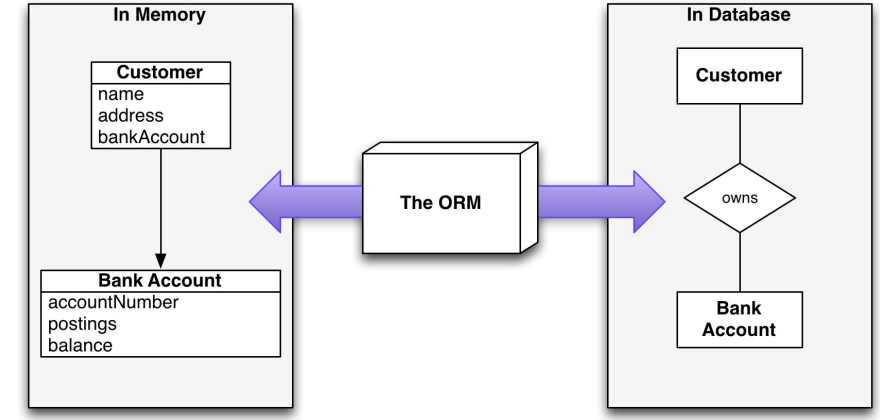

MONTANA
STATE UNIVERSITY

The O/R Impedance Mismatch

Why Some Folks Hate ORMs

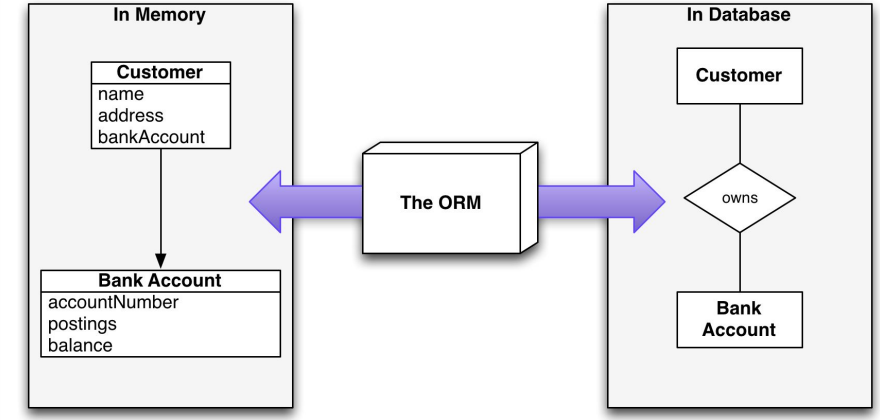
Object Relational Mapping

- As we discussed last time, the *Object/Relational Mapping* problem is the problem of mapping in memory objects to relations stored in a DBMS
- We took at look at Hibernate, a mature OR framework for java



Object Relational Mapping

- Seemed like a pretty good tool, right?
- Better than writing SQL in string literals, anyway?
- Not everyone agrees...



ORM Hate

“While I was at the QCon conference in London a couple of months ago, it seemed that every talk included some snarky remarks about Object/Relational mapping (ORM) tool” --Martin Fowler

ORM Hate

“Object/relational mapping is the Vietnam of Computer Science”. --Ted Neward

ORM Hate

“All the solutions they come up with seem to make the problem worse. I agree with Ted completely; there is no good solution to the object/relational mapping problem.”. --Jeff Atwood
https://blog.codinghorror.com/

The O/R Impedance Problem

- Why all the complaining?
- Because of a class of problems that has come to be known as

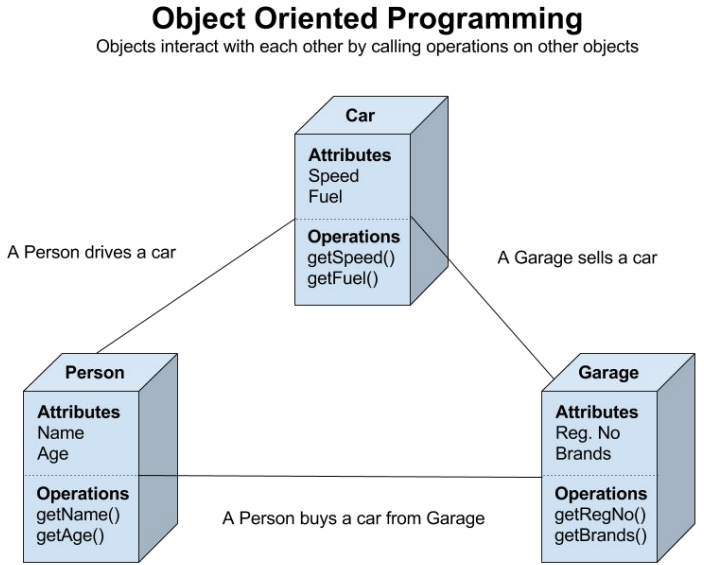
The Object/Relational Impedance Mismatch

The O/R Impedance Problem

- Objects relate to one another in a *GRAPH*
  - Object A has a pointer to Object B, and navigates it via memory references
    - This is true even in java
- Relational Schemas are *Tabular*
  - Based on the Relational Algebra

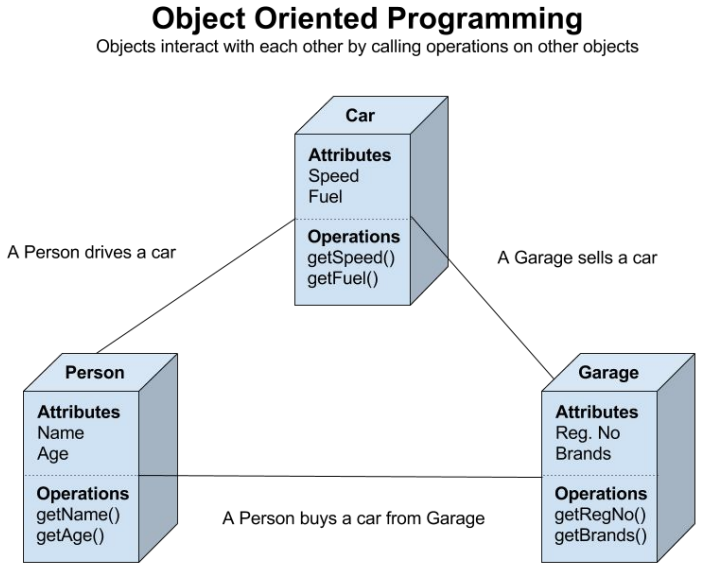
OO Concepts

- Encapsulation
  - Internal representation of data is hidden
  - *Mismatch:* Compare with a database row: all the data is “public”



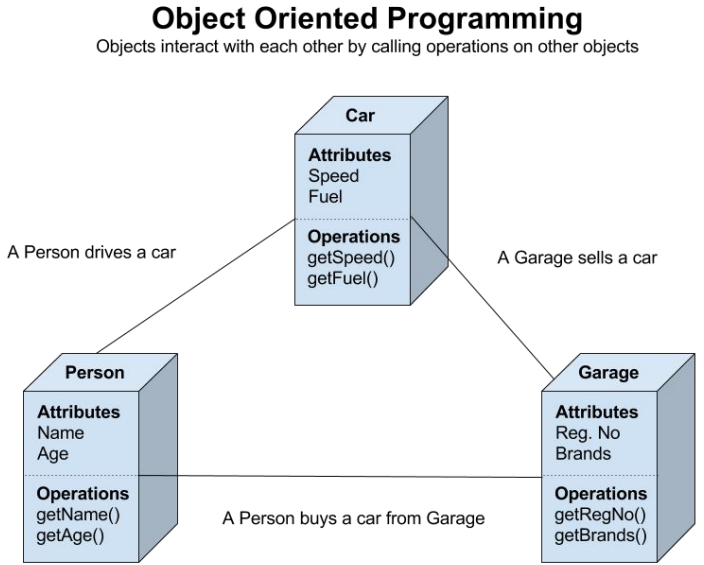
OO Concepts

- Accessibility
  - OO may have a well developed sense of accessibility
    - Public
    - Private
    - Protected
  - *Mismatch:* Compare with a database row: the data is there, or it isn't



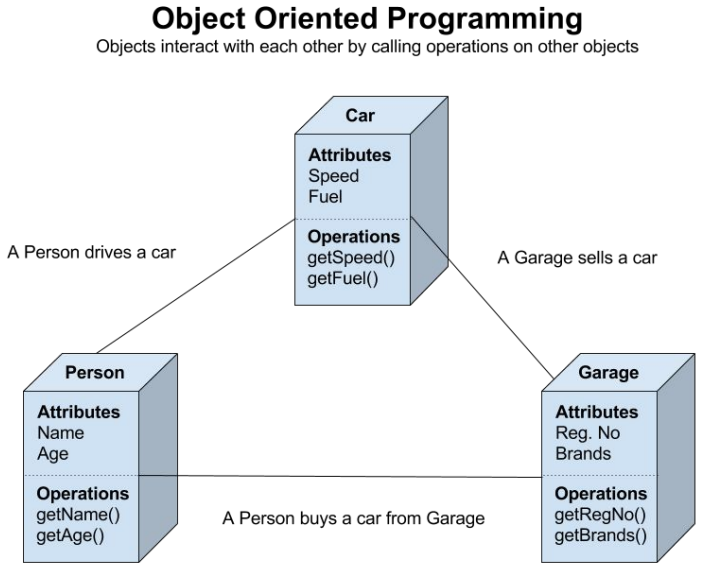
OO Concepts

- Polymorphism
  - OO has a notion of polymorphism: objects may share a base class or implement a common interface
  - *Mismatch:* No native notion of interfaces or inheritance in the relational model



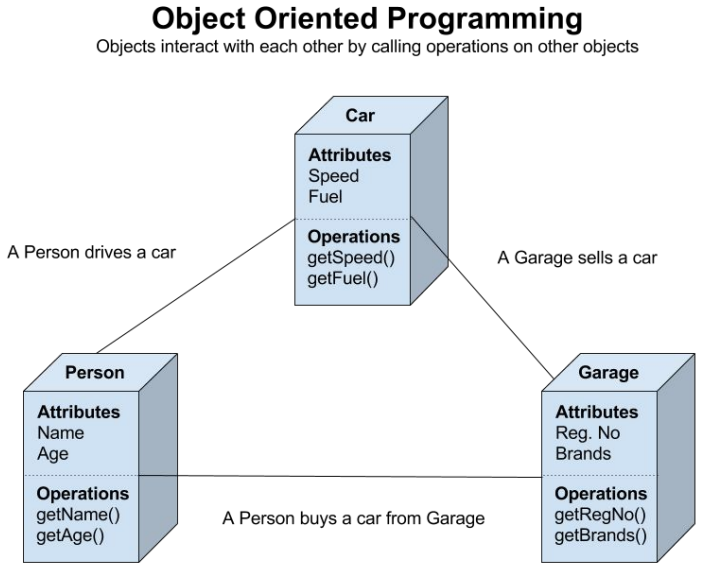
OO Concepts

- Polymorphism
  - Recall that we discussed *three mechanisms* for dealing with polymorphism at the DB level
    - Single-table inheritance - One table for all subclasses
    - Table per concrete class - One table for each concrete subclass
    - Table per class - one table for each class



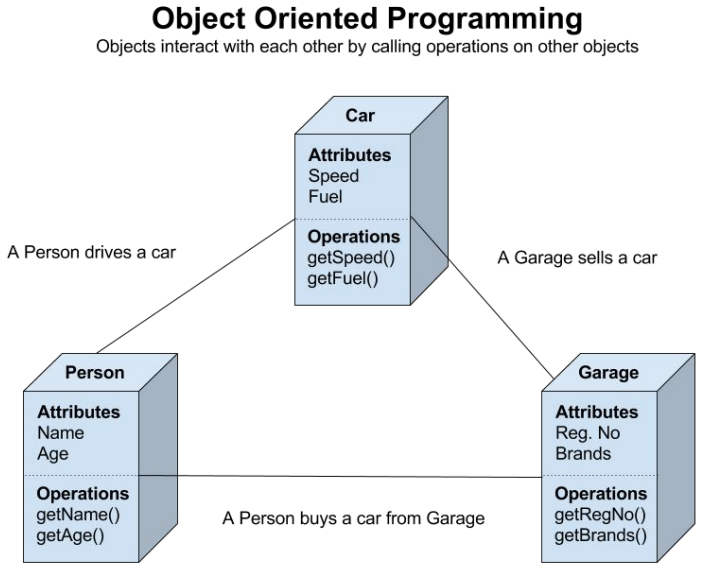
OO Concepts

- Polymorphism
  - Each mechanism had tradeoffs that had to be accepted
  - No one *canonical, strictly better* solution to the problem



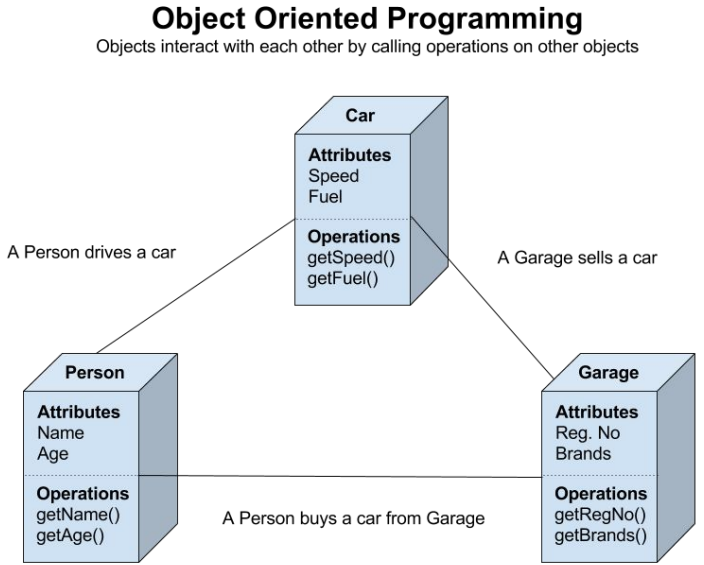
Data Type Differences

- Database and programming languages often have different notions of both types as well as type rules
  - Fixed precision numbers in a DB do not map cleanly to floating point
  - Strings may behave differently with respect to white space
  - Etc.



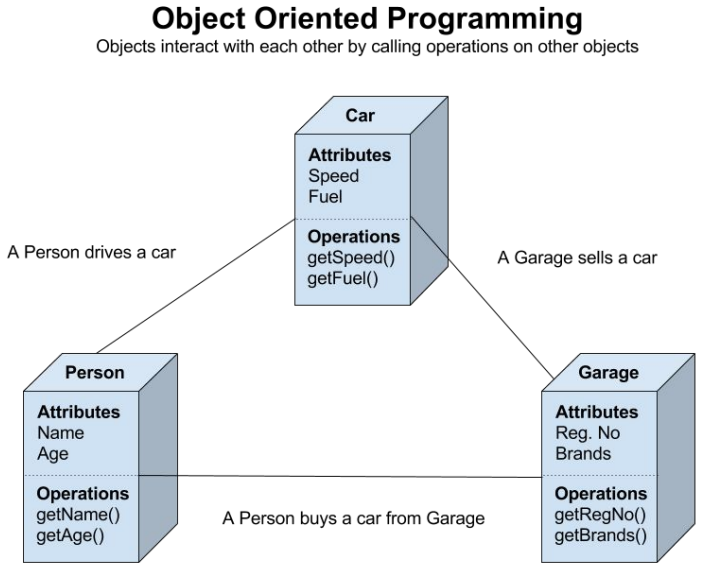
Integrity Differences

- Database and programming languages often have different notions of integrity
  - An object may allow a reference to be null for temporary reasons
  - Whereas a table may not allow null on the related relation



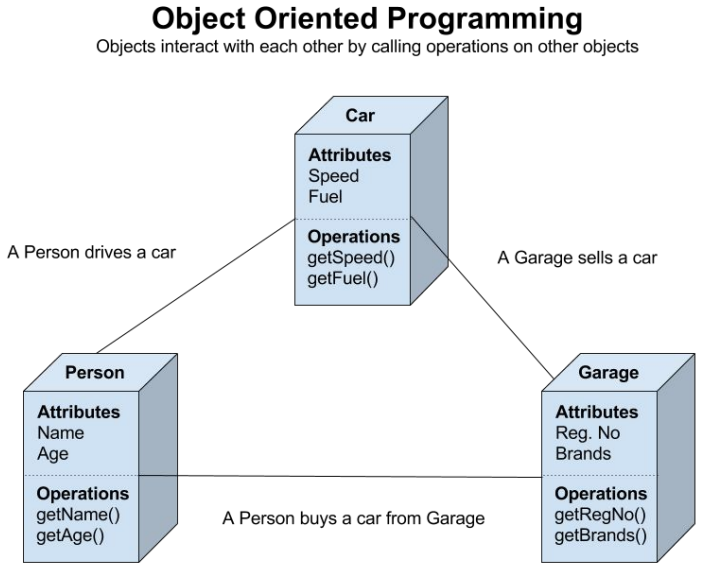
Conceptual Differences

- Databases work in SQL, which is a fundamentally *declarative* language
  - Tell the database what you want, not how to do it
- Most programming languages that work with database are *imperative*
  - Tell the program what to do, step by step



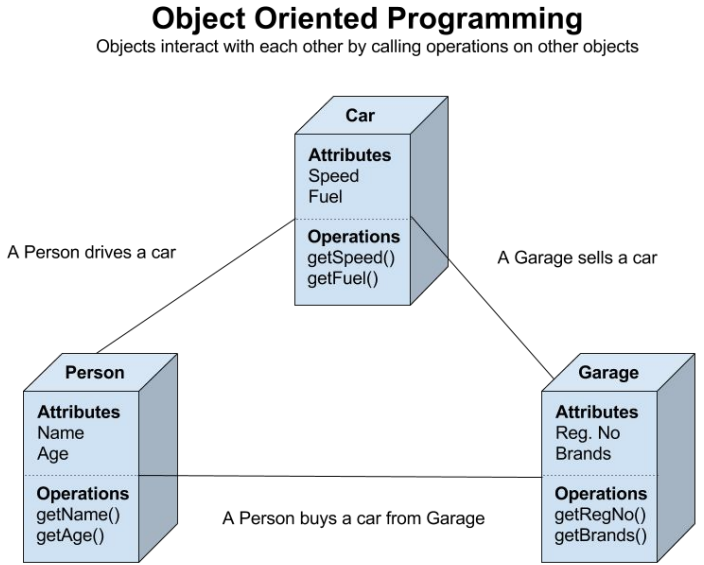
Transactional Differences

- In Databases, *Transactions*, the smallest amount of logical work can be very large
- Compare with OO programming, where a transaction is typically a single operation
  - OO languages typically have no notion of isolation or durability



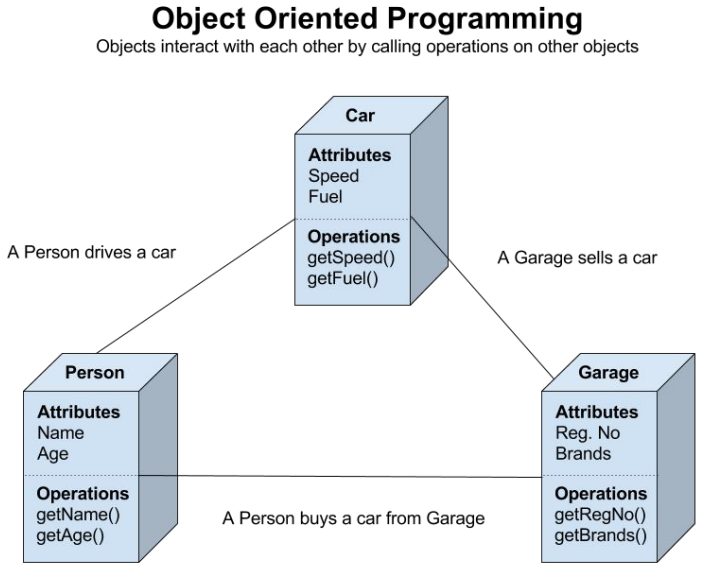
Philosophical Differences

- Declarative vs Imperative
- Nouns & Verbs
  - OO encourages a tight binding between data and actions
- Identity
  - Primary Key vs. Object Identity
- Normalization
  - Not a feature of OO
- Set vs. Graph



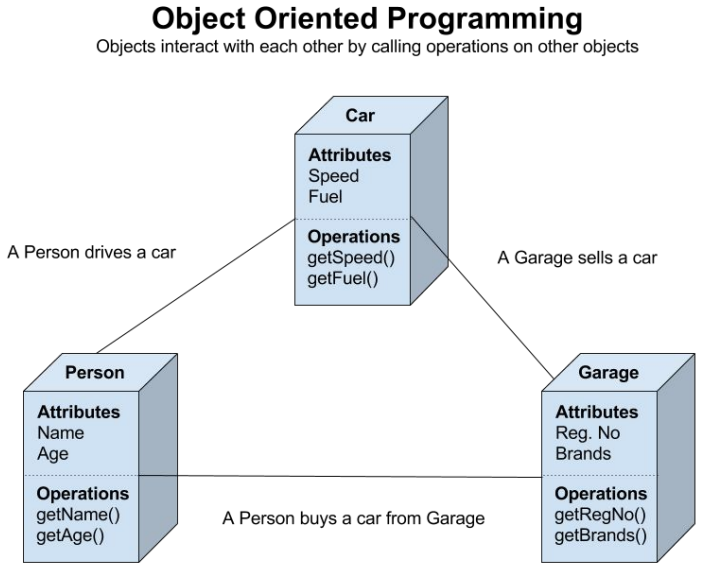
Equality

- What does equality mean in an OO system?
- What does it mean in SQL?
- If two objects have the same ID, are they equal?
  - This is a *generally* hard problem in OO



The O/R Impedance Mismatch

- All of these problems, taken together, form the O/R Impedance Mismatch problem
- How much of a problem is it?



The O/R Impedance Mismatch

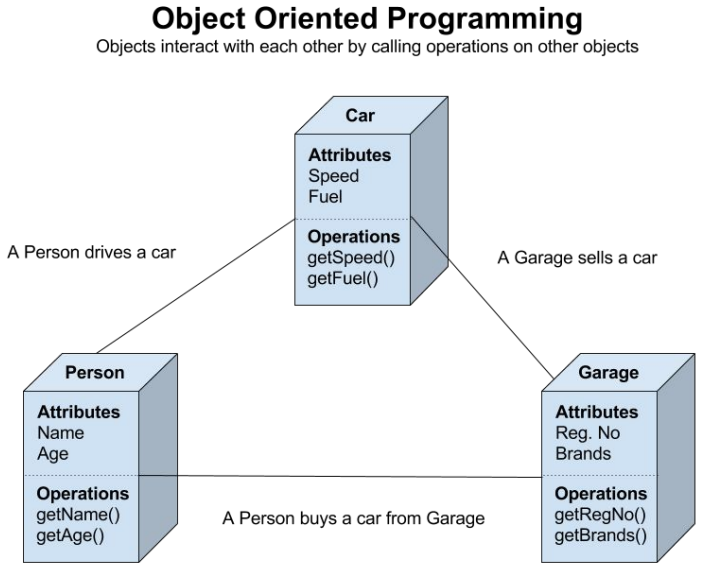
“ORMs help us deal with a very real problem for most enterprise applications. It's true they are often misused, and sometimes the underlying problem can be avoided. They aren't pretty tools, but then the problem they tackle isn't exactly cuddly either. I think they deserve a little more respect and a lot more understanding.” --Martin Fowler

The O/R Impedance Mismatch

“ I’ve come to the conclusion that, for me, ORMs are more detriment than benefit. In short, they can be used to nicely augment working with SQL in a program, but they should not replace it..” --Geoff Wozniak

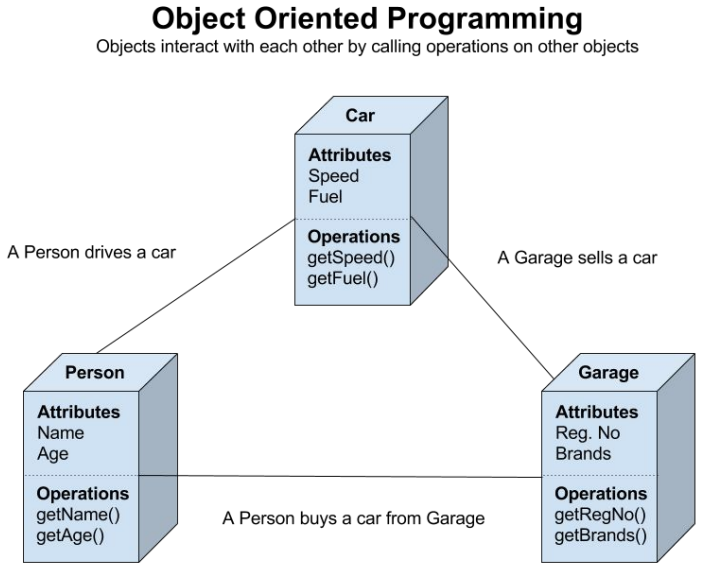
Dealing With It

- All of these problems, taken together, form the *O/R Impedance Mismatch problem*
- How much of a problem is it?



Solutions

- Many proposed solutions
  - “Better” ORMs
  - “Simpler” ORMs
  - NoSQL
  - Object-oriented database management systems



Jooq

- Jooq
  - “Database First”
  - Focuses on generating SQL
    - Not abstracting away the database
  - Generates code that represents the relational schema in Java terms

```
DSLContext create = DSL.using(conn, SQLDialect.MYSQL);
Result<Record> result = create.select().from(AUTHOR).fetch();
```

Jooq

- Jooq
  - Allows for a more straightforward mapping to the underlying DB
  - Not nearly the life-cycle functionality of other ORM systems
  - “Closer to the metal”
    - Good?
    - Bad?

```
create.select()
      .from(BOOK)
      .where(BOOK.AUTHOR_ID.eq(1))
      .and(BOOK.TITLE.eq("1984"))
      .fetch();
```

```
SELECT *
FROM BOOK
WHERE AUTHOR_ID = 1
AND TITLE = '1984'
```

LINQ (Microsoft)

- Microsoft introduced LINQ (Language Integrated Query) to .NET in 2007
- Integrated query language directly into the languages
- Can work with many different *providers*
  - LINQ to SQL works with DBMS
  - Can also work with XML documents or in memory objects

```
1  public bool IsValidUser(string userName, string passWord)
2  {
3  DBNameDataContext myDB = new DBNameDataContext();
4  var userResults = from u in myDB.Users
5                    where u.Username == userName
6                    && u.Password == passWord
7                    select u;
8  return Enumerable.Count(userResults) > 0;
9  }
```

LINQ (Microsoft)

- At right we can see that the query is *integrated* directly into the language
  - In a microsoft IDE, you get code completion, find usages, etc.
- Additionally this sort of syntax can be used against an array, or an XML document, etc.

```
1  public bool IsValidUser(string userName, string passWord)
2  {
3  DBNameDataContext myDB = new DBNameDataContext();
4  var userResults = from u in myDB.Users
5                    where u.Username == userName
6                    && u.Password == passWord
7                    select u;
8  return Enumerable.Count(userResults) > 0;
9  }
```

LINQ (Microsoft)

- Here is how you insert a user object into a database
- Note that there isn't any special syntax here compared with other O/R solutions

```
User user = new User();
user.Username = userName;
user.Password = passWord;
user.RoleId = roleId;
UserDataContext UDB = new UserDataContext();
UDB.Users.InsertOnSubmit(user);
UDB.SubmitChanges();
```

LINQ (Microsoft)

- The embedded syntax is nice
  - Really nice, actually
- But it only really solves the query problem
  - And only part of the query problem
  - What if you want to *conditionally include* some criteria?
    - E.g. our advanced search dialog

```
User user = new User();
user.Username = userName;
user.Password = passWord;
user.RoleId = roleId;
UserDataContext UDB = new UserDataContext();
UDB.Users.InsertOnSubmit(user);
UDB.SubmitChanges();
```

LINQ (Microsoft)

- Ultimately, although it is a great technical achievement and interesting programming language problem, I don’t think LINQ solves the O/R impedance mismatch

```
User user = new User();
user.Username = userName;
user.Password = passWord;
user.RoleId = roleId;
UserDataContext UDB = new UserDataContext();
UDB.Users.InsertOnSubmit(user);
UDB.SubmitChanges();
```

Active Record

- ActiveRecord
  - Originally out of the Rails community
  - Old, very mature
  - My favorite ORM

```
# return the first user named David
david = User.find_by(name: 'David')
```

```
user = User.find_by(name: 'David')
user.name = 'Dave'
user.save
```

```
user = User.find_by(name: 'David')
user.destroy
```

Active Record

- Allows validations to be declared in model classes
  - Many useful validations
    - Regexp
    - Presence
    - etc

```
class User < ApplicationRecord
  validates :name, presence: true
end

user = User.new
user.save  # => false
user.save! # => ActiveRecord::RecordInvalid:
Validation failed: Name can't be blank
```

Active Record

- Life cycle callbacks allow you to insert logic over the life cycle of the object
  - E.g. after insert, denormalize some data to another table

```
class User < ApplicationRecord
  validates :login, :email, presence: true

  before_validation :ensure_login_has_a_value

  private
    def ensure_login_has_a_value
      if login.nil?
        self.login = email unless email.blank?
      end
    end
end
```

Active Record

- Query syntax is nice
  - 90% solution for most application queries you have
  - You can bail out to SQL if necessary really easily

```
# find all users named David who are Code Artists and sort by created_at in reverse chronological order
users = User.where(name: 'David', occupation: 'Code Artist').order(created_at: :desc)
```

Active Record

- Migrations
  - Has a baked in notion of schema migrations
  - As you DB is updated, you write migrations from one state to the next
  - Your DBs can be migrated over time
    - Dev
    - Staging
    - Production

```
class CreatePublications <
ActiveRecord::Migration[5.0]
  def change
    create_table :publications do |t|
      t.string :title
      t.text :description
      t.references :publication_type
      t.integer :publisher_id
      t.string :publisher_type
      t.boolean :single_issue

      t.timestamps
    end
    add_index :publications, :publication_type_id
  end
end
```

Active Record

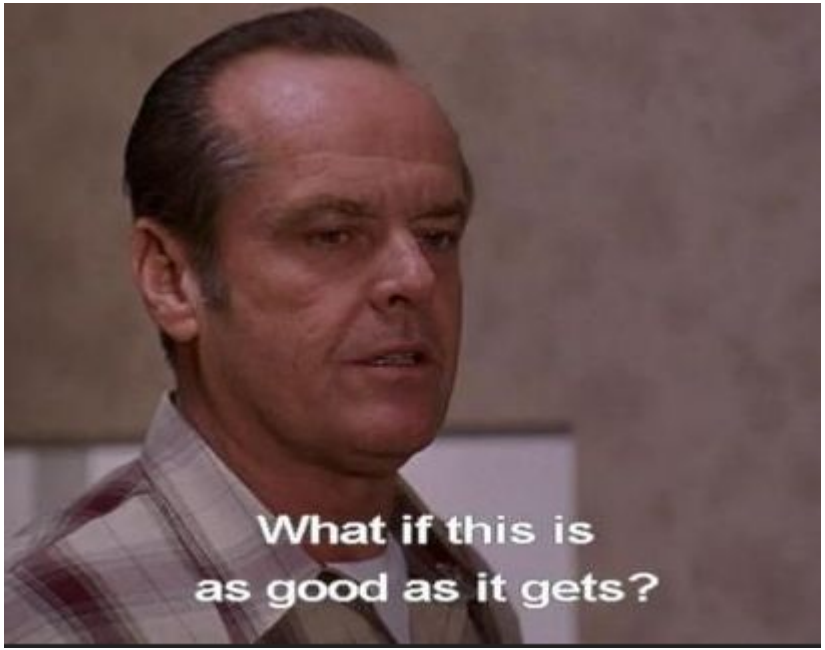
- Other niceties
  - You can get the old values for columns before you save
  - Good, explicit caching support
  - Doesn’t try to do too much, allows you to bail out to SQL when necessary

```
class CreatePublications <
ActiveRecord::Migration[5.0]
  def change
    create_table :publications do |t|
      t.string :title
      t.text :description
      t.references :publication_type
      t.integer :publisher_id
      t.string :publisher_type
      t.boolean :single_issue

      t.timestamps
    end
    add_index :publications, :publication_type_id
  end
end
```

Active Record

- My take: ActiveRecord is the best ORM solution I have worked with
  - Not perfect, but good enough
  - Gets out of the way where necessary
  - Provides a lot of value *outside* of the O/R mapping problem



The O/R Impedance Mismatch

- The O/R Impedance Mismatch: problems caused by trying to integrate the Object and Relational models with one another
- Different world views
    - Relations v. Objects
    - Declarative v. Imperative
    - Tuples v. Polymorphism
    - etc.
- Some possible solutions to the O/R problem were examined
    - Jooq
    - Linq
    - ActiveRecord

Articles Worth Reading

- The Vietnam of Computer Science - Ted Neward
  - http://blogs.tedneward.com/post/the-vietnam-of-computer-science/
- OrmHate - Martin Fowler
  - https://www.martinfowler.com/bliki/OrmHate.html
- What ORMs have taught me: just learn SQL - Geoff Wozniak
  - https://wozniak.ca/blog/2014/08/03/1/
- And many, many more...


MONTANA
STATE UNIVERSITY