Unit tests

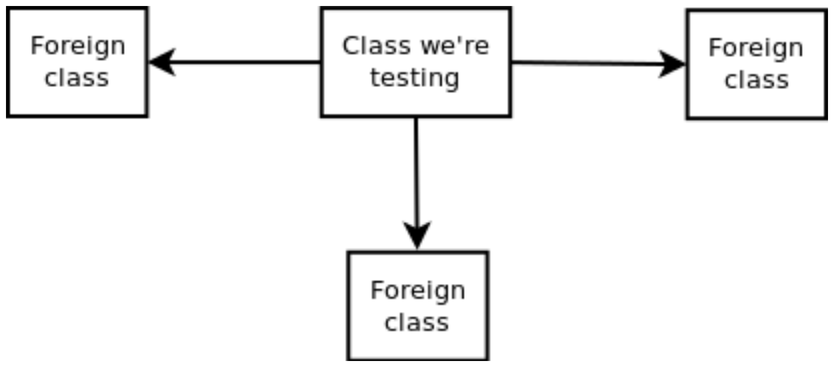
Unit testing for Javascript

Outline

- Concepts
  - Unit testing
  - Testable code
  - Dependency Injection
- AngularJS
  - Karma-runner
  - Principles
  - Todo app testing
- Groovy testing

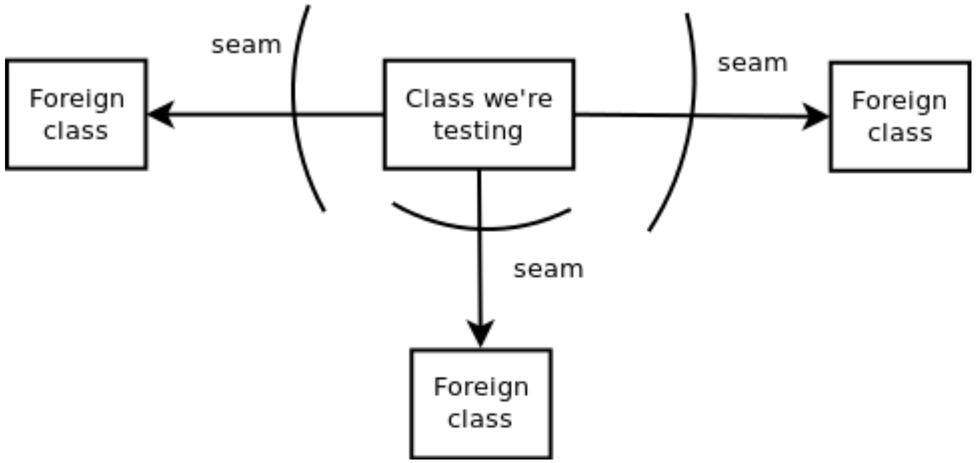
Concepts: Unit Testing

- Testing a small unit of code
    - .. a class
    - .. a method
    - .. a static method



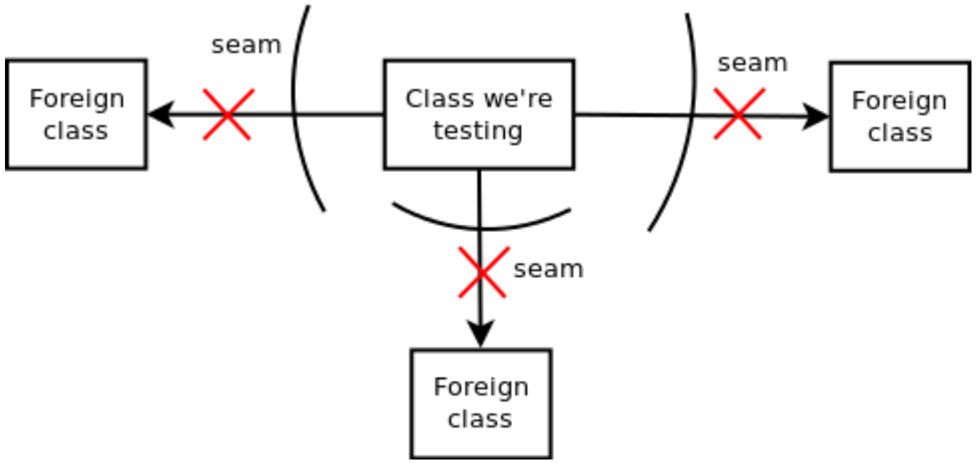
Concepts: Unit Testing

- Your enemy; "the seam"



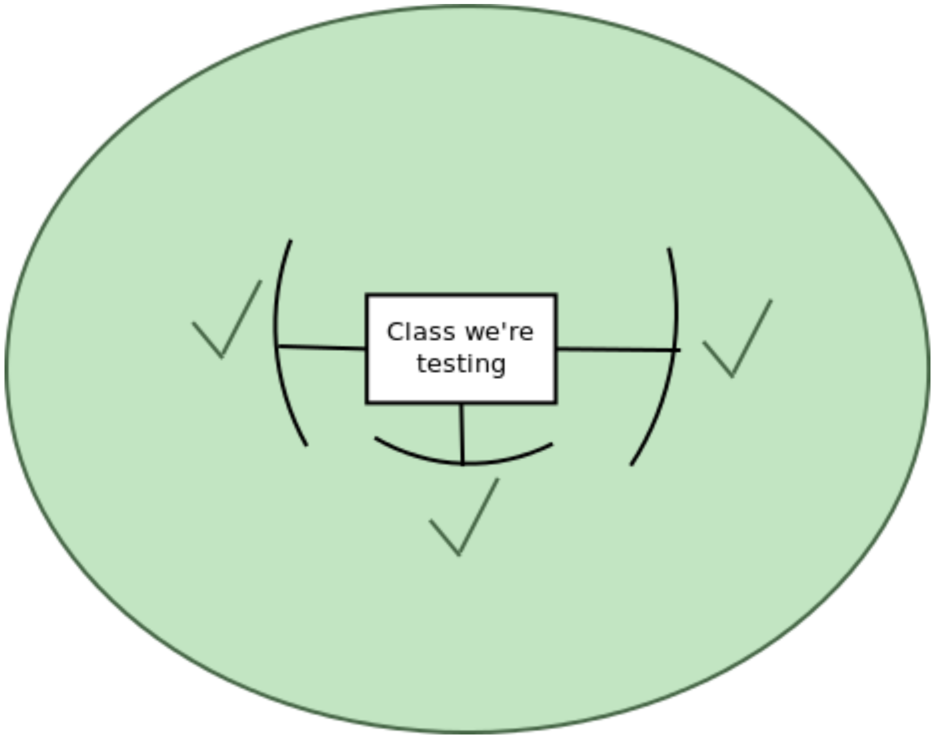
Concepts: Unit Testing

- Your enemy; "the seam"



Concepts: Unit Testing

- Your friend; "a test's warm embrace"



Concepts: Testable code

- The class seam

```
class Register {

        Receipt receipt;
        float gst;

        public Register(float gst) {
                this.receipt = new Receipt();
                this.gst = gst;
        }

        public void addProduct(String productName, int priceInCents) {
                Calculator calculator = new Calculator();
                float inDollars = priceInCents * 0.01f;
                float total = calculator.withTax(inDollars, gst);
                this.receipt.addToList(
                                new ReceiptElement(productName, total)
                        );
        }

        public float totalPriceInDollars() {
                float sum = 0.0f;
                for (ReceiptElement el :
                        this.receipt.getElementStore().all())
                {
                                sum += el.price();
                }
                return sum;
        }

}
```

Concepts: Testable code

- The class seam

```
class Register {

	Receipt receipt;
	float gst;

	public Register(float gst) {
		this.receipt = new Receipt();
		this.gst = gst;
	}

	public void addProduct(String productName, int priceInCents) {
		Calculator calculator = new Calculator();
		float inDollars = priceInCents * 0.01f;
		float total = calculator.withTax(inDollars, gst);
		this.receipt.addToList(
				new ReceiptElement(productName, total)
			);
	}

	public float totalPriceInDollars() {
		float sum = 0.0f;
		for (ReceiptElement el :
			this.receipt.getElementStore().all())
		{
				sum += el.price();
		}
		return sum;
	}

}
```

Concepts: Testable code

- The method seam

```
public void addProduct(String productName, int priceInCents) {
     Calculator calculator = new Calculator();
     float inDollars = priceInCents * 0.01f;
     float total = calculator.withTax(inDollars, gst);
     this.receipt.addToList(
               new ReceiptElement(productName, total)
          );
}
```

Concepts: Testable code

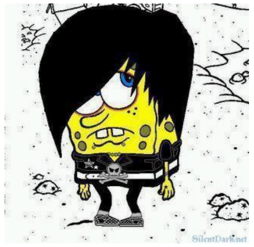
- The method seam

```
public void addProduct(String productName, int priceInCents) {
     Calculator calculator = new Calculator();
     float inDollars = priceInCents * 0.01f;
     float total = calculator.withTax(inDollars, gst);
     this.receipt.addToList(
               new ReceiptElement(productName, total)
          );
}
```

Concepts: Testable code

- Hard to test code
  - signals that your code probably is hard to test
    - .. using the `new`-keyword
    - .. knowing about internals of friend's implementation
    - .. unwarranted use of static calls

Concepts: Testable code



- Easy to test code is emo
  - classes shouldn't care about their friends
    - use interfaces and interface implementations
    - inject interfaces into constructor
    - .. or using a dependency injection framework

Concepts: Testable code

- Easy to test code is local
    - when a method needs to cross over to a different class, extract that code into a separate method

```
public float totalPriceInDollars() {
     float sum = 0.0f;
     for (ReceiptElement el : this.receipt.getElementStore().all()) {
          sum += el.price();
     }
     return sum;
}
```

Concepts: Testable code

- Easy to test code is local
  - when a method needs to cross over to a different class, extract that code into a separate method

```
public float totalPriceInDollars() {
     float sum = 0.0f;
     for (ReceiptElement el : this.getReceiptElements()) {
          sum += el.price();
     }
     return sum;
}

// now overridable in unit-test
protected List<ReceiptElement> getReceiptElements() {
     return this.receipt.getElementStore().all();
}
```

Concepts: Testable code

- Before

```
class Register {

       Receipt receipt;
       float gst;

       public Register(float gst) {
              this.receipt = new Receipt();
              this.gst = gst;
       }

       public void addProduct(String productName, int priceInCents) {
              Calculator calculator = new Calculator();
              float inDollars = priceInCents * 0.01f;
              float total = calculator.withTax(inDollars, gst);
              this.receipt.addToList(
                            new ReceiptElement(productName, total)
                     );
       }

       public float totalPriceInDollars() {
              float sum = 0.0f;
              for (ReceiptElement el :
                     this.receipt.getElementStore().all())
              {
                            sum += el.price();
              }
              return sum;
       }

}
```

Concepts: Testable code

- After

```
class Register {

        Receipt receipt;
        Calculator calculator;
        float gst;

        public Register(Receipt receipt, Calculator calculator, float gst) {
                this.receipt = receipt;
                this.calculator = calculator;
                this.gst = gst;
        }

        public void addProduct(String productName, int priceInCents) {
                float inDollars = priceInCents * 0.01f;
                float total = this.calculateTax(inDollars, gst);
                ReceiptElement receiptElement =
                        new ReceiptElement(productName, total);
                this.addToReceipt(receiptElement);
        }

        public float totalPriceInDollars() {
                float sum = 0.0f;
                for (ReceiptElement el : this.getReceiptElements()) {
                        sum += el.price();
                }
                return sum;
        }

        // now overridable in unit-test
        protected float calculateTax(float amount, float gst) {
                return this.calculator.withTax(amount, gst);
        }

        // now overridable in unit-test
        protected List<ReceiptElement> getReceiptElements() {
                return this.receipt.getElementStore().all();
        }

        // now overridable in unit-test
        protected void addToReceipt(ReceiptElement receiptElement) {
                this.receipt.addToList(receiptElement);
        }

}
```

Concepts: Dependency Injection

- Dependency Injection
  - external party takes care of who is your friend
  - several flavours:
    - constructor arguments
    - internal overrideable instantiations
    - after init configuration

Concepts: Dependency Injection

- Sample implementations

AngularJS: karma

- Configuration
  - Karma
    keeps track of your changes and automatically runs tests
  - Istanbul
    performs Javascript code coverage analysis
  - Bamboo/Sonar
    build server / static analysis server
  - PhantomJS
    headless javascript execution on Bamboo

AngularJS: Karma

- Installing Karma
  - install NPM (nodejs package manager)

    https://npmjs.org/

```
$ npm install -g karma
```

AngularJS: Istanbul

- Installing Istanbul

```
$ npm install -g istanbul
```

- Running Istanbul

```
$ istanbul cover **/*.js
$ istanbul report
```

AngularJS: Unit-testing

- First setup karma

```
$ karma init
```

AngularJS: Unit-testing

- First setup karma

```
// list of files / patterns to load in the browser
files = [
  JASMINE,
  JASMINE_ADAPTER,
  'libs/*.js',
  'libs/test/angular-mocks.js',
  'angular/common.js',
  'angular/services/*.js',
  'angular/directives/*.js',
  'angular/controllers/*.js',
  'angular/app.js',
  'test/**/*.js'
];
```

AngularJS: Unit-testing

- First setup karma

```
$ karma run
```

AngularJS: Unit-testing

- Principles
  - Isolate the code you want to test
  - Use Angular's dependency injection
  - Set scope, trigger action, assert results
  - Mock and REST services separation

AngularJS: Unit-testing

- Isolate the code you want to test
    - your code should be testable
    - be very strict in what concept does what and how they interact

      (services, directives, controllers)

AngularJS: Unit-testing

- Use Angular's dependency injection

show the code (testBreadcrumbs.js)

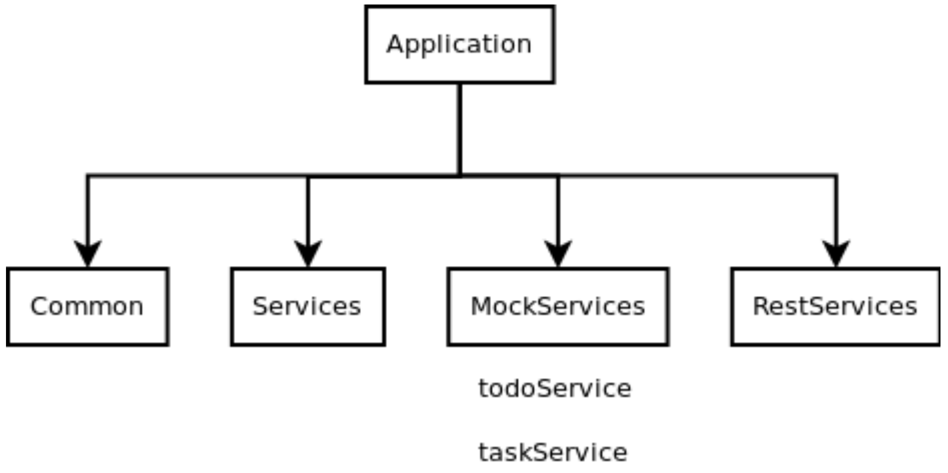
AngularJS: Unit-testing

- Set scope, trigger action, assert results

show the code (testAgencyDashboardController.js)

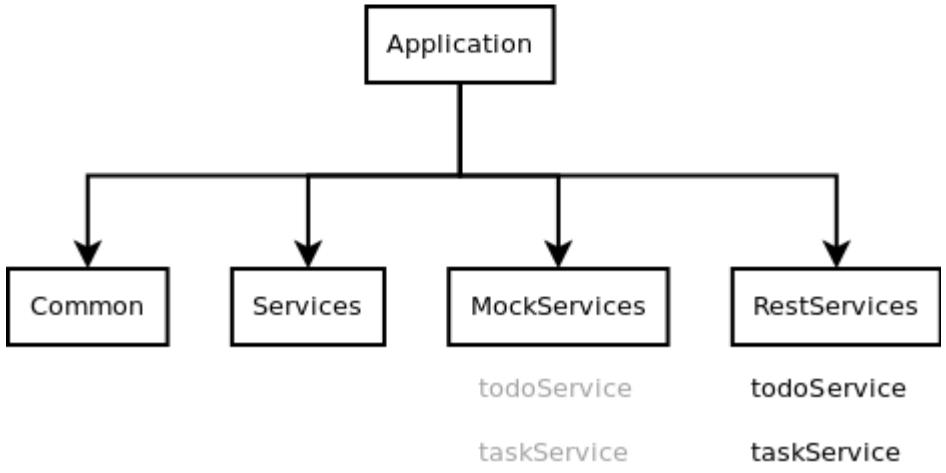
AngularJS: Unit-testing

- Mock and REST services separation



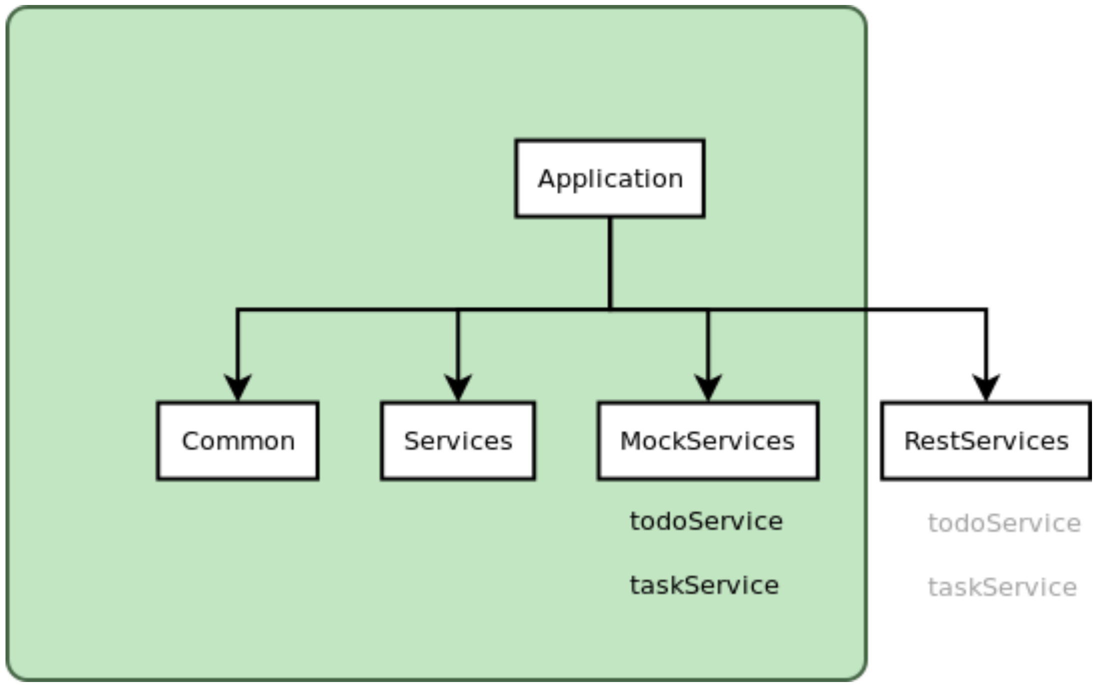
AngularJS: Unit-testing

- Mock and REST services separation



AngularJS: Unit-testing

- Mock and REST services separation



AngularJS: Unit-testing

- Some recipes
  - describe
  - it should ..
  - it should with inject ..
  - directive describe

AngularJS: Unit-testing

- Testing our TODO example

show the code