Creating Models

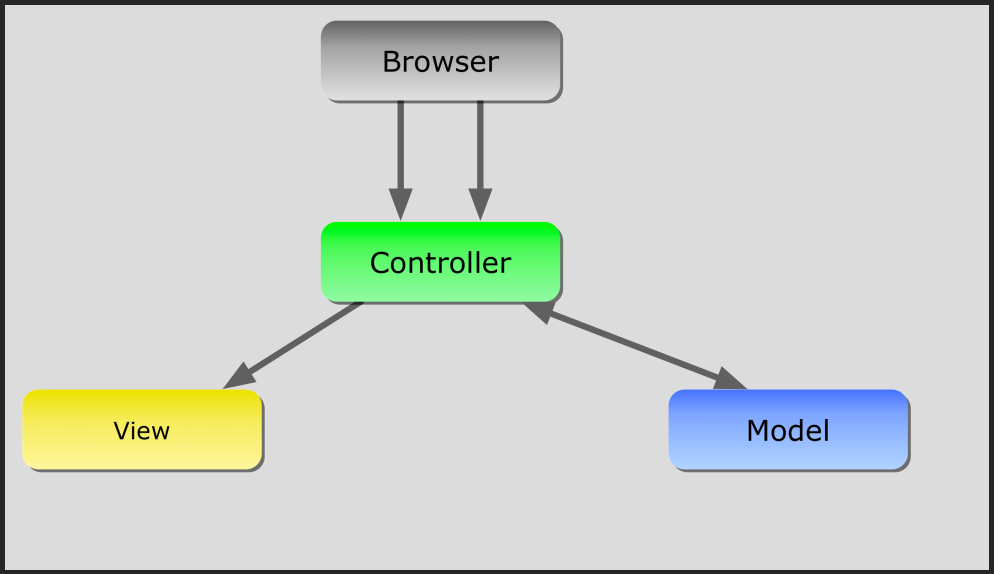
Rob Allen ~ March 2015

I make business websites

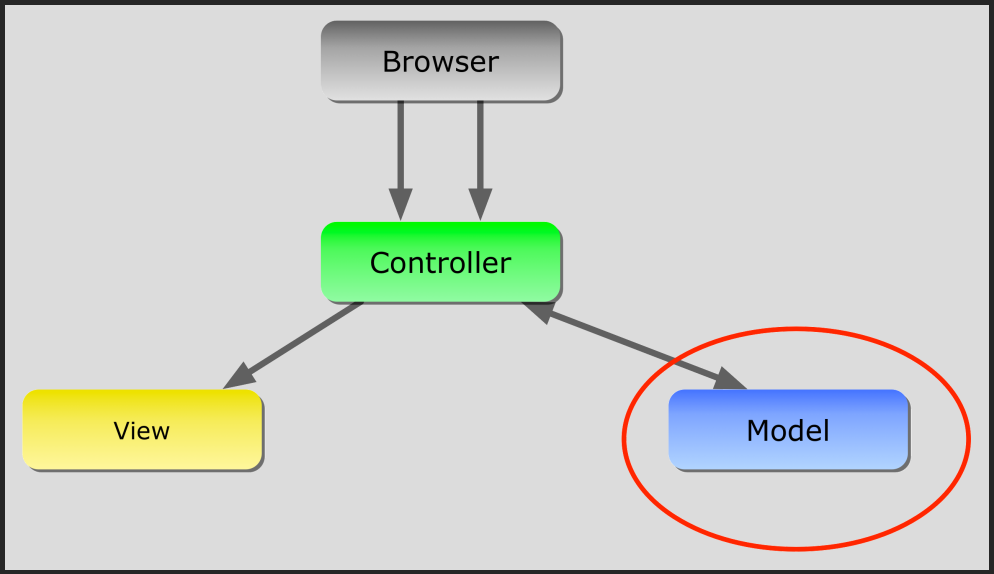
19ft.com

The business logic
is the hard part

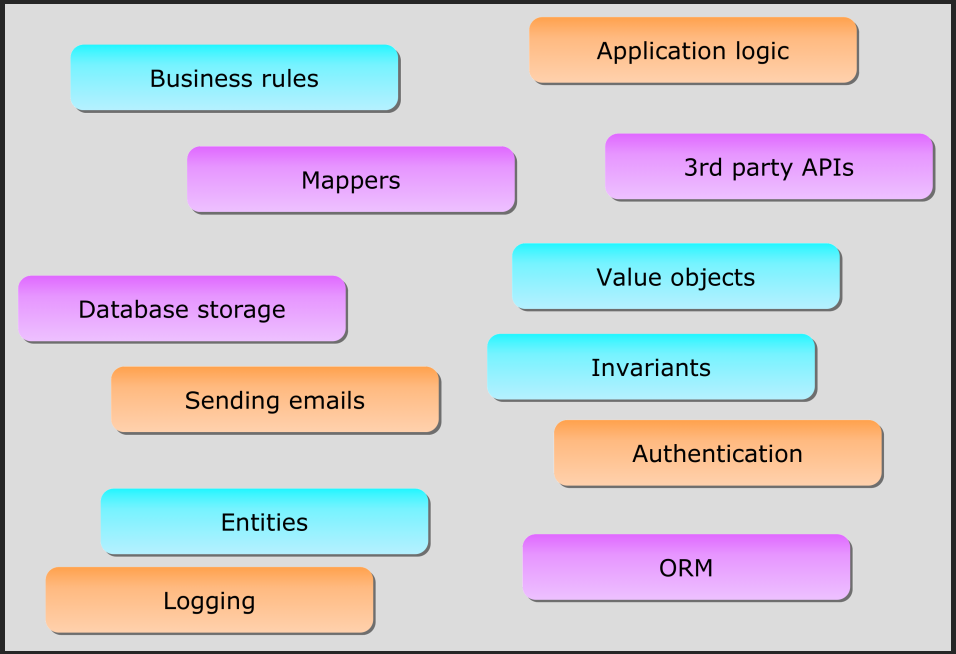
MVC

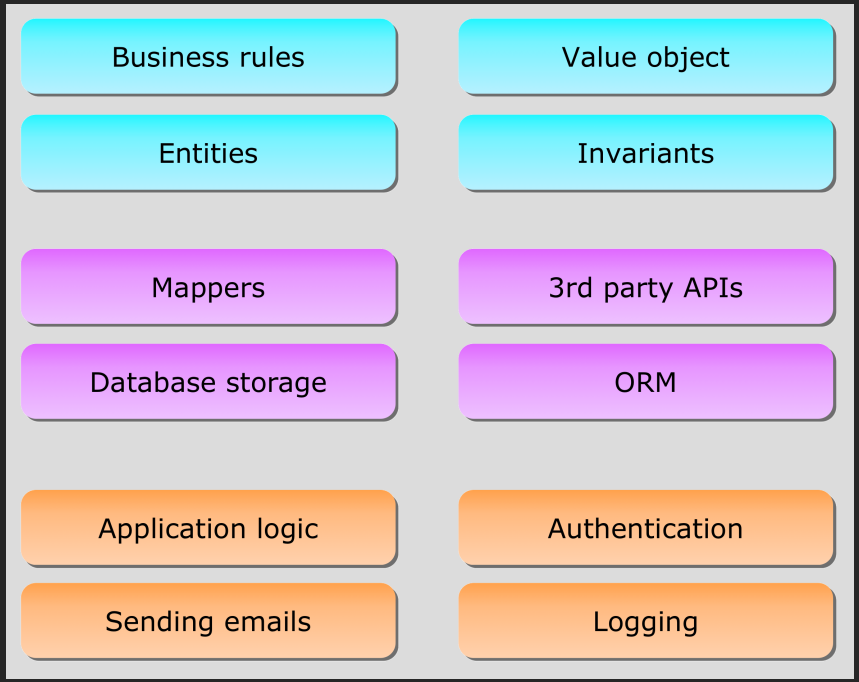


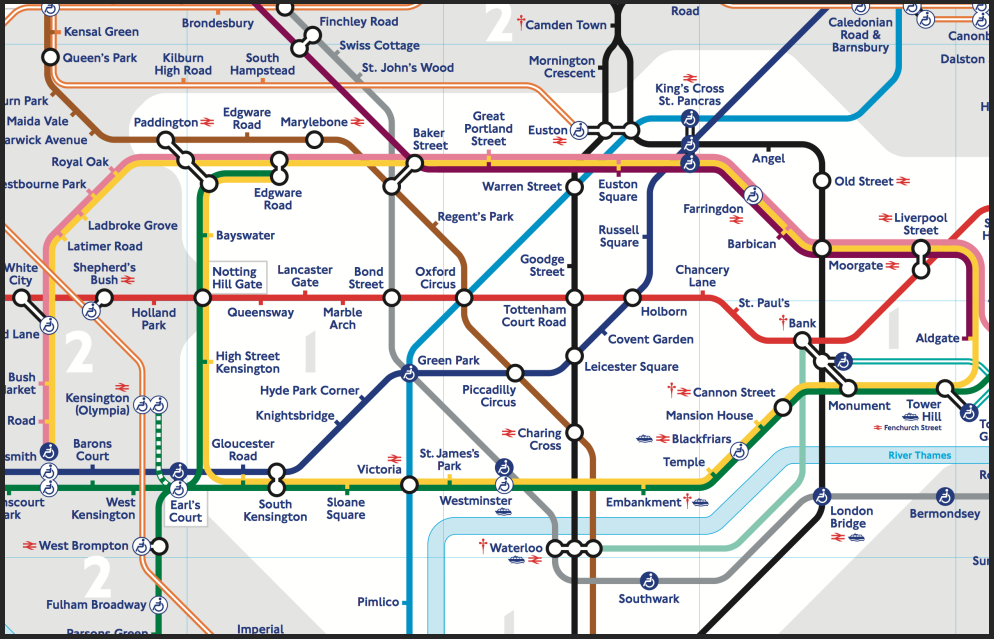
MVC



The model is the solution to a problem


Business rules
Application logic
Mappers
3rd party APIs
Value objects
Database storage
Invariants
Sending emails
Authentication
Entities
ORM
Logging


Business rules
Value object
Entities
Invariants
Mappers
3rd party APIs
Database storage
ORM
Application logic
Authentication
Sending emails
Logging


Kensal Green
Queen's Park
Kilburn High Road
South Hampstead
Brondesbury
Finchley Road
Swiss Cottage
St. John's Wood
Camden Town
Mornington Crescent
Road
Caledonian Road & Barnsbury
Canonbury
Dalston
King's Cross St. Pancras
Euston
Great Portland Street
Baker Street
Paddington
Edgware Road
Marylebone
Maida Vale
Warwick Avenue
Royal Oak
Westbourne Park
Edgware Road
Warren Street
Euston Square
Angel
Old Street
Farringdon
Liverpool Street
Barbican
Moorgate
Regent's Park
Russell Square
Ladbroke Grove
Latimer Road
Bayswater
Notting Hill Gate
Lancaster Gate
Bond Street
Oxford Circus
Goodge Street
Chancery Lane
St. Paul's
Bank
Aldgate
White City
Shepherd's Bush
Holland Park
Queensway
Marble Arch
Tottenham Court Road
Holborn
Covent Garden
High Street Kensington
Green Park
Leicester Square
Piccadilly Circus
Cannon Street
Monument
Tower Hill
Fenchurch Street
Kensington (Olympia)
Hyde Park Corner
Knightsbridge
Mansion House
Barons Court
Gloucester Road
Charing Cross
Blackfriars
Temple
River Thames
Victoria
St. James's Park
West Kensington
Earl's Court
South Kensington
Sloane Square
Westminster
Embankment
London Bridge
Bermondsey
West Brompton
Waterloo
Fulham Broadway
Pimlico
Southwark
Imperial

A problem

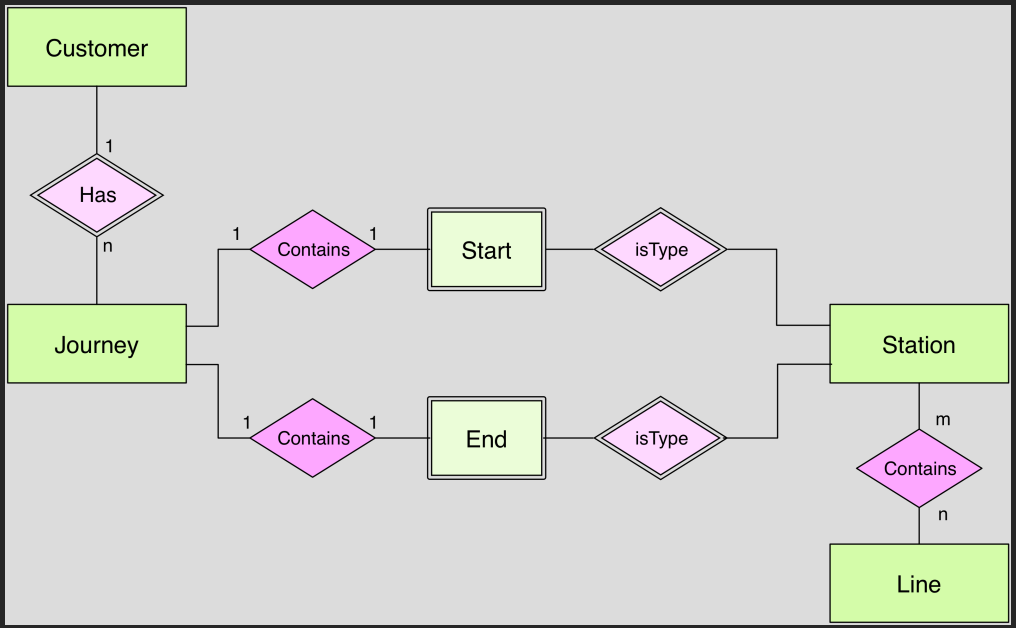
A customer wants to plan a journey between two stations.

How do we model this?

Identify the key objects

- Customer
- Journey
- Station
- Line

E-R Diagram


Customer
1
Has
n
Journey
1
Contains
1
Start
isType
Station
1
Contains
1
End
isType
m
Contains
n
Line

Entities represent things in the problem-space

Entity

- Means something to the customer
- An object defined primarily by its identity
- Mutable
- Persisted
- Has a life cycle

Identity

- The identity of an object is what it means to the customer
- Unique within the scope of the domain

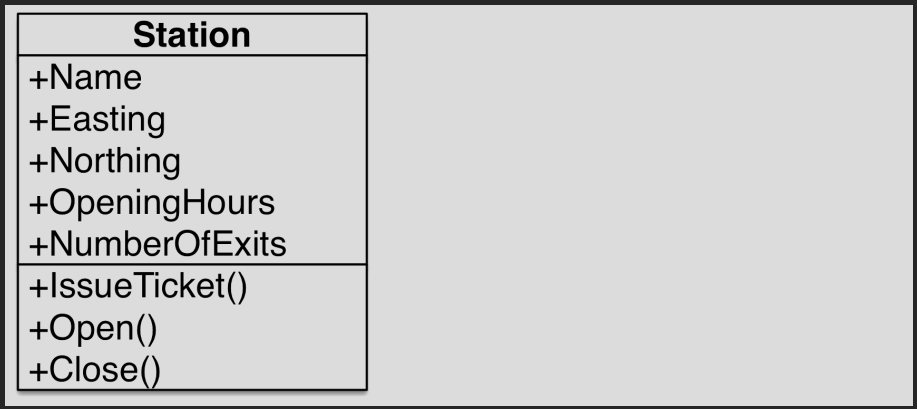
For a tube station, this is it's *name*, not its database id.

My customer is never going to talk to me about station 43, they are going to say "Euston Square".

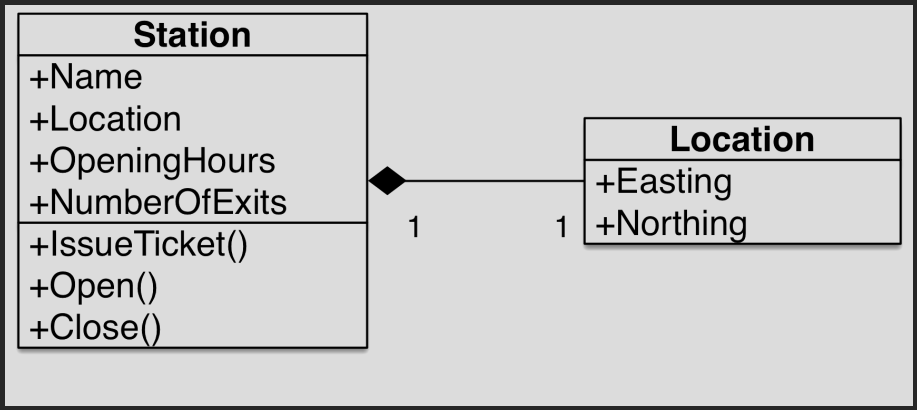
Value objects

- Defined primarily by its attributes
- Immutable
- Simple!
- Do not exist without an entity

A station has a location



A station has a location



Domain services

If a SERVICE were devised to make appropriate debits and credits for a funds transfer, that capability would belong in the domain layer.

Eric Evans

Domain services

- We map the business processes to services
- Represents behaviour in the domain
- A service does not have internal state
- Usually a point of connection for many objects

Let’s look at some code

Some code

```
class Journey {
  function getStart() {}
  function setStart(Station $start) {}

  function getStop() {}
  function setStop(Station $stop) {}

  function setRoute() {}
  function getRoute() {}
}

class RoutingService {
  function route(Station $start, Station $stop) {}
}
```

Anaemic domain model

When you look at the behavior, and you realize that there is hardly any behavior on these objects, making them little more than bags of getters and setters.

Instead there are a set of service objects which capture all the domain logic.

Martin Fowler

Entity with behaviour

```
class Journey {
  function getStart() {}
  function setStart(Station $start) {}

  function getStop() {}
  function setStop(Station $stop) {}

  function route() {}
}
```

What happens if route() is complex?

Double dispatch

The entity calls the helper domain service, passing a reference to itself.

```
// Helper service
class JourneyRouter {
  function route(Journey $journey) {}
}

// Journey class
function route() {
  $router = new JourneyRouter();
  $this->route = $router->route($this);
}
```

Persistence

Persistence options

A simple domain model can use ActiveRecord/TDG;
a complex one will require mapping.

I don’t really care what you choose!

I lied.

Don’t use ActiveRecord!

It integrates the database code into your domain model

Table Data Gateway

- Operates on a single database table
- Contains all the SQL for accessing the table
- Doesn’t know anything about the entity.
- Simple to implement

Table Data Gateway

```
class JourneyGateway {
  function __construct($dsn, $username, $password) {}

  function find($id) {}
  function findForStartingStation($stationId) {}

  function insert($startId, $stopId) {}
  function update($id, $startId, $stopId) {}
}
```

Data Mapper

- Class to transfer data from objects to the database and back.
- Entity aware
- Isolates the domain model from the database
- Not limited to a single table

Data Mapper

```
class JourneyMapper {
  function __construct($dsn, $username, $password) {}

  function find($id) {}
  function findForStartingStation($stationId) {}
  public function save(Journey $journey) {}
}
```

Increased scope: ORM

Data mappers can be limited in scope to an entity or generic enough to work with full object graphs.

This is known as Object Relational Mapping

Persistence layer is more complicated:

- Identity map to hold loaded objects
- Storage of entire object graphs to the database
- Unit of Work to track changed objects for saving

If you need this, then use a pre-written ORM library!

Web services

- The persistence storage could be a web service.
- Data mappers work really well

Integrating our model into the application

The service layer*

> *It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down.*
>
> Eric Evans

* (Known as *application layer* in DDD)

Service layer

We can sub-divide:

- Application services
- Infrastructural services

Application services

If the banking application can convert and export our transactions into a spreadsheet file for us to analyze, this is an application SERVICE.

Eric Evans

Infrastructural services

A bank might have an application that sends out an email to a customer when an account balance falls below a specific threshold. The interface that encapsulates the email system, and perhaps alternate means of notification, is a SERVICE in the infrastructure layer.

Eric Evans

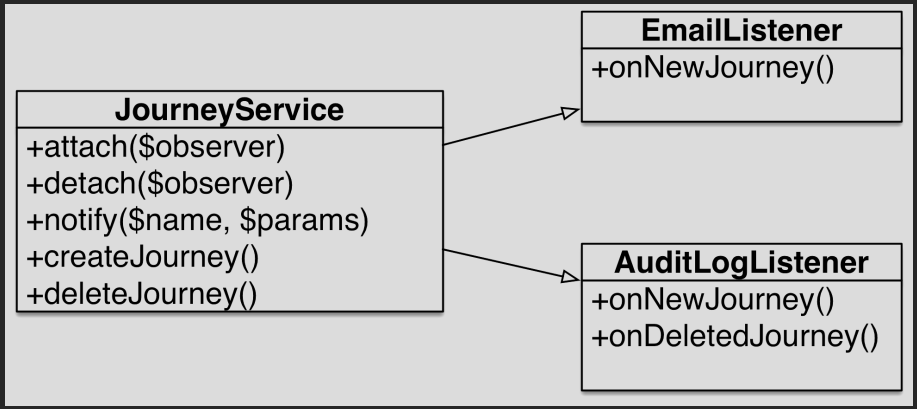
Application service

```
class JourneyService {
  function createJourney($customer, $start, $stop)
  {
    $journey = $customer->createJourney($start,$stop);
    $journey->route();

    $this->entityManager->flush();
    $this->mailer->newJourneyNofication($journey);
    $this->auditor->log('newJourney', $journey);
  }
}
```

Beware the Fat service

Decouple using Observer pattern



CQRS: Separating reading and writing

CQRS

Command Query Responsibility Segregation.

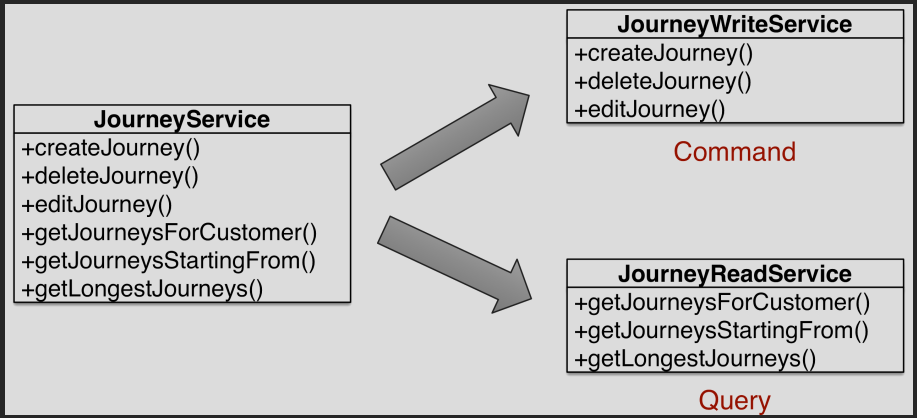
- *Commands* change data
- *Queries* read data

Most useful when:

- Separate hardware
- Optimise performance

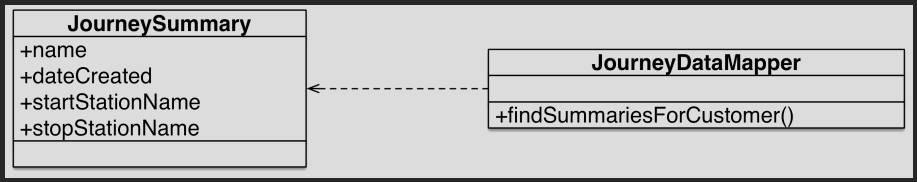
CQRS at its most basic

Two services where there was one



CQRS

One useful case is a summary object for a read-only list



Final point

The success of a design is not necessarily marked by its longevity. It is the nature of software to change.

Eric Evans

To sum up

Entity:

Object with identity that do stuff

Value object:

Immutable with no identity

Domain service:

Behaviours that don’t belong to an entity

To sum up

Mappers / Repository:

Transfer your model to and from persistence

Application services:

Isolate your domain model from your controllers

Infrastructure services:

Support services for your application

Thank you!

Rob Allen - http://akrabat.com - @akrabat