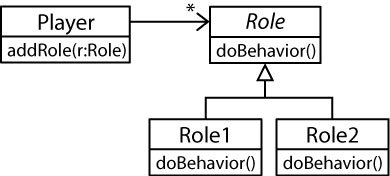
**Chapter 1b – Player-Role Pattern**

**Definition**

1. Intent: Model the situation where a Player can change roles or possess multiple roles depending on the context.

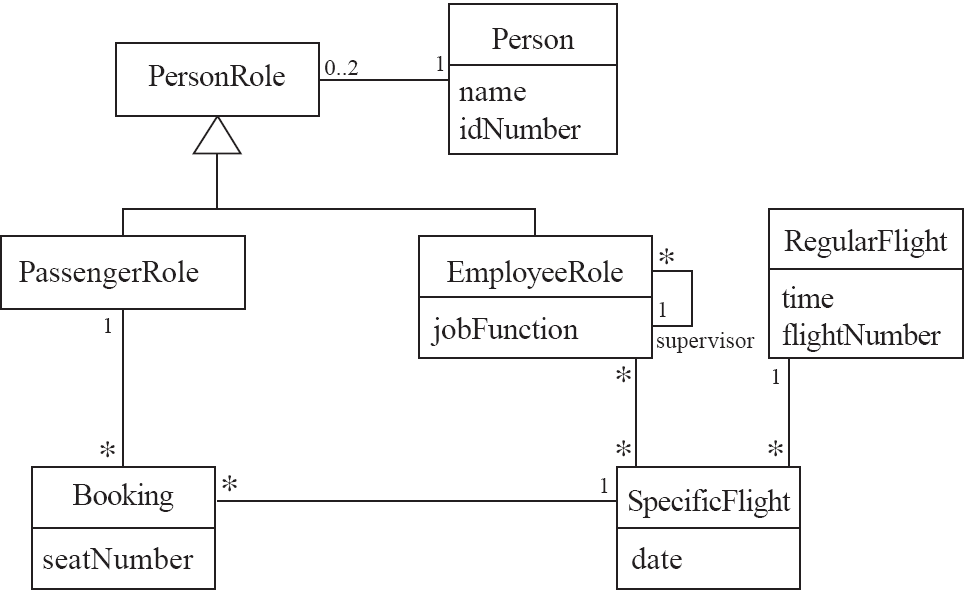


1. In some ways, this pattern is similar to the Strategy pattern; however, they differ in *intent*. The Strategy pattern wraps up interchangeable algorithms. The Player-Role pattern solves the modeling problem where an object may need to play different roles in different contexts.

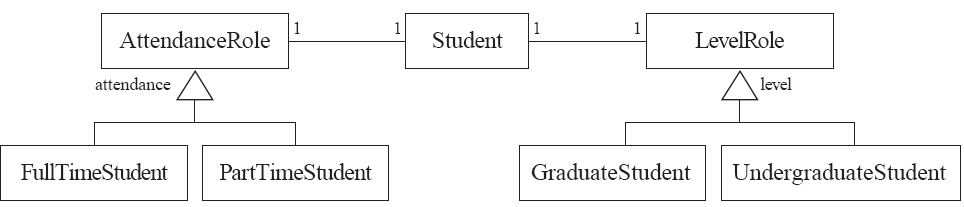
**Examples**

1. In the situation below, we see that a Person can have 0, 1, or 2 *PersonRoles*. In other words, a Person could be:

* in the system, but not currently registered for any flights, nor an employee (0 roles).
* registered for a flight (1 role, PassengerRole)
* an Employee (1 role, EmployeeRole)
* an Employee who is also registered for a flight (on their personal leave time) (2 roles)



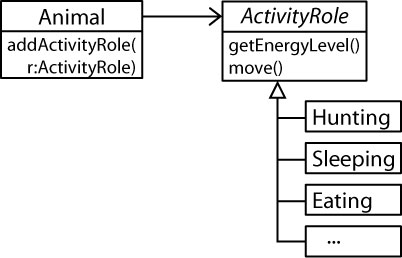
1. A student may be part-time or full-time and either a graduate student or undergraduate student. These roles can change. A part-time student one semester may be a full-time student in another semester. In this case, inheritance can be used but leads to class explosion and is not very flexible or extensible.



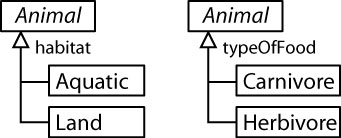
In this sense, it can be seen that the Player-Role pattern prevents an object from having to change classes, which is what would happen if we employed an inheritance based solution. Thus, a *player* can simply change *roles* as opposed to having to change classes. We like to avoid objects having to change classes for several reasons:

* When an object changes classes, then we have increased the coupling by introducing another class that clients must interact with.
* When an object needs to change classes, presumably, only some things need to change. In other words, much of the state may remain the same. So, it is wasteful and error-prone to copy such an object to a new class.

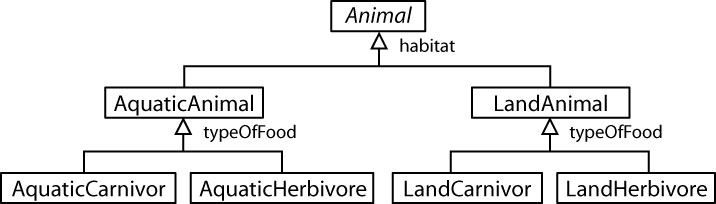
1. An animal may need to *sleep* during the day and *hunt* at night. Thus, its behavior needs to change according to time of day. Hunting could also have a strategy: wolves hunt hares and other small prey but prefer large hoofed animals like elk, deer, moose.



1. The Player-Role pattern solves the multiple-discriminator problem where we want to differentiate objects based on multiple criteria. For instance, consider modeling animals based on their habitat and the type of food that they eat:



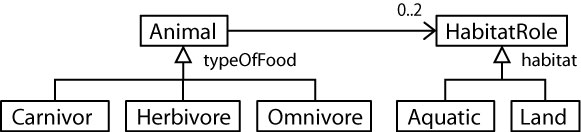
To combine these, we could model the situation like this:



The problems with this approach:

* We may have duplication of code in some of the deepest classes, e.g. AquaticCarnivore and LandCarnivore.
* This is a completely inflexible approach; it is not extensible. For instance, what happens when we introduce another discriminator? The number of classes increases exponentially. For example how about a *when active* discriminator (night, day, twilight).

One solution to this problem is to use the Player-Role pattern in this way:



Instead of using inheritance for *typeOfFood*, we could have used association to make it a *typeOfFoodRole*. In other words, there may be situations where we need multiple types of roles.

**References**

1. *Object-Oriented Software Engineering*, by Lethbridge and Laganiere, pages 228-230.
2. *Object Oriented Modeling and Design*, by Rumbaugh, 1991.