Creating Models

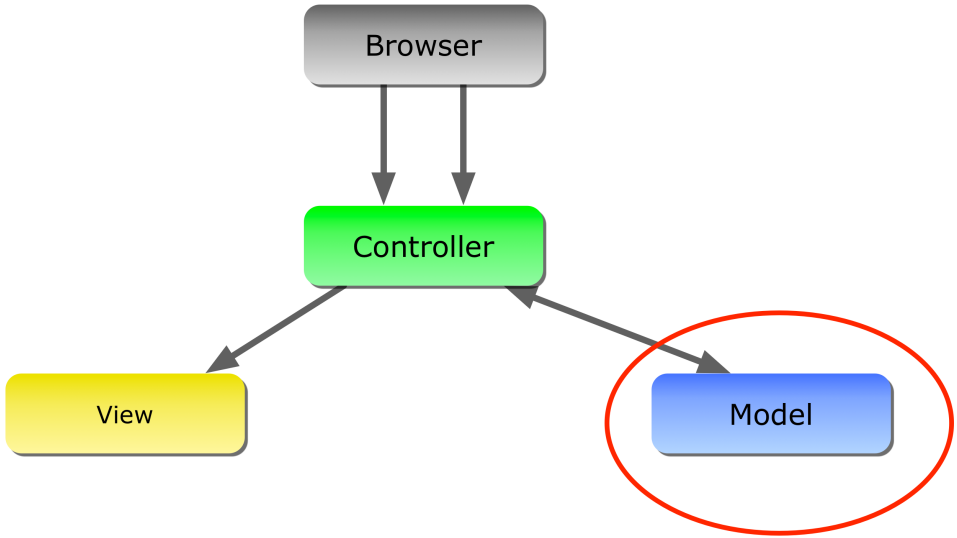
Rob Allen, June 2014

I make business websites

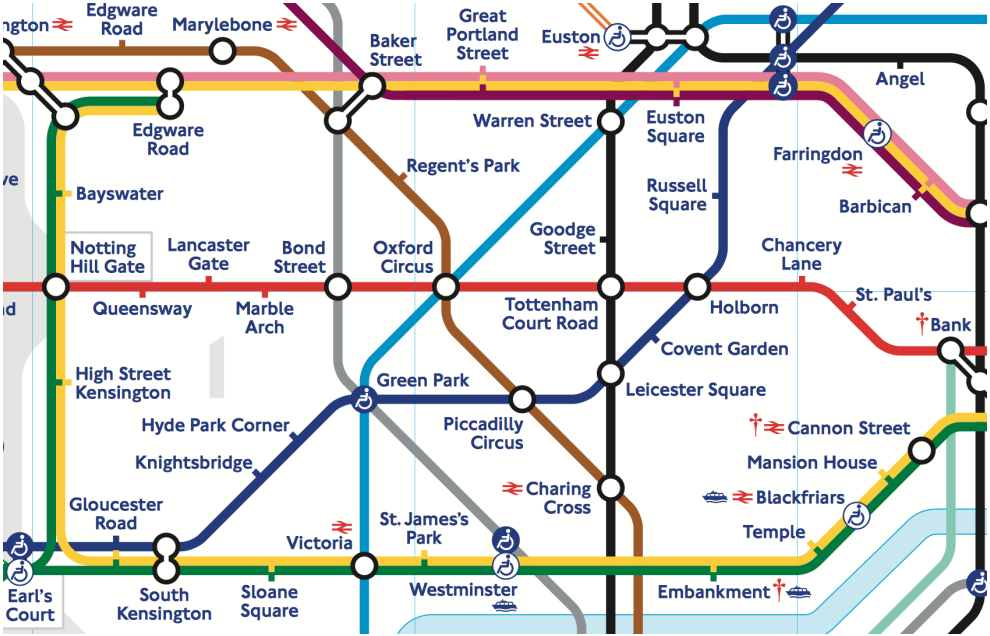
19ft.com

Business logic is hard

MVC



The model is the solution to a problem


Edgware Road
Marylebone
Baker Street
Great Portland Street
Euston
Angel
Edgware Road
Warren Street
Euston Square
Farringdon
Regent's Park
Bayswater
Russell Square
Barbican
Notting Hill Gate
Lancaster Gate
Bond Street
Oxford Circus
Goodge Street
Chancery Lane
Queensway
Marble Arch
Tottenham Court Road
Holborn
St. Paul's
Bank
Covent Garden
High Street Kensington
Green Park
Leicester Square
Hyde Park Corner
Piccadilly Circus
Cannon Street
Knightsbridge
Mansion House
Charing Cross
Gloucester Road
Victoria
St. James's Park
Blackfriars
Temple
Earl's Court
South Kensington
Sloane Square
Westminster
Embankment

A problem

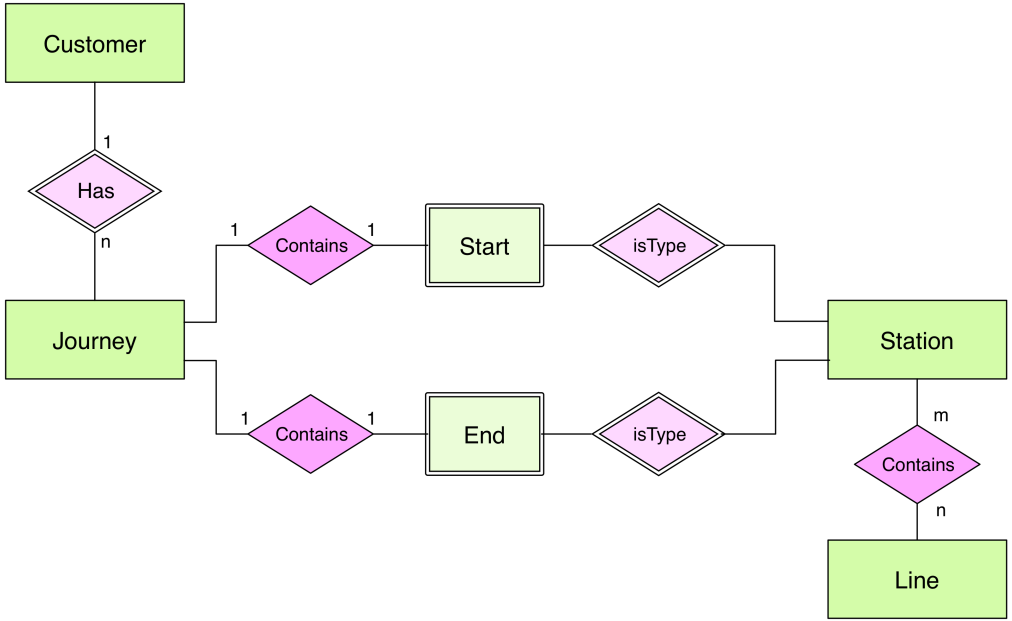
A customer wants to plan a journey between two stations.

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

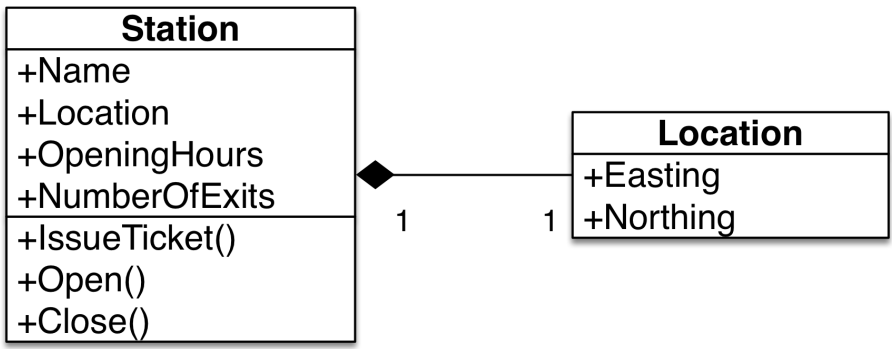
For a tube station, this is it's *name*, not its database id.

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



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 RoutePlannerService {
  function planRoute(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 planRoute() {}
}
```

What happens if planRoute() is complex?

Double dispatch

Private class for use by the entity, which passes a reference to itself.

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

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

Works with full object graphs:

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

Use a pre-written ORM library!

Not just RDBMS!

Persistence could be to:

- NoSQL
- Web services
- Flat files

Use the same techniques.

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

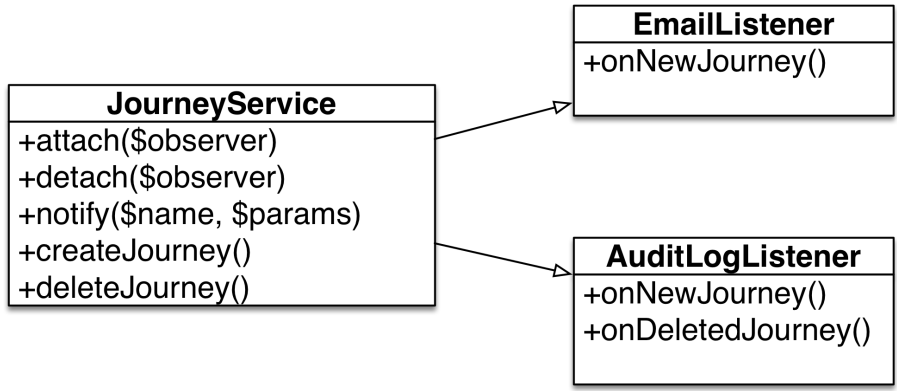
Application service

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

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

Beware the Fat service

Decouple using Observer pattern



Final point

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

Eric Evans

Thank you!

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