SOEN 387 Web-based Enterprise Application Design

Stuart Thiel

Concordia University
Department of Computer & Software Engineering

Fall, 2015

Outline

Dealing with Interdependency

- A Domain Model can be complex
- Things in Domain Model can reference cyclically
- Things can even reference themselves (e.g. Person and Buddy)
- Easy to deal with in code, but makes a mess

Terms around Lazy Load

- Larman refers to getting the object when it is first accessed as “Eager Loading”
- You eager load, even if you never look at the data, just because you reference it
- Lazy Loading is when you only load data as you try to read it

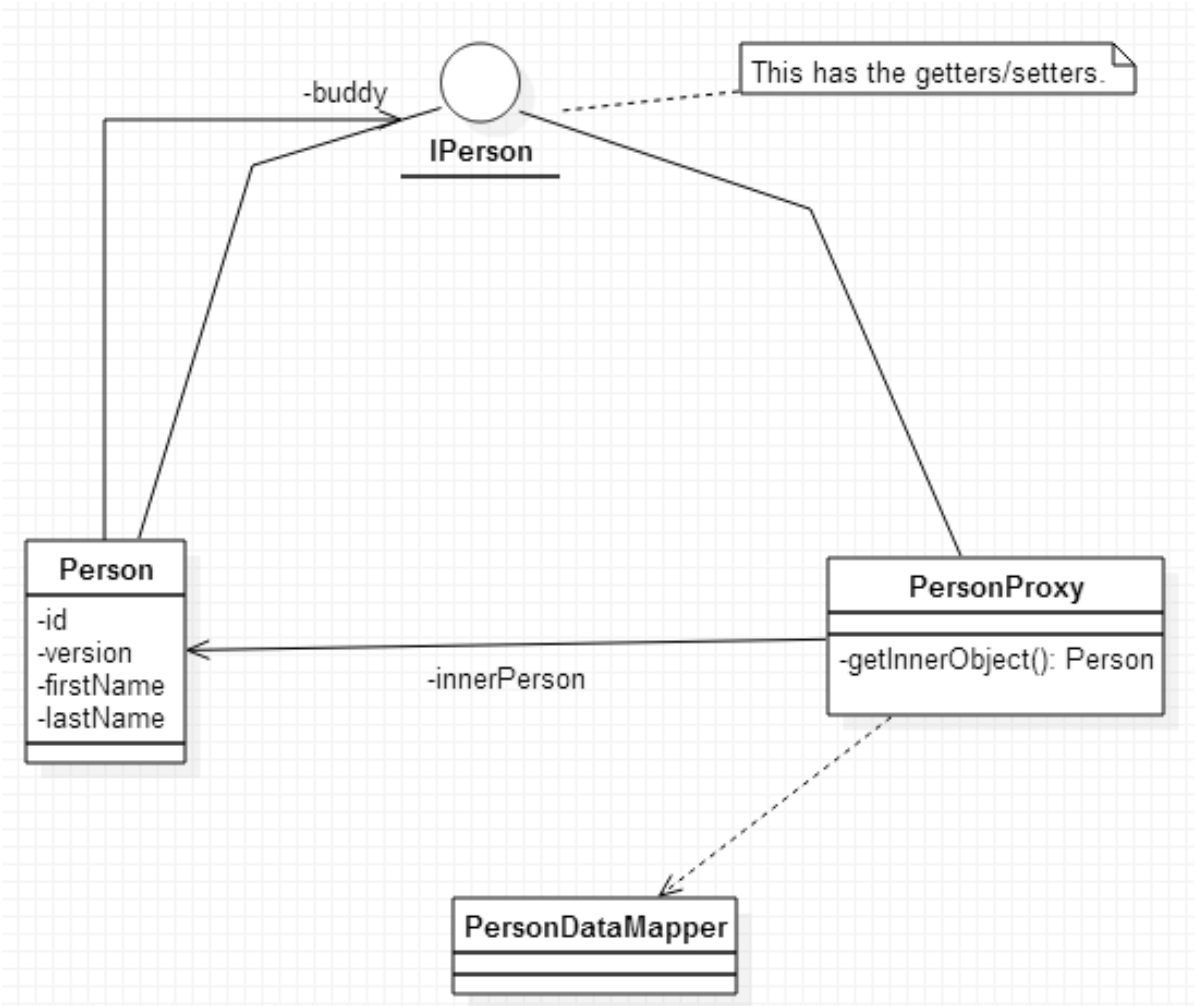
Fowler's Three Types of Lazy Loading

- Virtual Proxy
  - Needs a base class and interface
  - Needs Identity Field
  - Needs another class to load the data
- Ghost
  - Like Virtual Proxy, but loads its own data
  - So it doesn't need an interface or base class
  - Mixes POJO with Mapper, null fields
- Lazy Initialization
  - Like Virtual Proxy, but loads its own data
  - Loads it on a per-field basis...
  - So it doesn't need an interface or base class
  - Mixes POJO with Mapper, null fields, many small calls, messy

Virtual Proxy

- This solution is consistent with the rest of stuff
- The interface can actually have more uses (inheritance. . . if you have to)
- We already have the Data Mapper to do the loading behaviour
- The Virtual Proxy just needs the Identity Field on initialization

Lazy Load Diagram



Lazy Load Code

```
public class PersonProxy implements IPerson {

        private long id;
        private Person innerObject = null;

        public PersonProxy(long id) {
                this.id=id;
        }

        private Person getInnerObject() {
                if(innerObject==null) {
                        innerObject = PersonMapper.find(id);
                }
                return innerObject;
        }

        public IPerson getBuddy() {
                return getInnerObject().getBuddy();
        }

        public void setBuddy(IPerson buddy) {
                getInnerObject.setBuddy(buddy);
        }
}
```

When to use it, differs from Fowler

- Fowler says to use it based on performance needs
- Our experience says it is cleaner and simplifies cyclic dependencies
- We suggest always using it when a POJO references any other POJO
- Special treatment with lists of other POJOs. . . after midterm

Deal with it in Mappers

- We suggest to use Lazy Load whenever you reference other POJOS
- DataMapper find methods become the prime place to take the foreign keys from the db and turn them into proxies
- This is incredibly simple
- When things do go wrong, easier to isolate

Potential Problems

- Reading the same thing twice in the same request is bad
- A minor performance problem...
- Imagine changing one and writing the other to the DB? Whoops!

Reminder of Identity Map per request

- Identity Map protects within a single request
- Fowler mentions using it for a session cache. . . avoid this
- Within a request you don't need your Identity Map to deal with concurrency

How it works with Lazy Load

- When Lazy Load asks the Data Mapper
- Data Mapper Looks into Identity Map
- If it's there, use it
- If not, load it, put it in map, use it

Deal with it in Mappers

- Mappers are the obvious place to use it
- A find call can check the Identity Map before hitting DB
- And since they create the POJOs, Data Mapper can stick the newly created POJO in the Identity Map
- This implies each Data Mapper has its own Identity Map. Nobody else needs to see it
- Do not have your Proxy check the Identity Map first, let the Mapper do that

Alternate Approaches

- If DataMapper has an identity map, why keep a local copy in proxy?
- I like having the local reference, but it's a fair point.