1 Question

(10 points) Solve Exercise 4.2.6 in your textbook. After completing parts (a)–(c), perform the following additional modification without introducing any new classes:

a) The type of the attribute commander is changed to be a pair of strings, the first of which is the rank and the second of which is the name.

Solution: (Underlined lines are the properties added to solve the problem). See Figure 1.

```plaintext
(01) class TG {
(02)    attribute real number;
(03)    attribute struct Com {String rank, String name} commander;
(04)    relationship Set<ship> unitsOf
(05)    inverse Ship::assignedTo;
(06) }
```

Figure 1: Solution to Question 1(a)

b) A ship is allowed to be assigned to more than one task group.

Solution: See Figure 2.

```plaintext
(01) class Ship {
(02)    attribute string name;
(03)    attribute integer yearLaunched;
(04)    relationship Set<TG> assignedTo
(05)    inverse TG::unitsOf;
(06) }
```

Figure 2: Solution to Question 1(b)

c) Sister Ships are identical ships made from the same plans. We wish to represent, for each ship, the set of its sister ships (other than itself). You may assume that each ship’s sister ships are Ship objects.

Solution: See Figure 3.

d) Each ship assigned to a task group is given a priority (for that task group). The ship with the highest priority is always given a 1 (for that task group) and the numbers increase in steps of 1. Ships may thus get different priorities for different task groups they are part of. However, ships do not conversely assign any priority for the task groups they are part of (they don’t care!).

Solution: See Figure 4.
2 Question

(20 points) Express the E/R diagram in Fig. 2.7 in the textbook in ODL.

Solution: See Figure 5.

3 Question

(20 points) Solve Exercise 2.1.3 in your textbook but in ODL. Even though the question in the textbook asks for an E/R diagram, you should provide an ODL design and not an E/R diagram. Note the constraint that the team captain is one of the players. Explain how you enforce this constraint in your design. In other words, what aspect of your design prevents a player from another team from being named as a captain of a team? Note that the favorite players of a fan might not belong to the fan’s favorite team(s).

Solution: See Figure 6.

Note: In the Team class, the hasPlayers attribute has the type a list <Player> and the attribute captain is an integer. The value of this attribute is the index of the player who is the captain of the team. In this manner, we ensure that the team captain is one of the players in the team. However, making hasPlayers a list implies that we cannot automatically ensure that each player in the team appears only once in the list.
class Movie {
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enum Type {color, blackAndWhite} filmType;
    relationship Set<Contract> movieContractsFor inverse Contract::theMovie;
}

class Star {
    attribute string name;
    attribute string address;
    relationship Set<Contract> starContractsFor inverse Contract::theStar;
}

class Studio {
    attribute string name;
    attribute string address;
    relationship Set<Contract> studioContractsFor inverse Contract::theStudio;
}

class Contract {
    attribute real salary;
    relationship Movie theMovie inverse Movie::movieContractsFor;
    relationship Star theStar inverse Star::starContractsFor;
    relationship Studio theStudio inverse Studio::studioContractsFor;
}

Figure 5: Solution to Question 2
(01) class Team {
(02)     attribute string name;
(03)     attribute integer captain;
(04)     relationship set<Color> uniformColor inverse Color::byTeam;
(05)     relationship set<Fan> teamFavBy inverse Fan::favTeam;
(06)     relationship list<Player> hasPlayers inverse Player::playFor;
(07) }

(01) class Player {
(02)     attribute string name;
(03)     relationship Set<Fan> playerFavBy inverse Fan::favPlayer;
(04)     relationship Team playFor inverse Team::hasPlayers;
(05) }

(01) class Color {
(02)     attribute string name;
(03)     relationship Set<Team> byTeam inverse Team::uniformColor;
(04)     relationship Set<Fan> colorFavBy inverse Fan::favColor;
(05) }

(01) class Fan {
(02)     attribute string name;
(03)     relationship Set<Team> favTeam inverse Team::teamFavBy;
(04)     relationship Set<Color> favColor inverse Color::colorFavBy;
(05)     relationship Set<Player> favPlayer inverse Player::playerFavBy;
(06) }

Figure 6: Solution to Question 3
4 Question

(25 points) For this question and the next, we underline attributes in a schema if the attribute is part of the key for the schema (See Chapter 3.4.2 of your textbook). An E/R diagram when converted to relations (using the mechanical construction that we know and love) gives rise to the following relations:

\[ R(a, b, c) \quad S(a, d) \quad T(a, d, f, g) \]

You may assume that the same symbols refer to the same attribute and different symbols refer to different attributes (e.g., the attributes \(a\) in the relations \(R\), \(S\), and \(T\) are the same). Your task is to reverse-engineer the E/R diagram from these relations; in other words, what E/R diagram could have produced these relations?

For full credit, give two different E/R diagrams that could have produced these (and only these) relations.

**Solutions:**

![E/R Diagram 1]

![E/R Diagram 2]

5 Question

(25 points) An ODL schema when converted to relations (using the mechanical construction that we know and love) gives rise to the following relations:

\[ R(a) \quad S(b) \quad T(c) \quad U(d) \quad V(a, b, c, d) \]

You may assume that the same symbols refer to the same attribute and different symbols refer to different attributes (e.g., the \(a\) in \(R\) and \(V\) are the same). Your task is to reverse-engineer the ODL schema from these relations; in other words, what ODL listing could have produced these relations?
For full credit, give an ODL schema and describe what kind of situation would have created this schema in the first place.

**Solution:** See Figure 7.

(01) class R (extent R1 key a) {
    (02)    attribute a;
    (03)    relationship set<V> RtoV inverse V::VtoR;
    (04) }

(01) class S (extent S1 key b) {
    (02)    attribute b;
    (03)    relationship set<V> StoV inverse V::VtoS;
    (04) }

(01) class T (extent T1 key c) {
    (02)    attribute c;
    (03)    relationship set<V> TtoV inverse V::VtoT;
    (04) }

(01) class U (extent U1 key d) {
    (02)    attribute d;
    (03)    relationship set<V> UtoV inverse V::VtoU;
    (04) }

(01) class V {
    (02)    relationship R VtoR inverse R::RtoV;
    (03)    relationship S VtoS inverse R::StoV;
    (04)    relationship T VtoT inverse R::TtoV;
    (05)    relationship V VtoU inverse R::UtoV;
    (06) }

Figure 7: Solution to Question 5

We underline only the primary key attributes in a relation, so all attributes in V are needed for the key. Therefore, using subclasses is incorrect since the superclass contains the key attribute.

An E/R diagram where V is a 4-way relationship between the entity sets R, S, T, and U is equivalent to the ODL class.