimmy Tran, Brian Crites

Introduction:

As programmers it is tempting for us to get our hands dirty with code immediately after receiving the specs. In any complex software development, there are many moving parts and pieces that can quickly turn out project into spaghetti code. It is imperative that before we open vim we take a step back, plan out the code we are going to write, and create a design to follow. One tool software engineers use for this is Unified Modeling Language (UML) which help visualize our project's architecture. In this assignment you will create a design document to map out your first coding assignment, Assignment 2. This will include a simple UML diagram to represent your components and their dependencies.

Note: All assignments must be done with exactly one partner.

Assignment

Your assignment is to write a design document that you will use for reference when developing your Assignment 2 submission. It's purpose is as an exercise to get you to start designing before you start coding. Like any good agile method, what you write in the design document is a starting point for your design, not a road map that you must rigidly follow (because that would be waterfall development, and this is not a government project). You should create a document with the following sections

- **Title Page:** including the title "Assignment 1: Design", the date, the quarter and year, and the authors names
- **Introduction**: Give a brief overview of your design
- **Diagram**: This will be the UML diagram for your system, and should show all classes you plan on developing. This diagram can take up multiple pages if necessary.
- **Classes/Class Groups**: descriptions of each class/class group that you plan on developing, this can be as simple as a description of what each class accomplishes and how, or a pseudo code level class definition. A class group would be a group of classes that all inherit from a single base class (composite classes are an example). For class group give a description of the base class, as well as the subtle differences between the child classes. Make sure to describe any key design choices, such as why you chose

certain containers for key data members, why a class needs to pointer member to another class, or how a key function will generally be written
- **Coding Strategy**: how you and your partner plan to break up the work, who will be in charge of which segments and how you will integrate the segments together
- **Roadblocks**: issues that you think will arise during development and what strategy you will use to overcome them

When you create your design document, do not think only about the current assignment. Think about how you would also extend the assignment to have new functionality (an important exercise in software construction, as new functionality is almost always necessary).

Please use some drawing program to create your UML diagram. UML diagrams that are drawn by hand and scanned/photographed and added to your design document will **not** be allowed. Programs such as GIMP, GraphViz, or even Google Docs should be capable of creating these diagrams.

Submission

Submit your design document as a single PDF to the submission link in iLearn. Only one person in your group needs to submit this document.

Grading

Your documents will be graded using the following breakdown

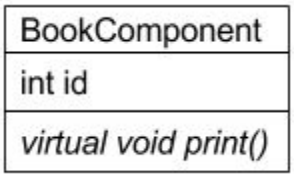
40 UML Diagram
30 Class Descriptions
20 All Other Sections
10 Structure
100 Total

An Introduction to UML

Below is an introduction to UML diagramming. There are lots of specification for UML diagramming, however these are the only specifics that you need to follow. If you find that this list is incomplete for your needs, please create an indicator that fits your needs and specify what it means using text either in the image or as a reference in the text.

In this extended example, we are given the task of designing a database for our local public library. Although there are source code provided for each example, you should know that this is for further clarity and is not intended to suggest immediate implementation.

One Class, One Box



In UML, every class is depicted by an enclosing box separated into three sections: the class name, fields, and methods. This class name is `BookComponent`. It has a single member variable, `int id`, and a single method `virtual void print()`. Constructors and destructors are generally excluded in UML classes.

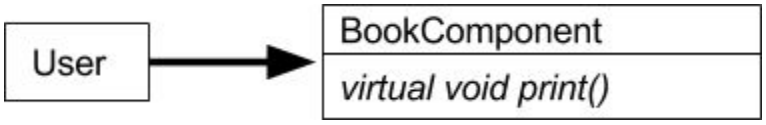
Following the composite pattern we've learned in class, we would like `BookComponent` to be the composite base class that all our other composite classes will inherit from and extend. We also want every inheriting class to implement its own version of `virtual void print()`, so it would be wise to make this class an abstract base class so we can use object-oriented polymorphism. The `BookComponent` class can be written as so:

BookComponent.h

```
class BookComponent(){
      protected:
            int id; //identification of type of composite/component
      public:
            BookComponent();
            ~BookComponent();
            virtual void print() = 0; //pure virtual function
};
```

NOTE: In future diagrams we will exclude this class's field for simplicity.

Solid Arrow (Association)



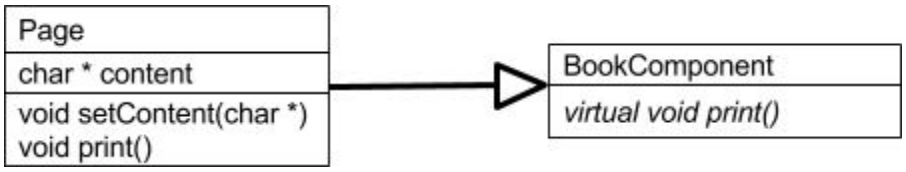
The solid arrow in this relationship represents a directional association between two components or classes. The direction of the arrow indicates that the User "knows" about the `BookComponent`. In computer science, "knowing" means that the User class has an instance variable or receives `BookComponent` class as an argument to its method. For example, the code for this relationship may be written as:

User.h

```
#include "BookComponent.h"

class User {
	private:
		BookComponent* b; //pointer instance variable
	public:
		/* Constructors and Destructors Omitted */
		void read(BookComponent* artemis_fowl); //read a pointer
};
```

Open Arrow (Inheritance)



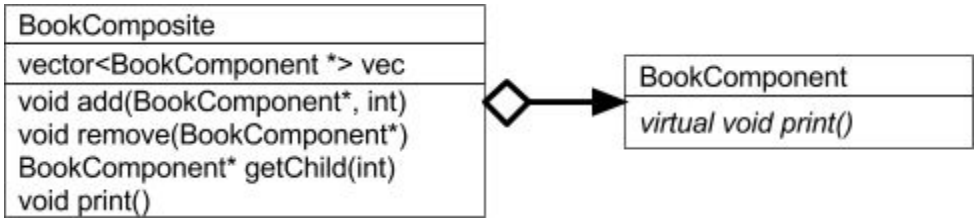
The class that the white arrow points to indicates the base class and the end is the derived class. The Page class inherit from the abstract base class `BookComponent`, and because of this the `print()` function must be implemented in the derived class.

Page.h

```
#include "BookComponent.h"
#include <iostream>

class Page : public BookComponent {
	private:
		char * content;
	public:
		/* Constructors and Destructors Omitted */
		void setContent(char * content) { this.content = content;
}
		void print() { std::cout << content << std::endl; }
};
```

Diamond to Arrow (Composition)



The diamond represents one class that holds a collection of instances of that other class that the arrow is pointed to. The `BookComposite` class store instances of `BookComponents` as vector of `BookComponent*`, as shown below.

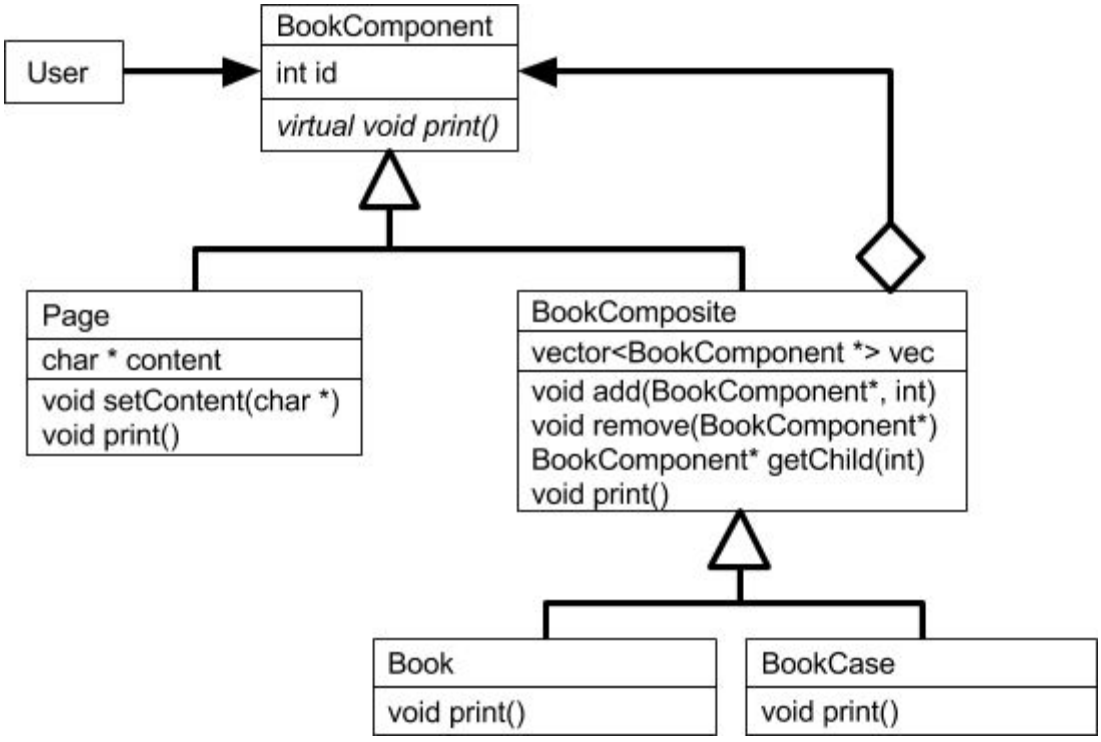
BookComposite.h

```
#include "BookComponent.h"

class BookComposite : public BookComponent {
      protected:
            vector<BookComponent*> vec;
            void add(BookComponent*, int); // Protected visibility to
defer type-checking to derived classes
      public
            /* Constructors and Destructors Omitted */
            void remove(BookComponent*);
            BookComponent* getChild(int);
            void print() { // Overriding base class method
                  for(unsigned i = 0; i < vec.size(); i++) {
                        vec.at(i)->print();
                  }
            }
};
```

Structural Diagram

Putting everything together from what we've learned from the examples above, we can design a blueprint for our library application.



Now you know the very basic structure of UML, you can begin to design the architecture of your program. Think in terms of objects and how inheritance allow you create a hierarchy of similar objects.

For more information on UML diagramming, you can refer to [this page on UML class diagrams.]