(1) **15 minutes max.** Recall the metamodel of all class diagrams with associations from class:

```
Class
- name
- anchoredAt
  1
  - attrOf
  *
  - hasAttr

Attributes
- name
- visibility?

Association
- name
- anchoredAt
  1

Association End
- name
- anchors
  *

- visible?
- cardinality
- endOf
  1

- hasEnd
  2
```

a) What is the minimal addition to this metamodel diagram that permits inheritance relationships among classes in instances of the revised metamodel?

b) What additional constraints, in English, would you need to retain the sanity of such diagrams?

c) Does the original diagram (above) conform to your revised metamodel of b)? Yes or No + briefly explain why.
(2) 15 minutes max. Short Answer:

a) One of these two class diagrams is wrong. Which one and why?

```
  A  *  1  B
   -End2   -End1

  C  *  1  D
   -End4   -End3
```

b) Consider the following 2-class Java program to the right. Rewrite the program after applying the following sequence of refactorings. You may create intermediate programs, but clearly identify the final result.

R1. create a factory for S named New.

R2. rename variable j to z;

R3. move method times to class R.

R4. move method sum to class R via field r.

```java
class S {
    R r;
    int i;
    int j;

    S(int ii, int jj) {
        i = ii;
        j = jj;
    }

    int sum() {
        int z = 8;
        return i+j;
    }

    int times(R rr) {
        return i*rr.k;
    }
}

class R {
    int k = 7;
}
```
(3) 15 minutes max. Again recall the class diagram from problem (1):

Here is a database that conforms to the above meta-model (and what you would expect to be reasonable constraints):

<table>
<thead>
<tr>
<th>class</th>
<th>attribute</th>
<th>assoEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>C#</td>
<td>Name</td>
<td>All</td>
</tr>
<tr>
<td>b</td>
<td>Books</td>
<td>a1</td>
</tr>
<tr>
<td>l</td>
<td>Librarian</td>
<td>a2</td>
</tr>
<tr>
<td>p</td>
<td>Publisher</td>
<td>a3</td>
</tr>
<tr>
<td>g</td>
<td>General Book</td>
<td>a4</td>
</tr>
<tr>
<td>r</td>
<td>Reference Book</td>
<td>a5</td>
</tr>
<tr>
<td>u</td>
<td>User</td>
<td>a6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Draw the UML class diagram that is represented by this database.
(4) **15 minutes max.** A common refactoring is to push an association “through” an abstract class to its subclasses:

By making class A associations reference abstract class B’s subclasses, a constraint must be added: each A instance is bound to a B1 or B2 instance, but never both.

a) What is the common IDE name of this refactoring, even though it doesn’t deal with the preservation of constraints?

b) Using the above refactoring and any that we have discussed in class along with their names – show that the left model can (or cannot) be mapped to the right model.

constraint: oval is connected to Box or Diamond, but not both
(1a) Minimal change is to add class-to-class link to represent inheritance relationships:

(1b) Additional constraints should be:

- no inheritance cycles
- Each class has 0..1 superclasses AND each class has any number of subclasses.

(1c) Yes it conforms. The original diagram just doesn’t have inheritance relationships. The tabular representation of the modified diagram adds a field to the class table. This column would contain nulls for the original diagram.
(2a) The top diagram is wrong: Open diamond implies a 0..1 cardinality.

```
A * 1 B
-End2 -End1
```

```
C * 1 D
-End4 -End3
```

(22) Refactored code:

```java
class S {
    R r;
    int i;
    int z;
    static S New(int ii, int jj) {
        return new S(ii, jj);
    }
    private S(int ii, int jj) {
        i = ii;
        z = jj;
    }
}
```
(3) The class diagram is:
(4a) push down – push member (association field) down to both (all) subclasses

(4b) Solution: