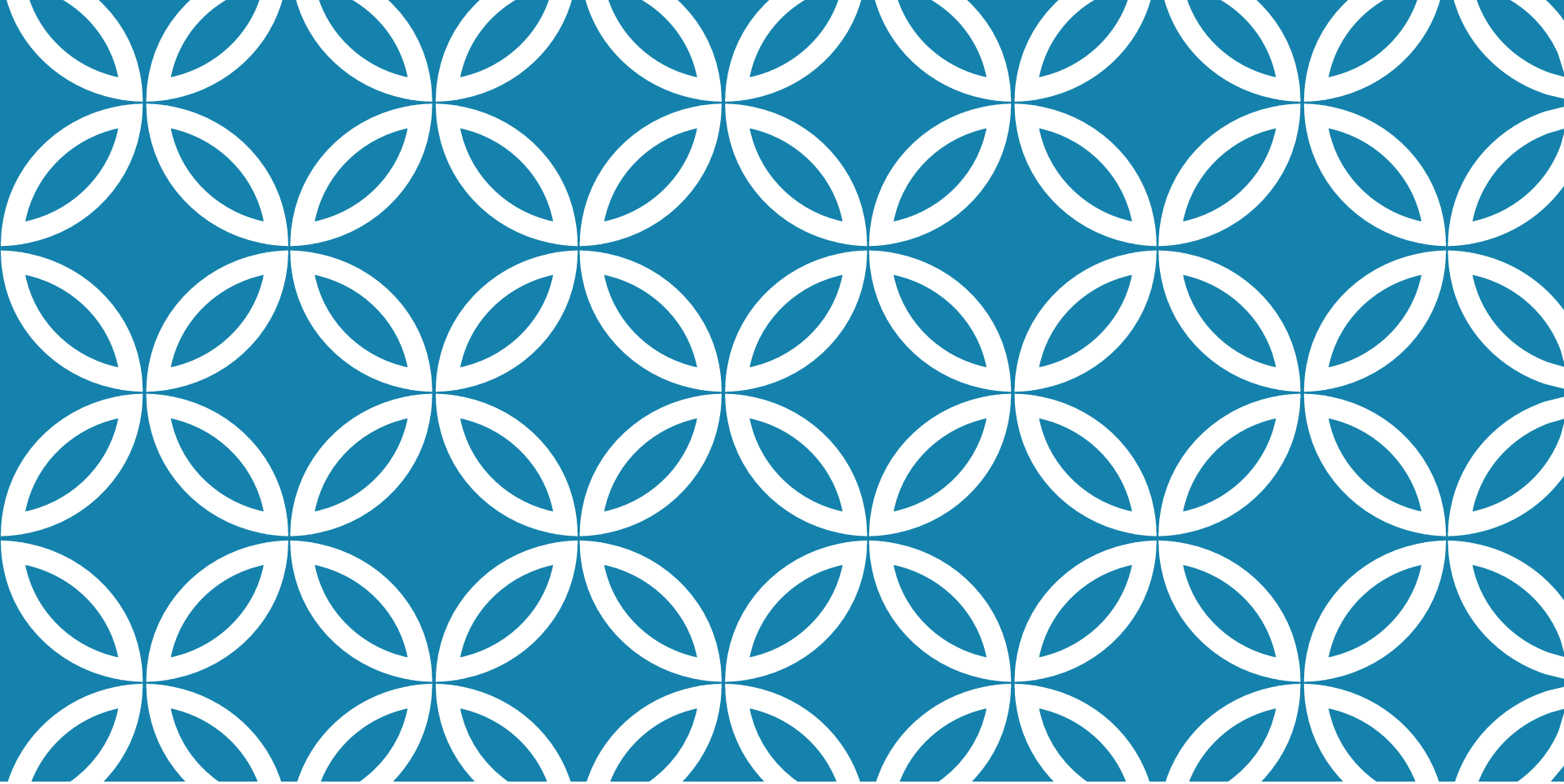


(IS352) SOFTWARE ENGINEERING I

Lecture 9
Player-Role, Proxy, & General-Hierarchy Design Patterns

LECTURE OBJECTIVES:

Discussing the following Design Patterns:

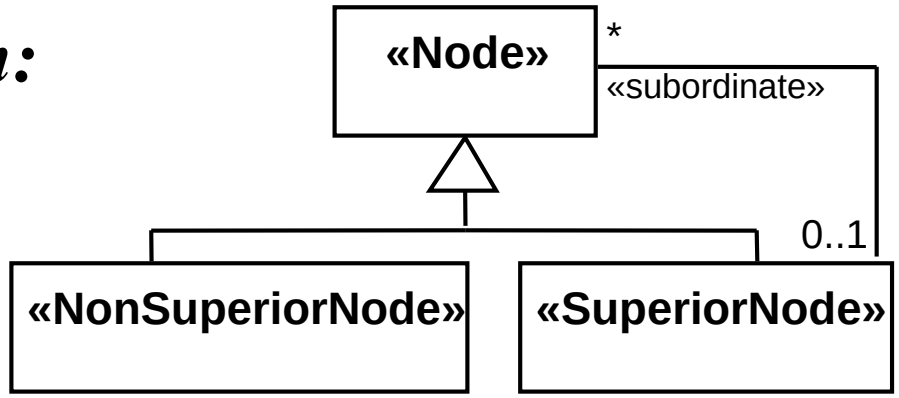
- Proxy
- General Hierarchy
- Player-Role

THE GENERAL HIERARCHY PATTERN

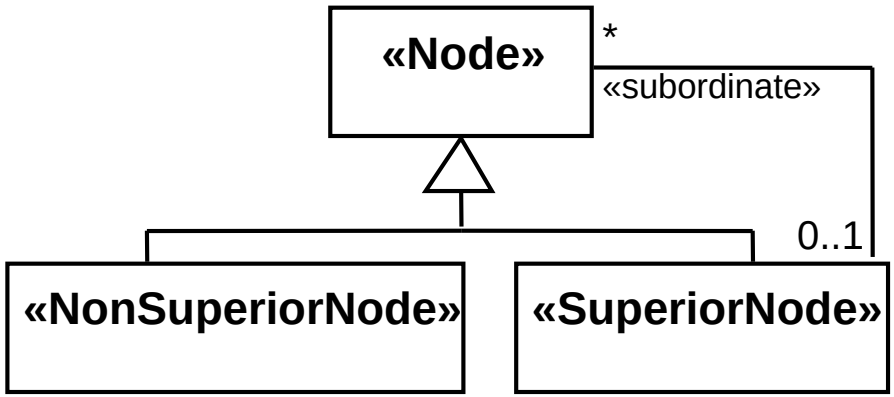
- ***Context*:**
  - **— Objects in a hierarchy can have one or more objects above them (superiors), and one or more objects below them (subordinates).**
  - **— Some objects cannot have any subordinates.**
- ***Problem*:**
  - **— How do you represent a hierarchy of objects, in which some objects cannot have subordinates?**
- ***Forces*:**
  - **— You need a flexible way of representing the hierarchy that prevents certain objects from having subordinates.**
  - **— All the objects have many common properties and operations.**

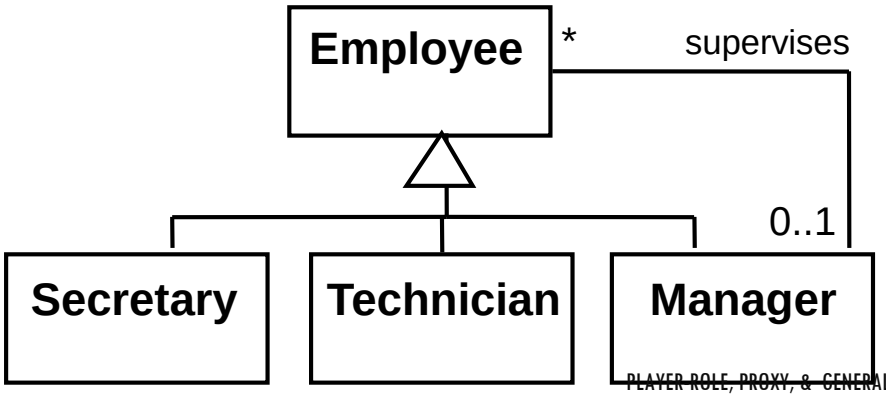
THE GENERAL HIERARCHY PATTERN

- ***Solution:***

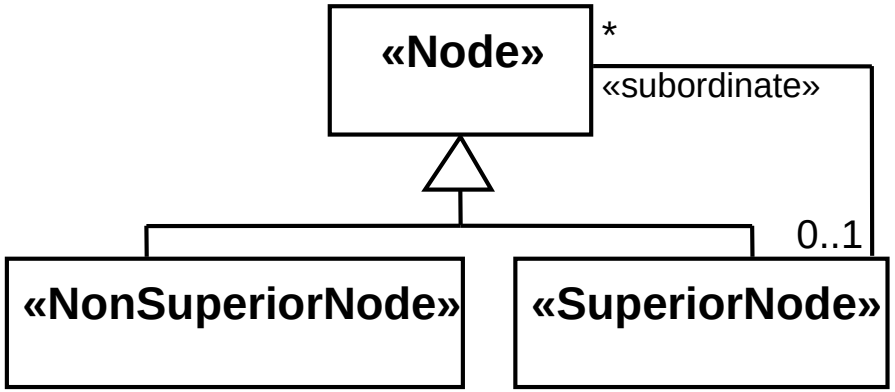


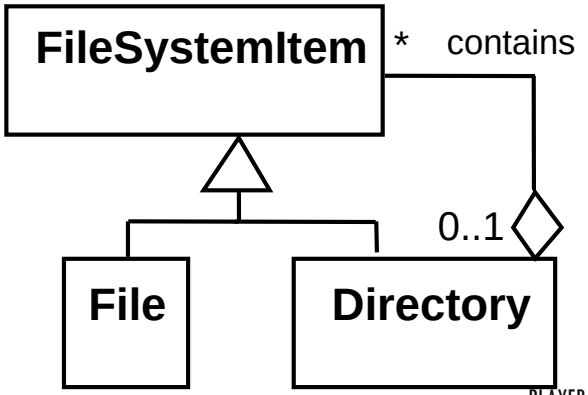
THE GENERAL HIERARCHY PATTERN





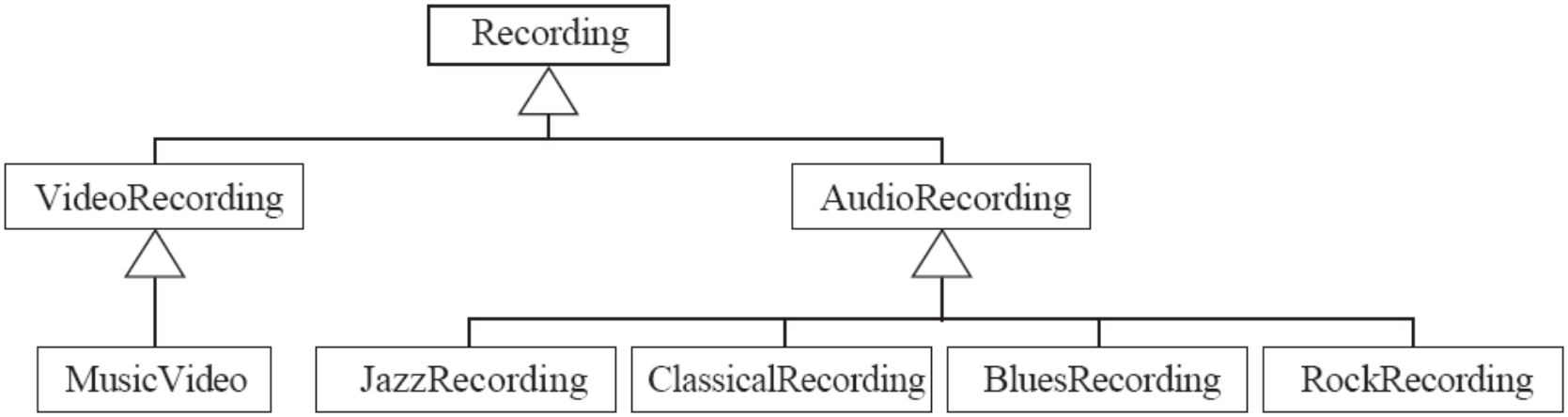
THE GENERAL HIERARCHY PATTERN





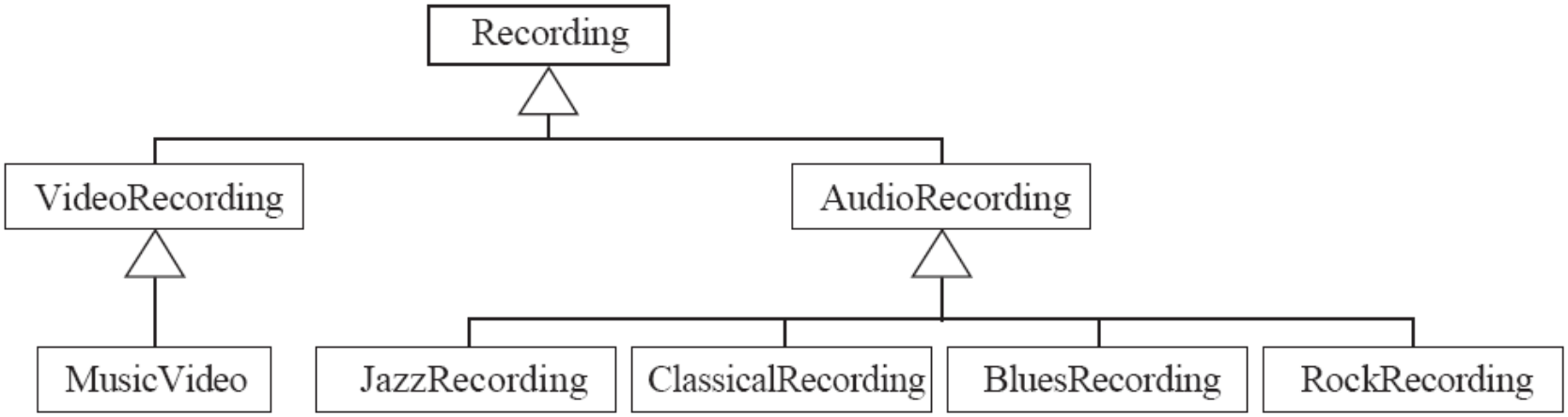
THE GENERAL HIERARCHY PATTERN

Anti-pattern:

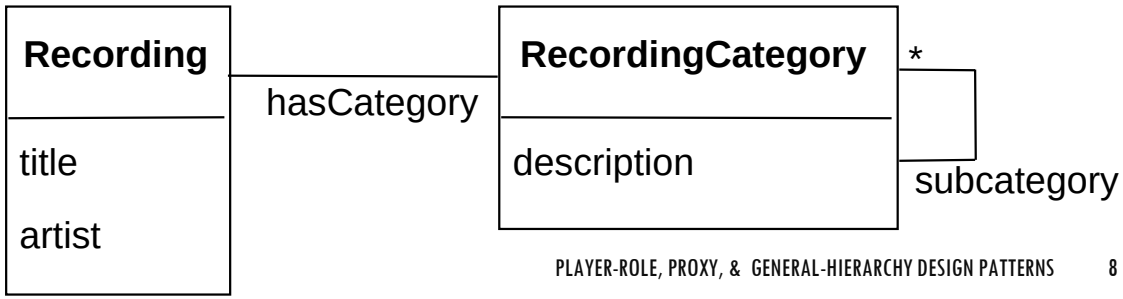


THE GENERAL HIERARCHY PATTERN

Anti-pattern:



Better:



THE PLAYER-ROLE PATTERN

- ***Context*:**
  - **An object may play different roles in different contexts of an application.**
  - **An object may need to change roles throughout the course of the application.**
  - **An object may not need to play one or both roles simultaneously.**
- ***Problem*:**
  - **How do you best model players and roles so that a player can change roles or possess multiple roles?**

THE PLAYER-ROLE PATTERN

- ***Forces*:**
  - **It is desirable to improve encapsulation by capturing the information associated with each separate role in a class.**
  - **Making two classes to describe the same object is not good OO design.**
  - **If you put the properties of each role in the same class, and if an object needs to play only one of the roles, then the properties of the other roles would not make sense in the class, since they would never be used.**
  - **You want to avoid multiple inheritance.**
  - **You cannot allow an instance to change class.**

THE PLAYER-ROLE PATTERN

Example 1: Class Design of Animals

Context:

– Aquatic and land animals have common and distinct properties.

– Should I create one class for aquatic animal and one class for land animal?

– Or, should I create one class for both aquatic and land animals?

– An animal's role may need to change.

Issues:

– It is desirable to improve encapsulation by keeping information related to aquatic properties in one class, and keep out land properties, and vice versa.

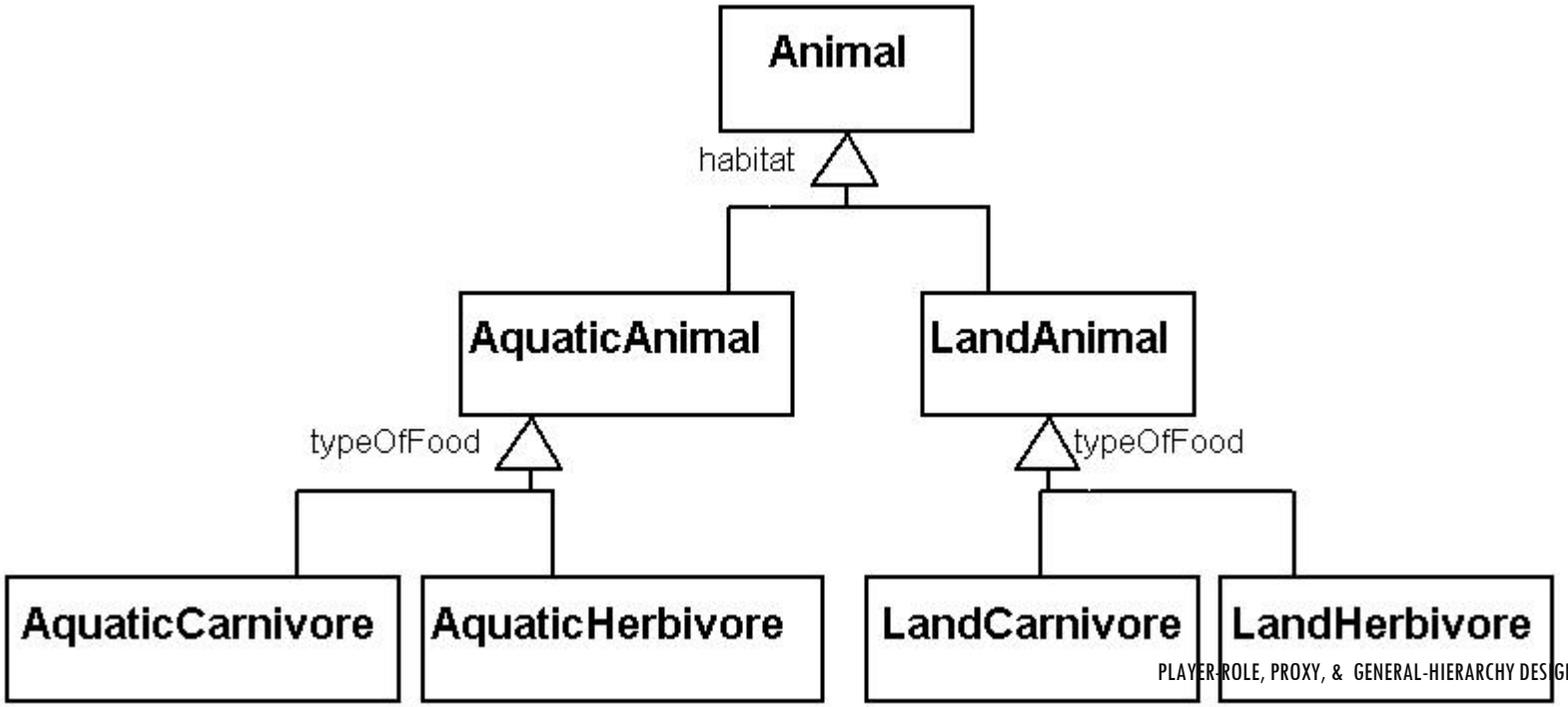
– Making two classes to describe a turtle is not good OO design.

– If you put the land and aquatic properties in the same class, and if Harry the turtle object needs to play only the land role, then the aquatic properties would not make sense in Harry, since they would never be used.

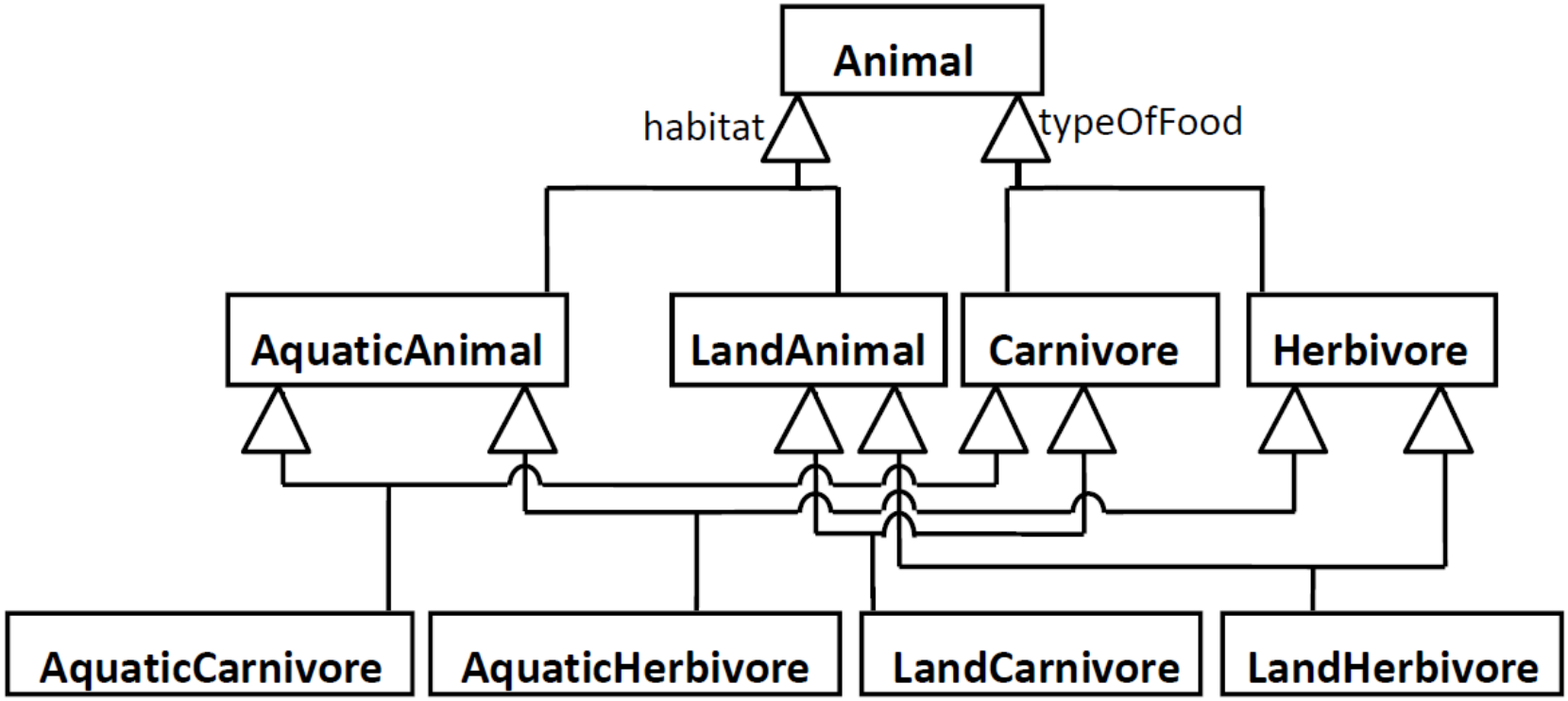
THE PLAYER-ROLE PATTERN

Two or more objects are required if the animal needs to play different roles, like aquatic and land carnivore, which is not good OO design.

- **Example of redundant hierarchy**



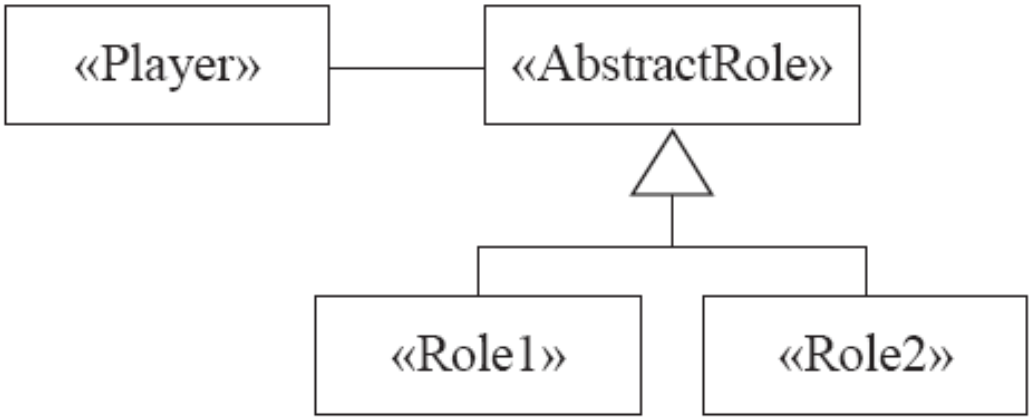
THE PLAYER-ROLE PATTERN



- Two or more objects are required if the animal needs to play different roles, like aquatic carnivore and land carnivore, which is not good OO design.
- An animal inherits properties it may not need: e.g., if it doesn't need to have habitat properties.
- Too complex: the class diagram is hard to read, let alone analysing the code.

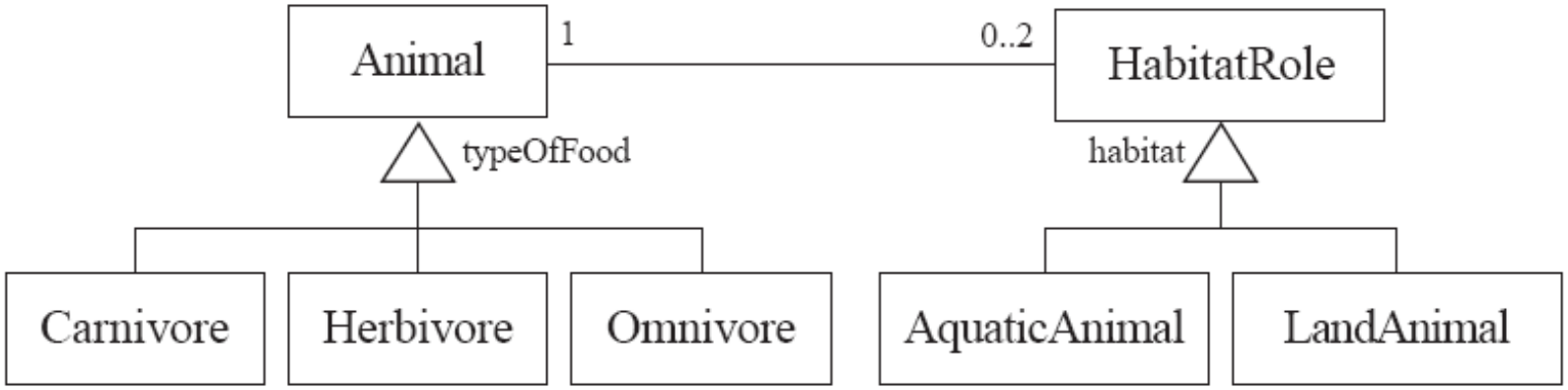
THE PLAYER-ROLE PATTERN

- ***Solution:***



THE PLAYER-ROLE PATTERN

Example 1:



An animal can be:

- ☐ Carnivore
- ☐ Herbivore
- ☐ Omnivore

An animal can have:

- ☐ No habitat role
- ☐ An aquatic habit role
- ☐ A land habitat role
- ☐ An aquatic and land habitat roles.

THE PLAYER-ROLE PATTERN

Example 2: Class Design of Students

Context:

– Part-time and full-time students have common and distinct properties.

– Should I create one class for part-time student and one class for full-time student?

– Or, should I create one class for both part-time and full-time students?

– A student's role may need to change.

Issues:

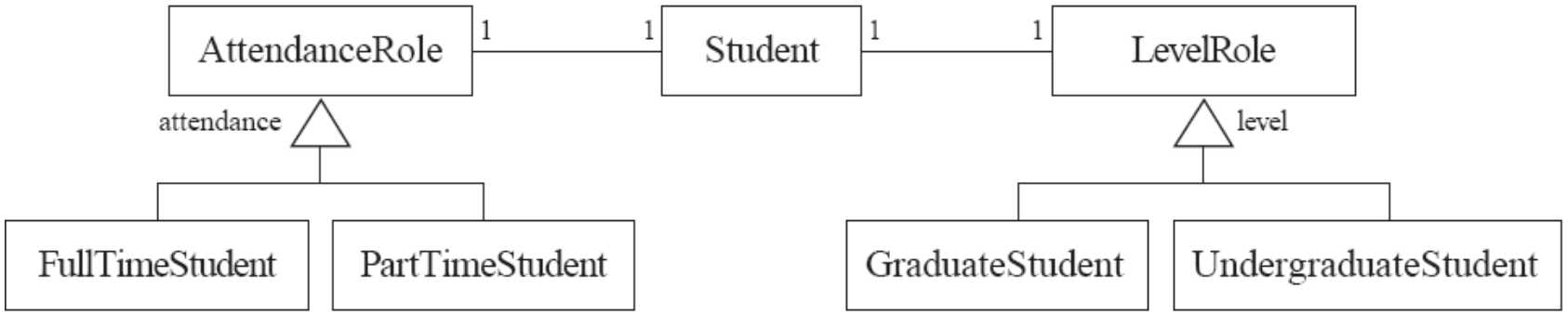
– It is desirable to improve encapsulation by keeping information related to only one role in one class.

– Making two classes to represent the same student is not good OO design.

– If you put the properties of each role in the same class, and if an object needs to play only one of the roles, then the properties of the other roles would not make sense in the class, since they would never be used.

THE PLAYER-ROLE PATTERN

Example 2:



A student can have:

- No attendance role.
- FT or PT, but not both.
- No Level role.
- GS or UG, but not both.

THE PLAYER-ROLE PATTERN

Antipatterns:

- Merge all the properties and behaviours into a single «Player» class and not have «Role» classes at all.
- Create roles as subclasses of the «Player» class.

THE PROXY PATTERN

- ***Context*:**
  - **Often, it is time-consuming and complicated to create instances of a class (*heavyweight* classes).**
  - **There is a time delay and a complex mechanism involved in creating the object in memory**
- ***Problem*:**
  - **How to reduce creating instances of a heavyweight class?**
- ***Forces*:**
  - **We want all the objects in a domain model to be available for programs to use when they execute a system's various responsibilities.**
  - **It is also important for many objects to persist from run to run of the same program**

THE PROXY PATTERN

- ***Cache Proxy***
- ***Security Proxy***
- ***Virtual (Model) Proxy***

THE PROXY PATTERN

CACHE PROXY

- ***A cache proxy improves performance when long-running tasks return seldom-changing results.***
- ***Obtaining data from a remote data server.***
  - ***The data seldom changes, but the size of each object is large, and the network connection is slow.***
  - ***Recently requested data can be stored (cached) locally, and subsequent calls for the same data are returned from the local cache.***
- ***A method calculates and returns prime numbers.***
  - ***1st call computes a large number or primes, and stores them in cache.***
  - ***Subsequent calls can be returned without recalculating the prime numbers.***

THE PROXY PATTERN

SECURITY PROXY

- ***A protection proxy adds a layer of security to the underlying subject object.***

- ***For example, the real object may access a database and retrieve sensitive data.***

- ***The protection proxy could add methods or properties that allow the client object to provide appropriate authentication before allowing the data to be returned.***

- ***It could also filter the data according to the rights of the authenticated user.***

THE PROXY PATTERN

MODEL PROXY (provides a form of lazy initialisation)

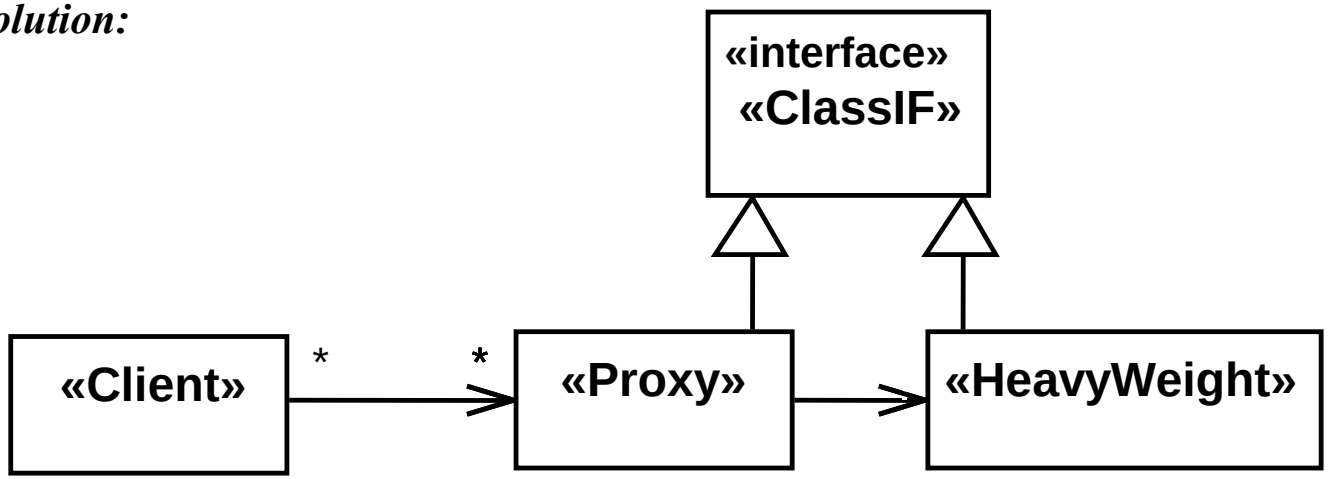
- ***A model proxy provides a simplified version of a complex object.***
- ***Only when the detail of the object is required is the main object actually populated, providing a form of lazy initialisation.***
- ***For example, a document may have a large number of high definition images, each of which is represented by an object***
  - ***The PC may not have enough memory to instantiate all of the objects***
  - ***When opening the document, the thumbnail, size and other easy-to-retrieve information would be held in a proxy object for each image.***
  - ***Only when an image needs to be displayed on the monitor would the entire image object be created and displayed on the monitor.***

THE PROXY PATTERN

- ***Solution:***

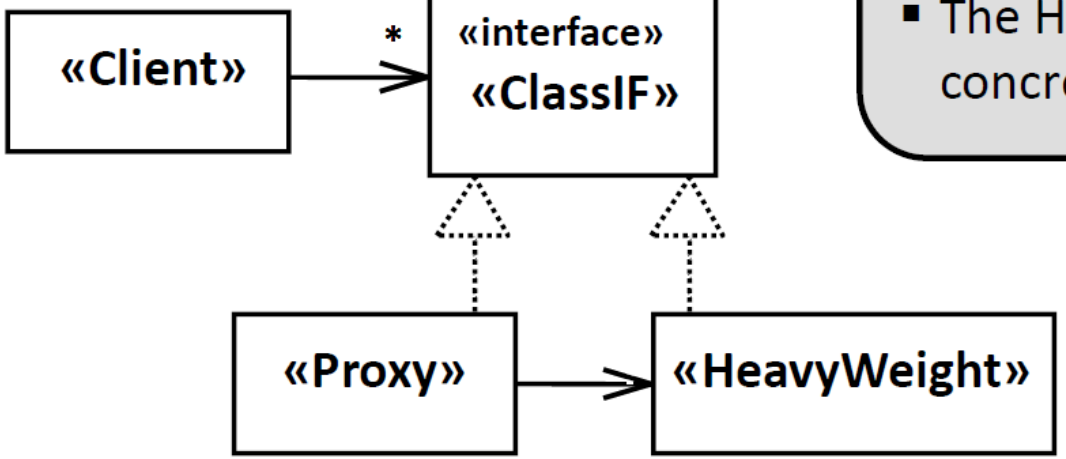
THE PROXY PATTERN

- ***Solution:***



THE PROXY PATTERN

- *Solution:*

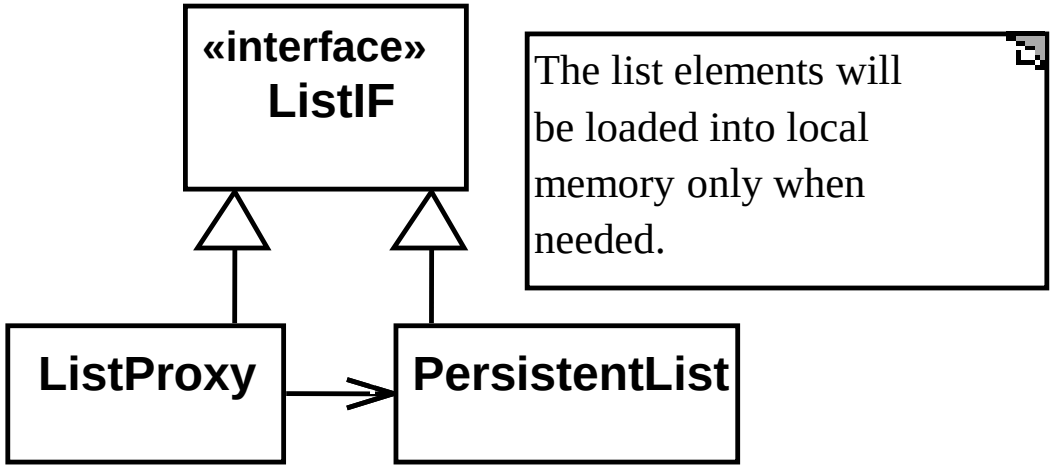


- The interface declares all of the methods implemented by the HeavyWeight.
- The HeavyWeight provides the concrete implementations.

- The Proxy implements a simplified construction of the HeavyWeight.
- The Proxy constructs only those portions of the HeavyWeight as required by the Client.

THE PROXY PATTERN

Example:



THE PROXY PATTERN

Example: