# Card-Carrying C++ Connoisseur

## Due Tuesday, February 16 at 8 a.m.

CSE 1325 - Spring 2021 - Homework #4 - 1

## Assignment Background

Flash cards are a traditional way to memorize definitions, such as for our vocabulary words. These usually had the question on one side, shown to the student, and the answer on the other, visible only to the TA. Since we have an exam coming up soon, let's automate some and save our TAs' time!

## Full Credit

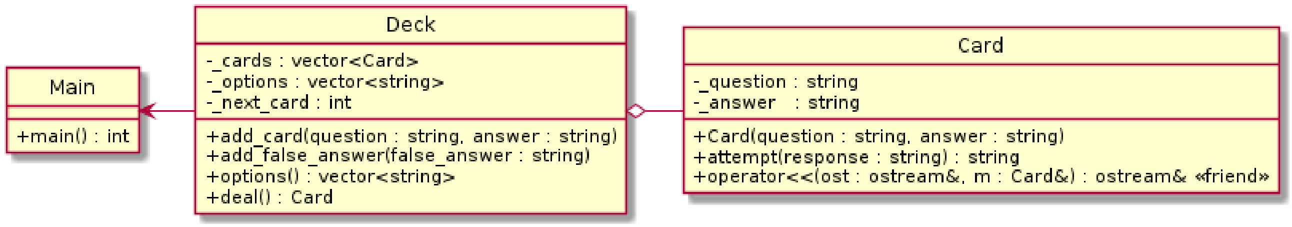
In your git-managed Ubuntu Linux 20.04 directory **cse1325/P04/full_credit**, create the TWO classes shown in the class diagram below.

The Card class represents an actual flash card, with a question and an answer (the two attributes). When streamed out, it prints the question. The attempt method accepts a response parameter, and returns a string indicating a correct or incorrect answer.

The Deck class composites Cards specified by the add_card method, also building an answer list along with some false "answers" (not on any card) added via the add_false_answer method. All of the correct and false answers are returned as a vector of strings by the options() getter (to print a menu of possible answers), and the deal() method returns the next Card object in the Deck.

The main program creates a Deck of flash Cards using at least 10 words and 4 false options from our Exam #1 vocabulary list. It then prints the question on each card along with the list of possible answers, accepts on answer, and prints whether that was the correct answer or not.

**Add, commit, and push all files** to your *private* cse1325 GitHub repository **often**. Much more detail is provided in Hints below.

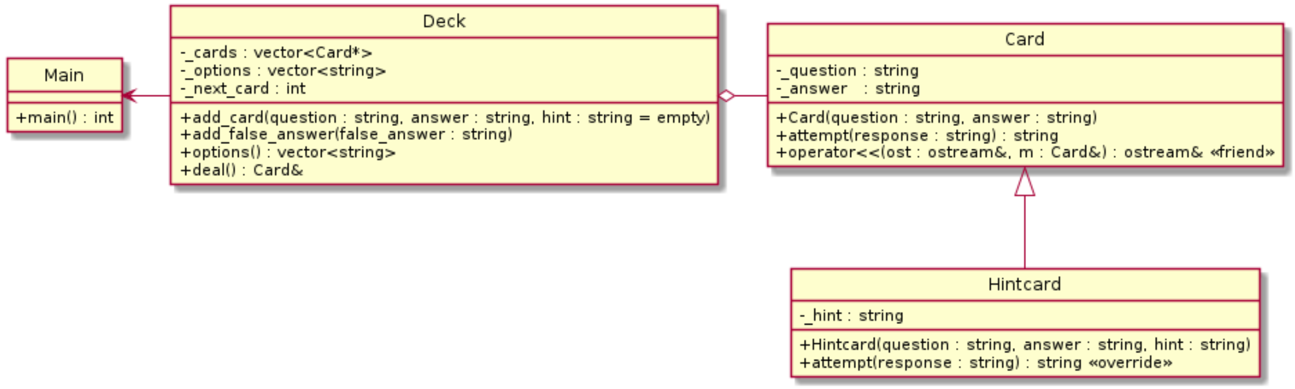

Main
+main() : int
Deck
-_cards : vector<Card>
-_options : vector<string>
-_next_card : int
+add_card(question : string, answer : string)
+add_false_answer(false_answer : string)
+options() : vector<string>
+deal() : Card
Card
-_question : string
-_answer : string
+Card(question : string, answer : string)
+attempt(response : string) : string
+operator<<(ost : ostream&, m : Card&) : ostream& «friend»

## Bonus

Derive (inheritance!) a new type of flash card called Hintcard from class Card. Hintcard takes a third parameter - std::string hint - which is returned by its attempt method (which overrides class Card's attempt) on a wrong answer instead of returning the correct answer.

The new class diagram is shown below.

**Add, commit, and push all files** to your *private* cse1325 GitHub repository **often**. Much more detail is provided in Hints below.

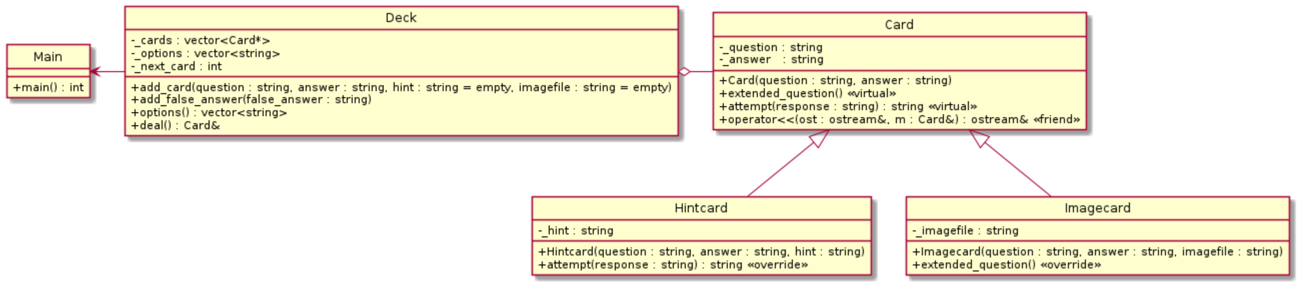

Main
+main() : int
Deck
-_cards : vector<Card*>
-_options : vector<string>
-_next_card : int
+add_card(question : string, answer : string, hint : string = empty)
+add_false_answer(false_answer : string)
+options() : vector<string>
+deal() : Card&
Card
-_question : string
-_answer : string
+Card(question : string, answer : string)
+attempt(response : string) : string
+operator<<(ost : ostream&, m : Card&) : ostream& «friend»
Hintcard
-_hint : string
+Hintcard(question : string, answer : string, hint : string)
+attempt(response : string) : string «override»

## Extreme Bonus

Derive (*more* inheritance!) a new type of flash card called Imagecard from class Card. Imagecard takes a third parameter - std::string imagefile - which is the filename of an image to be displayed just before _question, preferably in-terminal (although a pop-up window is acceptable). This enables questions such as "Which class relationship does this UML notation represent?"

A suggested class diagram is shown below, although you are free to deviate as needed to solve this extreme bonus. You may also install additional libraries and applications, as long as you provide screenshots of your code running to help us grade your work.

**Add, commit, and push all files** to your *private* cse1325 GitHub repository **often**. Much more detail is provided in Hints below.


Main
+main() : int
Deck
-_cards : vector<Card*>
-_options : vector<string>
-_next_card : int
+add_card(question : string, answer : string, hint : string = empty, imagefile : string = empty)
+add_false_answer(false_answer : string)
+options() : vector<string>
+deal() : Card&
Card
-_question : string
-_answer : string
+Card(question : string, answer : string)
+extended_question() «virtual»
+attempt(response : string) : string «virtual»
+operator<<(ost : ostream&, m : Card&) : ostream& «friend»
Hintcard
-_hint : string
+Hintcard(question : string, answer : string, hint : string)
+attempt(response : string) : string «override»
Imagecard
-_imagefile : string
+Imagecard(question : string, answer : string, imagefile : string)
+extended_question() «override»

# Hints

## Full Credit Hints

### *Class Card*

This class is fairly straight-forward, with just two strings as attributes - the question and the correct answer, both set by the constructor and neither are permitted getters.

The attempt method accepts a response as a parameter. If the response matches the answer, the string "Correct!" is returned. Otherwise, the string "X - Correct answer was " and the correct answer is returned. **Be sure to make the comparison case-insensitive** so that e.g., a response of "class" matches a correct answer of "Class". (Hint: The toupper() function might help.)

Overload the << operator, which streams the question.

### *Class Deck*

Deck includes a vector of Cards, which starts empty but is built by repeated calls to the add_card method. The add_card method accepts the same parameters as Card's constructor, enabling a new Card to be constructed and pushed onto the vector of Cards.

In addition to the vector of Cards, class Deck also keeps a vector of strings named _options listing the possible answers. The add_card method pushes its answer parameter to this vector. Additional "false answers" (that is, answers that have no matching card at all) can be added to the vector of strings using method add_false_answer. This permits the list of answers to be longer than the list of questions.

The remaining two methods are used for actually displaying and responding to the flash cards.

Method options returns a copy of the vector of strings named _options. This can be used to give the student a list of possible answers, which makes the flash cards a bit easier. Sort the _options vector before returning it to make it easier to find known answers. Hint:

Method deal returns a reference to one flash Card. If no cards have been added yet, throw a runtime_error exception. Otherwise, use the _next_card attribute to track which cards have already been dealt and which is up next. When method deal is called but no cards remain undealt, simply shuffle the vector of Cards and start over! (Optional Hint: Given this feature, one way to ensure the cards are shuffled on the first deal after a card is added is by setting _next_card to the number of Cards in the Deck in method add_card.) Consider the function std::shuffle in the standard library.

### *Writing a Makefile*

The Makefile will be similar to the one you created for P03, except you have an extra class to compile.

### *Writing Operators*

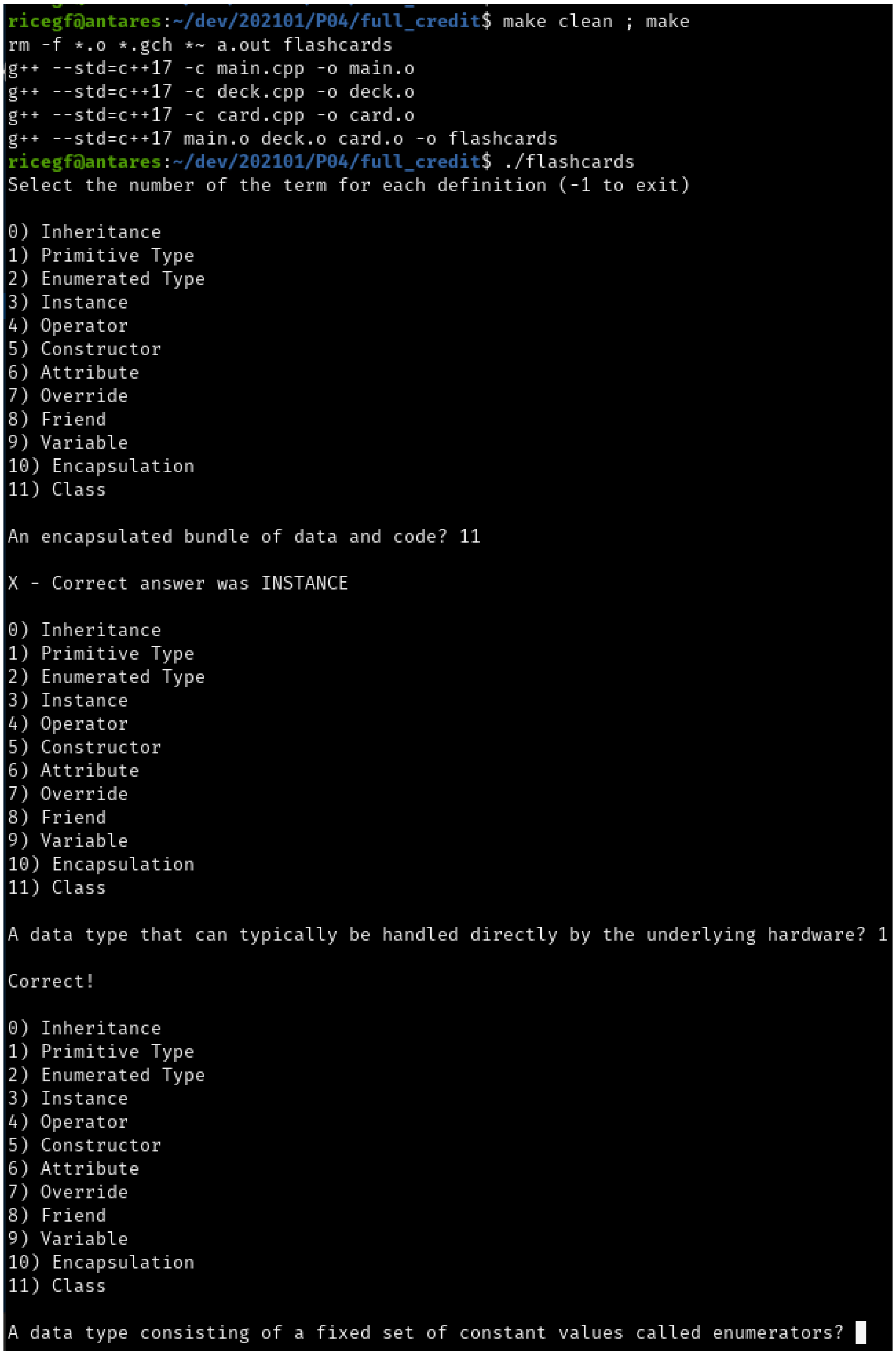
The streaming out operator for Card must be a friend function (why?). The function signature for class X is

```
std::ostream& operator<<(std::ostream& ost, const X& x);
```

**Remember to return ost** at the end of operator<<, otherwise you'll get a segfault!

Reminder: Operator signatures are provided in Lecture 05 or obtained from e.g., https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B. They will be provided to you on the exam - you do NOT need to memorize them.

An example program execution is on the next page.



```
ricegf@antares:~/dev/202101/P04/full_credit$ make clean ; make
rm -f *.o *.gch *~ a.out flashcards
g++ --std=c++17 -c main.cpp -o main.o
g++ --std=c++17 -c deck.cpp -o deck.o
g++ --std=c++17 -c card.cpp -o card.o
g++ --std=c++17 main.o deck.o card.o -o flashcards
ricegf@antares:~/dev/202101/P04/full_credit$ ./flashcards
Select the number of the term for each definition (-1 to exit)

0) Inheritance
1) Primitive Type
2) Enumerated Type
3) Instance
4) Operator
5) Constructor
6) Attribute
7) Override
8) Friend
9) Variable
10) Encapsulation
11) Class

An encapsulated bundle of data and code? 11

X - Correct answer was INSTANCE

0) Inheritance
1) Primitive Type
2) Enumerated Type
3) Instance
4) Operator
5) Constructor
6) Attribute
7) Override
8) Friend
9) Variable
10) Encapsulation
11) Class

A data type that can typically be handled directly by the underlying hardware? 1

Correct!

0) Inheritance
1) Primitive Type
2) Enumerated Type
3) Instance
4) Operator
5) Constructor
6) Attribute
7) Override
8) Friend
9) Variable
10) Encapsulation
11) Class

A data type consisting of a fixed set of constant values called enumerators?
```

## Bonus Hints

Class Hintcard is derived from class Hint. Change Card's attributes to protected so that Hintcard can access them.

Do NOT repeat inherited attributes (_question and _answer) or members and friends that don't change (operator<<) in hintcard.h - they are *inherited*.

Hintcard MUST have its own constructor - constructors never inherit. Delegate to Card's constructor to initialize the _question and _answer attributes, then construct Hintcard's attributes.

DO make Card::attempt virtual, and DO use the override keyword for Hintcard::attempt. This tells the compiler to ensure the latter overrides the former for instances of Hintcard.

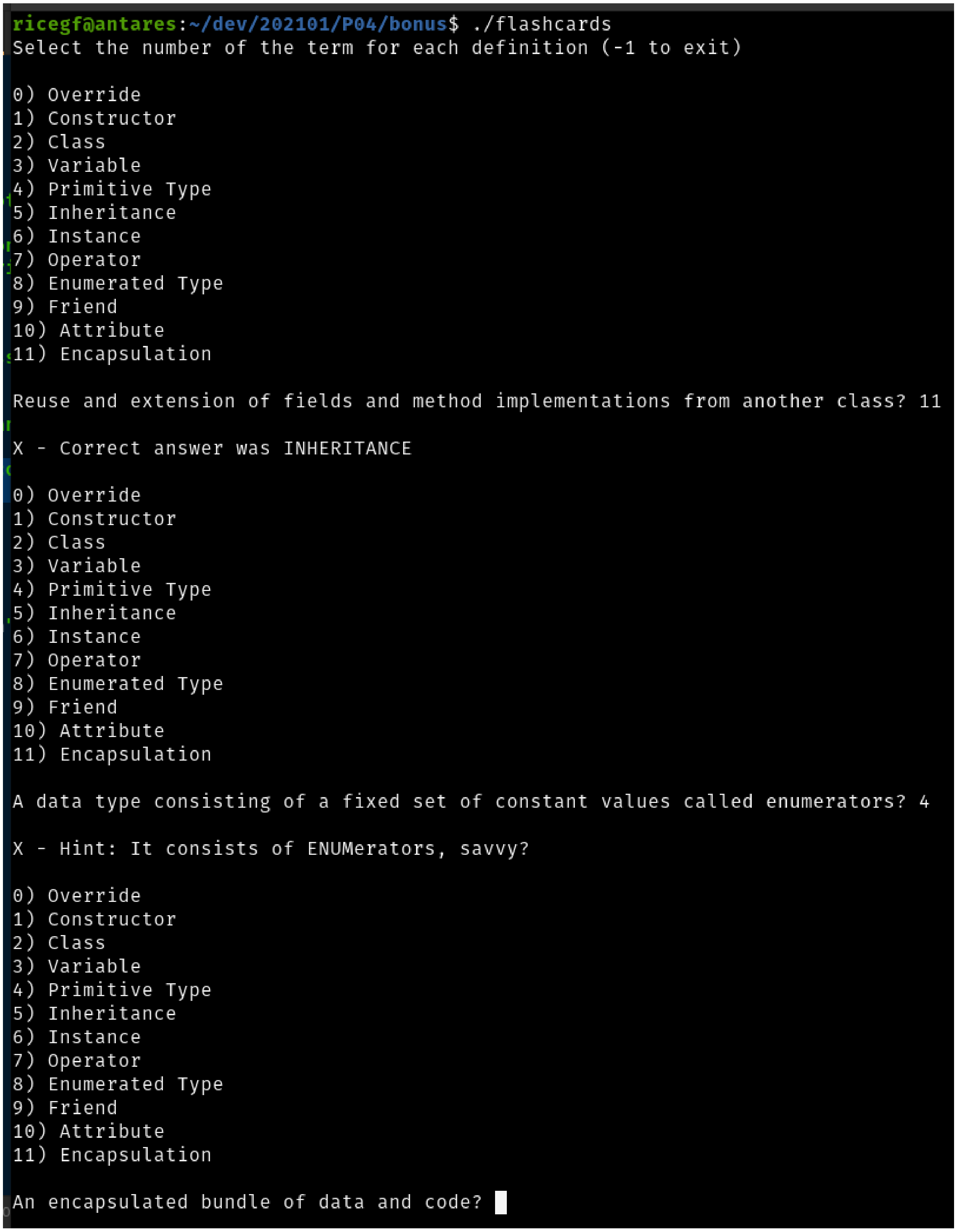
Modify Deck::_cards to store *pointers* to cards on the heap (created using `new Card{...` or `new Hintcard{...}`). Modify Deck::add_card to accept an *optional* third parameter named hint with "" as the default value. If hint is of zero length, push a pointer to a new Card onto _cards, otherwise push a pointer to a new Hintcard.

Update your main function to include hints for *some* (but not all) cards in the deck by adding a third parameter to Deck's add_card method, e.g., for 'class' your hint might be "We use it to write classy programs!"

Update your Makefile to build and link hintcard.cpp into executable flashcards.

NOTE: Deal *must* return a reference (or, if you love pointers, a pointer). Otherwise, only Card instances will be dealt. We'll explain why later this semester. Isn't C++ fun? (And yes, I realize inheritance isn't a particularly elegant solution to this problem, since the Card class could be modified to support both Card and Hintcard behaviors. But you need to practice inheritance on a simple problem, so we'll take this approach anyway. You can sue me for malpractice later.)

An example program execution is on the next page.



```
ricegf@antares:~/dev/202101/P04/bonus$ ./flashcards
Select the number of the term for each definition (-1 to exit)

0) Override
1) Constructor
2) Class
3) Variable
4) Primitive Type
5) Inheritance
6) Instance
7) Operator
8) Enumerated Type
9) Friend
10) Attribute
11) Encapsulation

Reuse and extension of fields and method implementations from another class? 11

X - Correct answer was INHERITANCE

0) Override
1) Constructor
2) Class
3) Variable
4) Primitive Type
5) Inheritance
6) Instance
7) Operator
8) Enumerated Type
9) Friend
10) Attribute
11) Encapsulation

A data type consisting of a fixed set of constant values called enumerators? 4

X - Hint: It consists of ENUMerators, savvy?

0) Override
1) Constructor
2) Class
3) Variable
4) Primitive Type
5) Inheritance
6) Instance
7) Operator
8) Enumerated Type
9) Friend
10) Attribute
11) Encapsulation

An encapsulated bundle of data and code? 
```

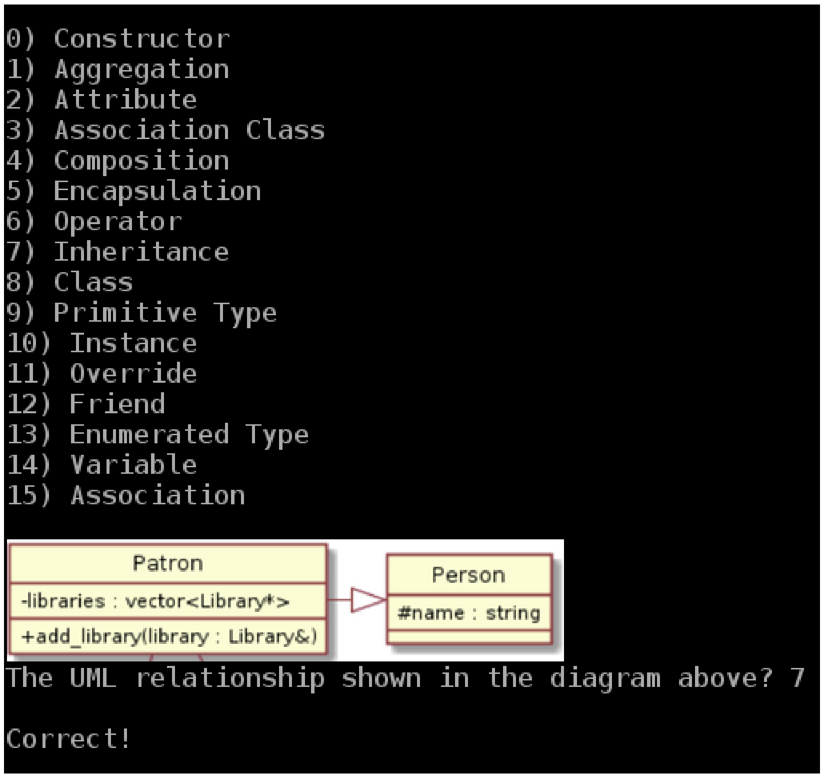
## Extreme Bonus Hints

You may modify the UML diagram as needed to accomplish this extreme bonus, AND you may install additional libraries and applications (but please include screenshots to help the graders!).

The class diagram shows one way to execute arbitrary code whenever a Card (or a class derived from it) is streamed to the terminal. In operator<<, calling the extended_question() method just before streaming out the question enables code to display the image. For Card, extended_question() does nothing (and Hintcard inherits this behavior). But Imagecard's overridden extended_question() method displays the image.

Displaying the image is part of the challenge. You may have a library in mind that works from the command line, or perhaps you'd prefer to just invoke an external application. Some terminal programs add ANSI-like control code extensions to display images within the terminal, in which case the extended_question method isn't even required. Ecosia (or other search engine of your choice) is your friend.

You might find some question resources at `cse1325-prof/P04/extreme_bonus` to adapt for your solution. Here's one part of an example execution.



```
0) Constructor
1) Aggregation
2) Attribute
3) Association Class
4) Composition
5) Encapsulation
6) Operator
7) Inheritance
8) Class
9) Primitive Type
10) Instance
11) Override
12) Friend
13) Enumerated Type
14) Variable
15) Association
```

Patron
-libraries : vector<Library*>
+add_library(library : Library&)
Person
#name : string

```
The UML relationship shown in the diagram above? 7

Correct!
```

## The Professor's cse1325-prof

The professor for this class provides example code, homework resources, and (after the due date) suggested solutions via his cse1325-prof GitHub repository.

If you haven't already, clone the professor's cse1325-prof repository with "`git clone https://github.com/prof-rice/cse1325-prof.git`". This will create a new directory called "cse1325-prof" in the current directory that will reflect the GitHub repository contents. To update it with the professor's latest code at any time, change to the cse1325-prof directory and type "`git pull`". Or, if you prefer, delete the old one (`rm -fr cse1325-prof`) and simply clone it again.