Assume that a new strategy game is released. Parts of the game are described from both the player and developer’s perspectives.

From a player’s perspective:

Background:

The player selects a faction to support before starting to play. The selected faction has a special building named a Construction Yard from which the player can request / buy units to battle enemy factions. The Construction Yard can supply two types of units, namely Ornithopters and Devastators. The following is a sequence of actions that the player takes and the events associated with them:
1. The player creates a Construction Yard.
2. The player then orders and receives 10 Ornithopters and 10 Devastators from the Construction Yard.
3. The player sends the 20 units to the enemy base camp for a first assault.
4. The assault does not work out in the player’s favour and all 20 units are destroyed.
5. The player returns to the Construction Yard at the base camp in order to get more units and tries again.

From the developer’s perspective:

Background:

The developer has created two classes representing units, namely Ornithopter and Devastator. Both of these classes inherit from an abstract class Unit. Having this arrangement will not only make it easy to handle units uniformly throughout the code, but will also allow for the easy addition of more units, like for example, if an expansion pack is created for the game at a later date. There is also a class named ConstructionYard which represents the Construction Yard of the game. This class encapsulates two objects, one of type Ornithopter and one of type Devastator. The following is a high level breakdown of what happens in the code from the developer’s
point of view corresponding to the actions taken by the player above.

1. The program receives input from the player requesting a Construction Yard building. A `ConstructionYard` object is instantiated. As part of the object’s initialization, an `Ornithopter` object and `Devastator` object are instantiated and placed in the `ConstructionYard` object.

2. The program receives input requesting 10 Ornithopters and 10 Devastators. This request is processed by asking the `ConstructionYard` object to return 10 clones of each type of unit.

3. The player provides input to move the units to a specified destination on the map and to start an attack. The program updates the units accordingly.

4. Game logic takes place in which the units are destroyed and the game state is updated. One of the actions includes deallocating the destroyed units.

5. The program once again receives input from the player requesting additional Ornithopters and Devastators. This request is processed by asking the (already existing) `ConstructionYard` object to return additional clones of each type of unit. Because the original units encapsulated within the `ConstructionYard` object have not been destroyed with the others, the request can be processed and the player is granted more units by cloning the original units.
Questions

- Can you draw a UML class diagram to represent the classes from the developer’s perspective?
- Can you motivate your choice of pattern?
- Can you identify all of the participants?