Com S 362
Fall 2003

Object-Oriented Analysis and Design

Solutions to
Exam 3: Analysis, Design, and Implementation

This test has 5 questions and pages numbered 1 through 7.

Exam Process

Questions 1 and 2 can be done at any time, and should be turned in at the end of the test along with all of the front matter in the test.

There is a use case and system sequence diagram for the remaining questions following the first two questions.

Starting with question 3 on this exam, each question builds on the answer from the previous question. To aid grading and to prevent you from getting too far off track, when you complete an answer for one question, you will trade your answer to that question for our standard solution for that question. You must then use our standard solution for this question when answering the next question. For example, when you finish question 3, you will trade in your answer for question 3 for our standard solution to question 3, and then you will then use our answer to question 3 to solve question 4. Similarly, you will use our answer for questions 3 and 4 to solve question 5. *For these questions, be sure to put your name on each page you hand in!*

Reminders

This test is open book and notes. However, it is to be done individually and you are not to exchange or share materials with other students during the test. So if you have materials on your team project you wish to refer to during the test, please make copies.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For diagrams and programs, clarity is important; if your diagrams or programs are sloppy and hard to read, you will lose points. Correct syntax also makes some difference.

1. (5 points) This is a problem about the GRASP design patterns. How does the Creator pattern lower coupling in a design? That is, explain briefly the way in which use of the Creator pattern acts to lower coupling.

By having the creation of an object of class *C* be created by a class that would be fairly strongly coupled to instances of class *C* anyway, the pattern lowers the overall coupling of the system, since it avoids introducing (stronger) coupling from addtional classes to *C*.

2. (5 points) This is another problem about GRASP design patterns. Recall the calculator classes we worked with in the beginning of the semester. In that design we used an interface, `FormulaType` (with an `evaluate` method), and several classes that implemented it: `Formula`, `BinaryFormula`, `Diff`, `Mult`, `Sum`, and `Negation`. In terms of cohesion, what would be the disadvantages of combining all of these into one class, `ArithFormula`?

The `ArithFormula` type would be less cohesive than the other classes. It would have more methods, and be responsible for knowing how to do more tasks (addition, subtraction, etc). This would make it harder to understand, reuse, and maintain. It woud be affected by more changes.

Use Case for the following problems

Use Case: Check Out Books

Primary Actor: Worker

Stakeholders and Interests:

- Worker: wants fast, and easy check out of books.
- Patron: wants fast check out, and does not want to be charged for books they did not check out.
- Library: wants fast check out of books, and wants to make sure that all books that leave the library have been checked out. Wants to allocate books fairly.
- Government: wants to protect investment in books and keep costs down. Wants to promote learning and citizen happiness.

Preconditions: The Worker has been authenticated.

Success Guarantee (Postconditions): The System remembers that the Patron has checked out the books.

Main Success Scenario (or Basic Flow):

1. The Worker tells the System the identity of a patron who wishes to check out books.
2. The System confirms that the patron is allowed to check out books, and remembers the patron's identity.
3. The Worker tells the system the identity of a book this patron is checking out.
4. The System confirms that the book can circulate, calculates the due date based on whether the patron is a faculty member or a student, and records that the patron has checked out this book, which is due on the calculated due date, and makes that information available from the library catalog.
5. The System tells the Worker the due date (which also confirms that the book has been checked out).

The Worker repeats steps 3-5 until indicates done.

Extensions (or Alternative Flows):

2a. If the patron is not allowed to check out books because they have violated some library policy (for example, if the patron has not paid their university bill or library fines):

1. The System tells the Worker that the patron is not allowed to check out books and the reason for this prohibition.
2. The use case ends.

4a. If the book that is being checked out is non-circulating:

1. The System tells the Worker why the book is non-circulating.
2. The use case continues from step 3 in the main success scenario.

Special Requirements

- There are different due dates depending on the kind of patron one is dealing with. For example, faculty can take out books for the whole academic year, whereas students can only take them out for a limited time.
- The System must respond to the Worker, at least giving some progress indication, within 3 seconds, 95% of the time.
- Workers are experts, because they use the system continuously, so the interface should have minimal interaction and should minimize the physical effort involved; for example, workers should not have to be prompted.
- Displays for the worker should be visible from one meter away.
- The system should be quiet.

Technology and Data Variations List:

3a. Barcode scanners are normally used to identify books.

3b. Books without barcodes have to be entered manually.

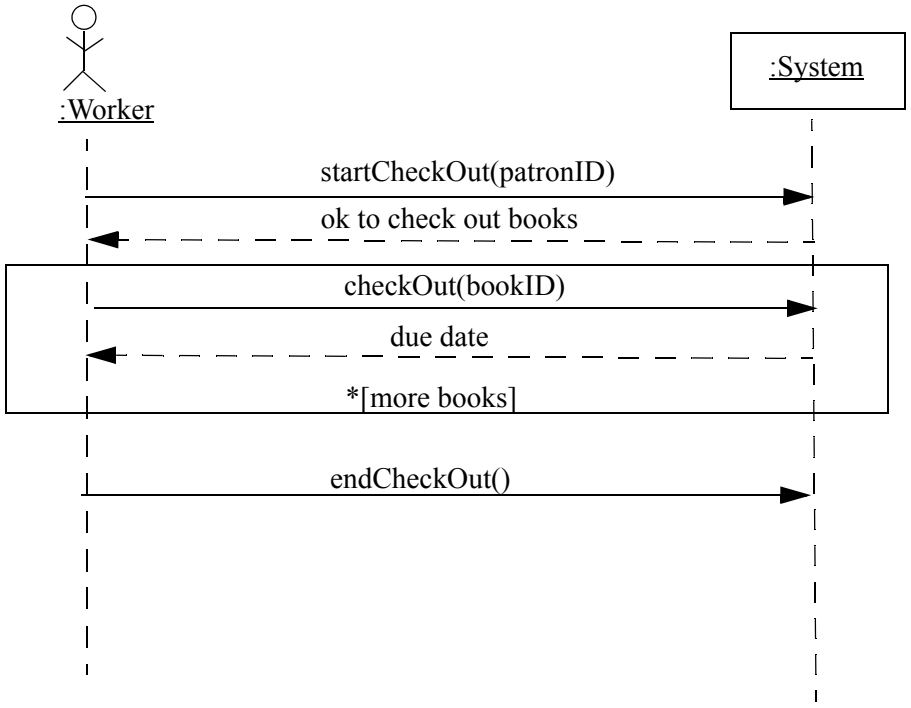
Frequency of Occurrence: nearly continuous.

Open Issues: How to deal with failures and recovery?.

How to deal with overnight check out of reserve items.

How to deal with library books that are unknown to the circulation system.

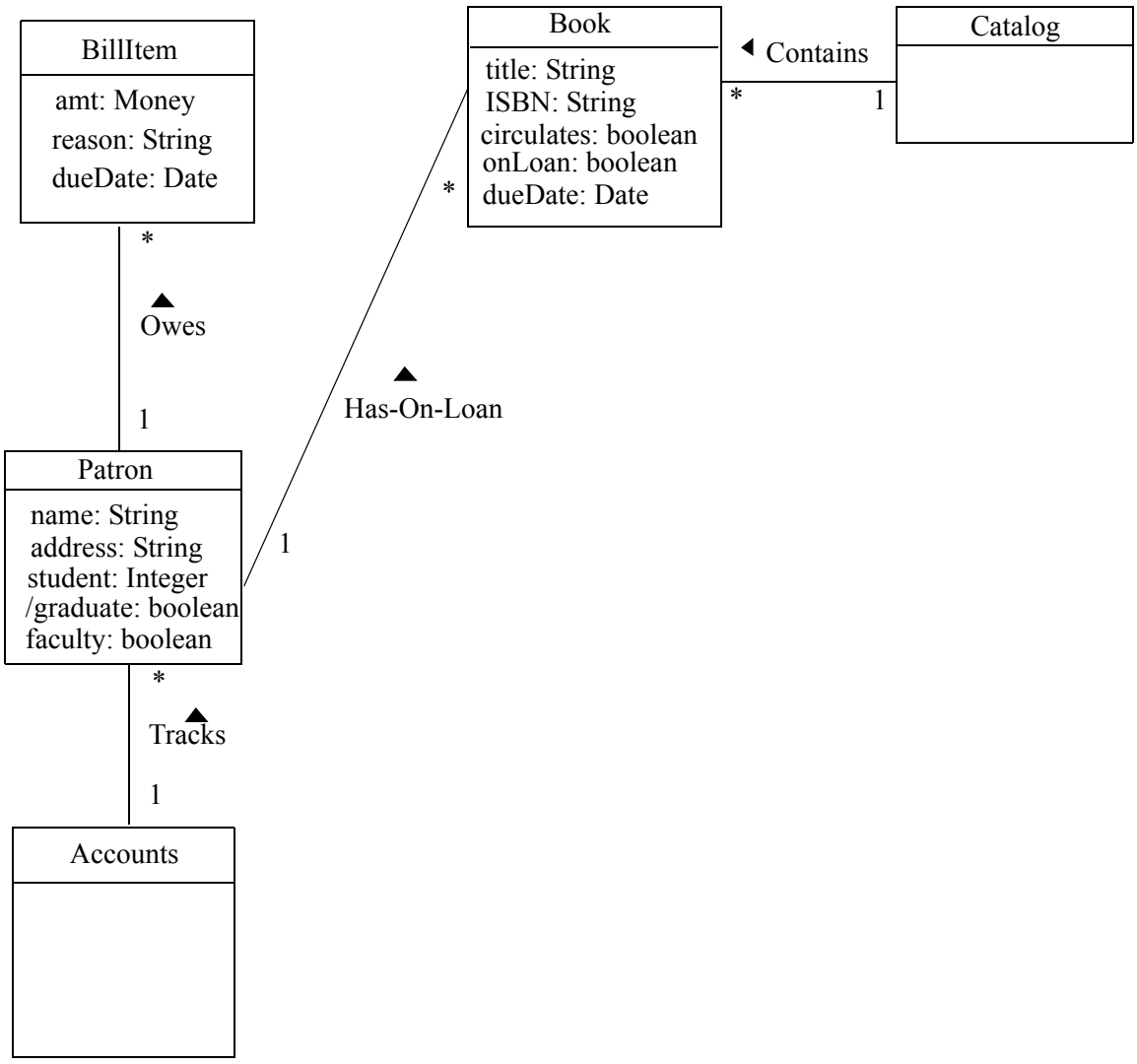
The following is a system sequence diagram (SSD) for the main success scenario of the above use case. The argument “patronID” is intended to be of type String, similarly for bookID.



3. (30 points) In this problem you will write a domain model, i.e., a conceptual class diagram, which should be relatively complete for the use case on the previous pages. It should include associations and any attributes that are useful for the application logic layer. Consider the entire use case. However, do not include conceptual classes for external systems or for basic classes that would be expected to be in the programming language, such as strings. You can include classes that are more useful versions of classes you would expect to find in the programming language.

   Answer: There may be other possibilities, of course; but please use this to solve the remaining questions.

Domain Model

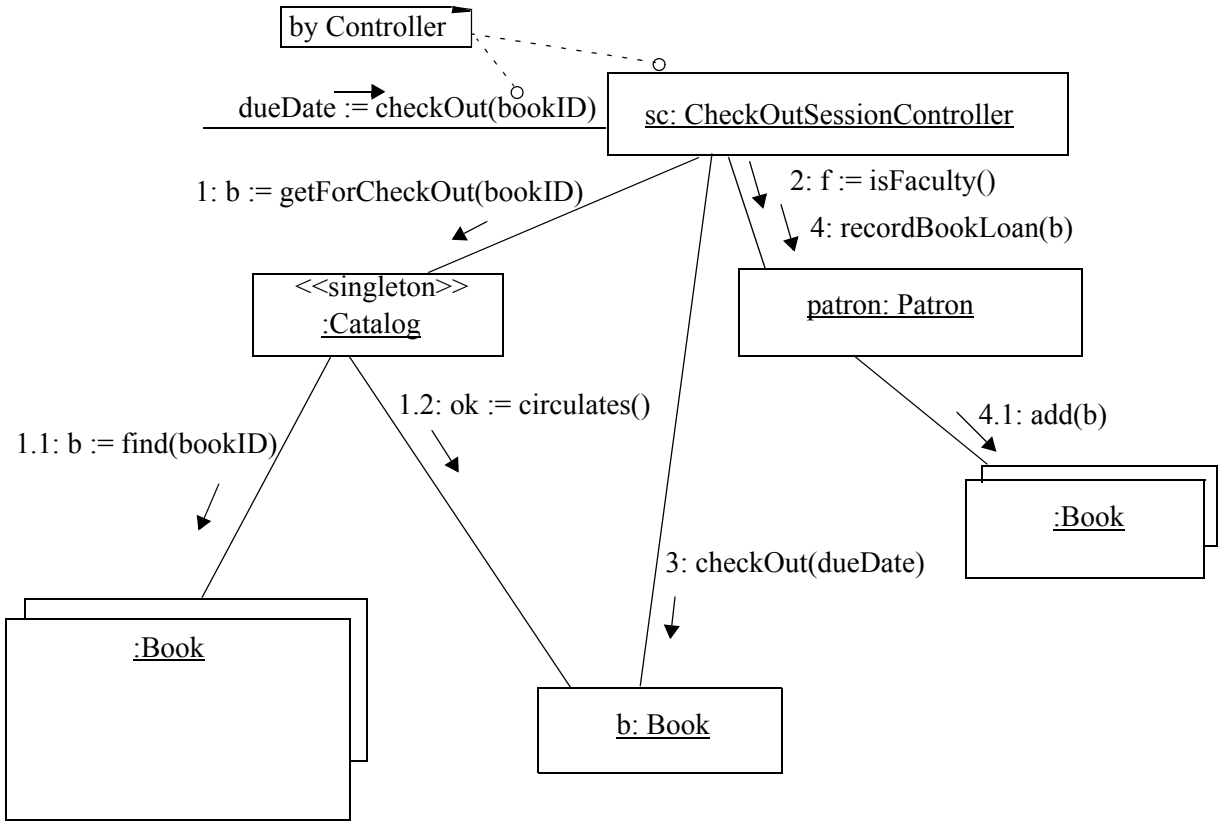


4. (30 points) In this question you will do the design for just the `checkOut` system operation from the system sequence diagram found on page 4. (Don't include the `startCheckOut` operation!) Your design should be based on the standard solution for the previous problem. You will be designing the `checkOut` operation in the main success scenario, but you must check the conditions that are needed to confirm that the `checkOut` operation stays on the happy path.

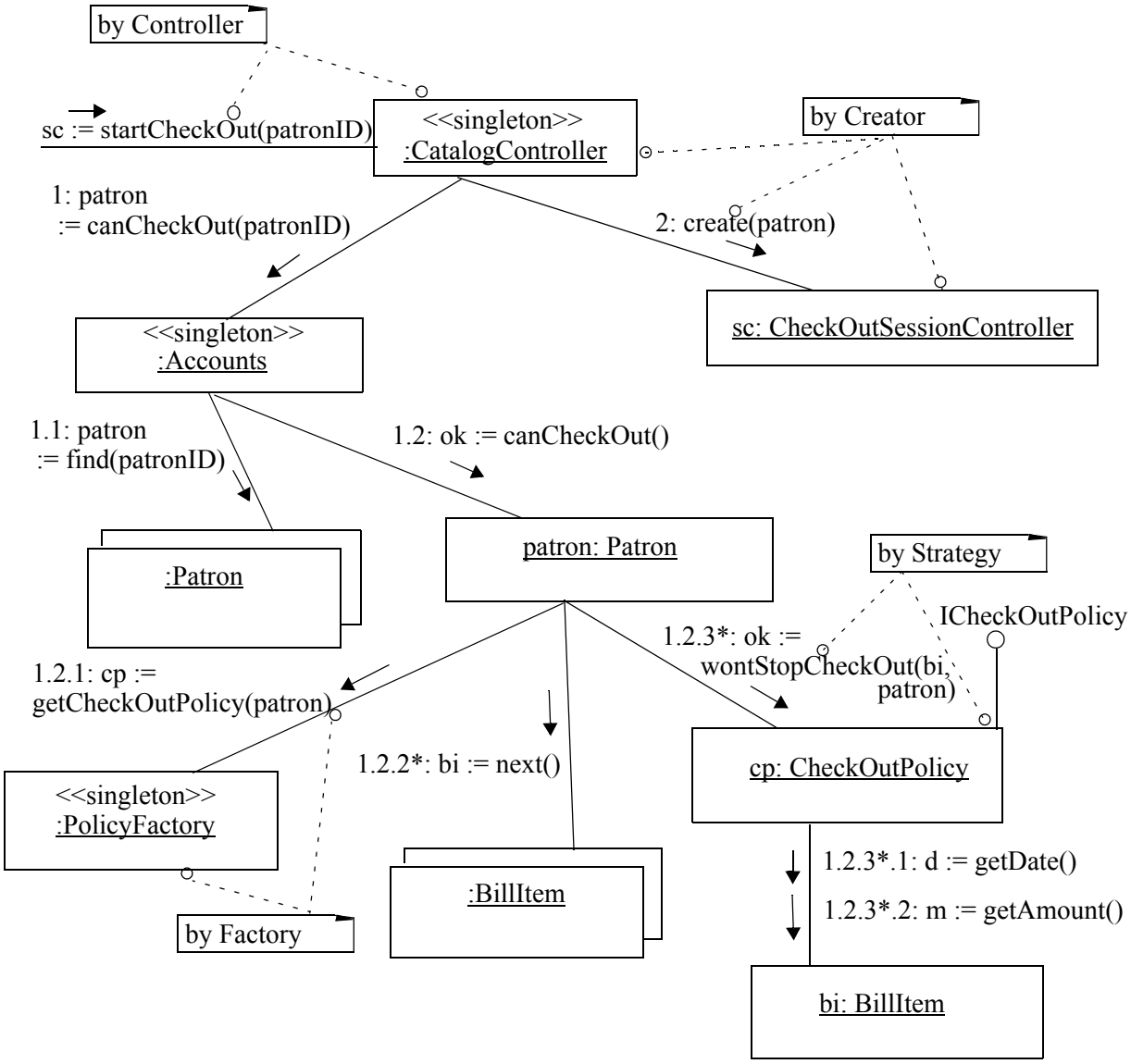
   Your design of the `checkOut` operation is to be recorded using an interaction diagram. You may use either the UML sequence diagram or collaboration diagram notation.

   You need only show the design patterns or principles are used to assign responsibilities that are *different than* Creator or Information Expert.

Answer: The following is a collaboration diagram for the `checkOut` system operation.(A sequence diagram would also be fine.) All of the design patterns used and not marked below are Information Expert.



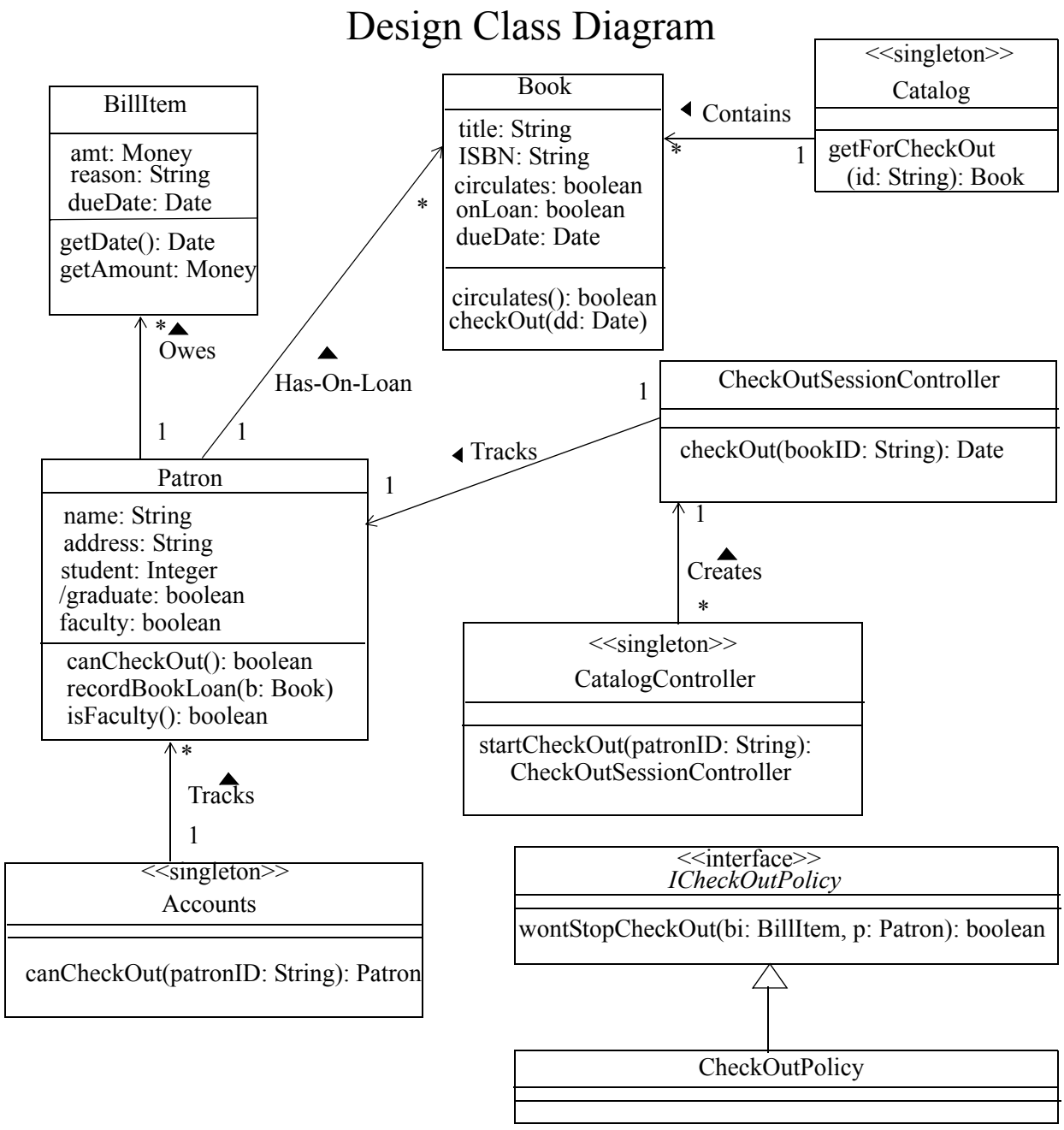
The following is a collaboration diagram for the startCheckOut system operation; it will be used in the next problem. Note that `ICheckOutPolicy` is an interface.



5. (30 points) Based on the standard solution for the previous problems, draw a UML design class diagram that summarizes the design of the two system operations `startCheckOut` and `checkOut`. (That is, you *don't* have to include classes, attributes, or methods needed to implement other system operations or other scenarios.)

   On your diagram, include types for all method arguments and results (for methods that have results). Don't include classes, such as collections, that are part of Java's built-in libraries. To save time, you can omit the interface `ICheckOutPolicy` and the class `CheckOutPolicy`.

   Answer: The following is our answer to this problem (including the optional classes).



Please don't show this to others still taking the test.
And please hold any arguments or comments about the test until after everyone is done.