Junit-contracts: A Contract Testing Tool

Claude N. Warren, Jr.

FOSDEM '16

January 30-31, 2016

Brussels, Belgium

Who Is Claude Warren?

- claude@xenei.com
- Apache Jena Project Management Committee Member and Committer.
- https://github.com/Claudenw
- Playing with java since version 0.8
- Developer/Architect > 25 years experience
- Currently employed by IBM (Galway, IE)
- Formerly employed by Digital Enterprise Research Institute (Galway, IE), National Renewable Energy Laboratory (Golden, CO, USA)
- Founding member of the Denver Area Mad Scientists Club
- Winner of the first Critter Crunch (Robotics Competition)
- Frustrated Musician
- Author of Junit Contract test extension.

What is Contract Testing

- A Java interface outlines a contract.

What is Contract Testing

- A Java interface outlines a contract.
- The contract is further refined and defined in other documentation.

What is Contract Testing

- A Java interface outlines a contract.
- The contract is further refined and defined in other documentation.
- Contract testing ensures that all testable facets of the contract are tested.

What is Contract Testing

- A Java interface outlines a contract.
- The contract is further refined and defined in other documentation.
- Contract testing ensures that all testable facets of the contract are tested.
- A jUnit extension written by Claude Warren and found at https://github.com/Claudenw/junit-contracts

Why Contract Testing

- Verify that implementations are correct.
  - Support can ask for proof of correctness of 2nd or 3rd party implementations.
  - Internal development teams can ensure that they are correctly implementing the interface long before integration test.
- Apply DRY (Don't Repeat Yourself) principles to interface testing. One test covers all implementations.
  - In Java a class can have multiple interfaces but only one parent, so consistent testing across interface implementations is difficult.

Who Benefits

- SPI/API Implementers – Insure full implementation.
- SPI/API Definers – Insure that other teams correctly implement contracts.
- QA Test – can easily validate that contracts are correctly implemented.
- QA Test Managers – can easily determine which contract tests need to be developed or implemented.

Problem

```
public interface Foo {
  // Add an object.  Implementations must log action.
  public void add( Object x );
  Public boolean contains( Object x );
  // register a logger to listen.  Multiple calls ok.
  public void register( Logger log );
}
```

Problem

```
public interface Foo {
  // Add an object.  Implementations must log action.
  public void add( Object x );
  Public boolean contains( Object x );
  // register a logger to listen.  Multiple calls ok.
  public void register( Logger log );
}
```

Problem

```
public interface Foo {
  // Add an object.  Implementations must log action.
  public void add( Object x );
  Public boolean contains( Object x );
  // register a logger to listen.  Multiple calls ok.
  public void register( Logger log );
}
```

How do you verify that all implementations adhere to the logging requirement?

Problem

```
public interface Foo {
  // Add an object.  Implementations must log action.
  public void add( Object x );
  Public boolean contains( Object x );
  // register a logger to listen.  Multiple calls ok.
  public void register( Logger log );
}
```

How do you verify that all implementations adhere to the logging requirement?

How do you find all the implementations?

Solution

- Define a “producer” that provides the instance of the interface.

Solution

- Define a “producer” that provides the instance of the interface.
- Define a concrete contract test that tests the instance returned by a “producer”

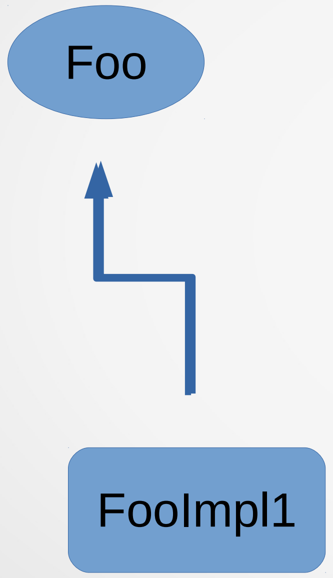
Solution

- Define a “producer” that provides the instance of the interface.
- Define a concrete contract test that tests the instance returned by a “producer”
- Define a jUnit extension that locates all the contract tests, associates them with the interface they test and locates all the classes that implement that interface.

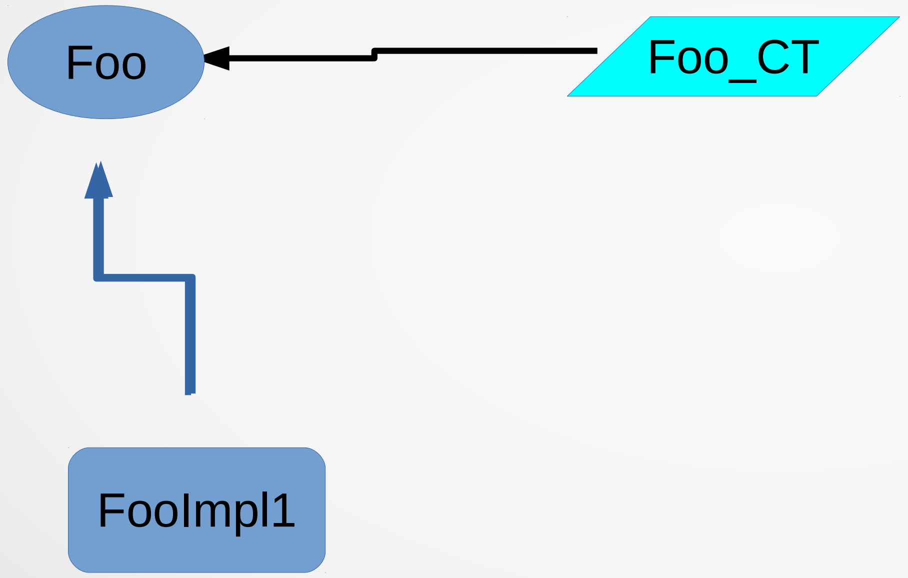
Producer Interface

```
public interface IProducer<T>() {
   public T newInstance();
   public void cleanUp();
};
```

Solution Diagram

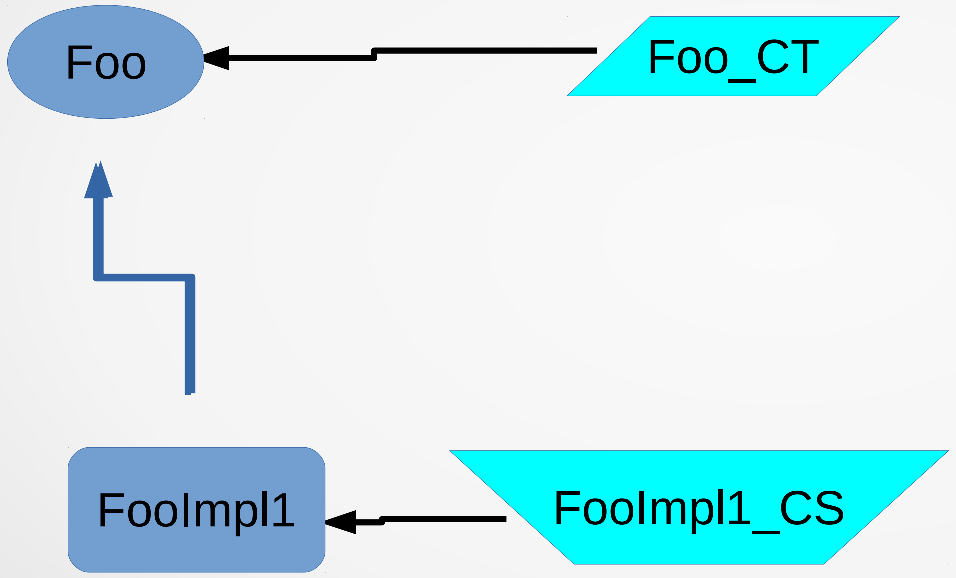


Solution Diagram



@Contract(Foo.class)

Solution Diagram

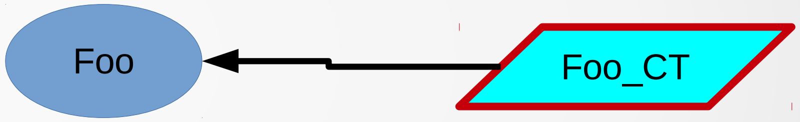


@Contract(Foo.class)

@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)

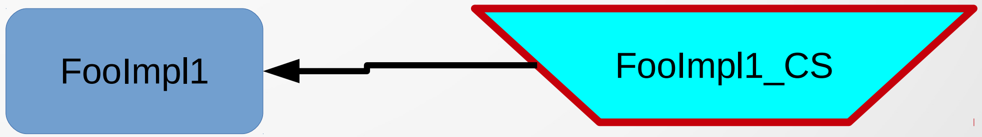
@Contract

```
@Contract(Foo.class)
public class Foo_CT<T extends Foo>  {

  private IProducer<T> fooProducer;

  @Contract.Inject
  public final void setFooContractTestProducer(IProducer<T> fooProducer) {
    this.fooProducer = fooProducer;
  }

  @ContractTest
  public void testAdd() {
    TestingLogger logger = …
    Object testObject = …
    Foo foo = fooProducer.newInstance();
    foo.register( logger );
    foo.add( testObject );
    assertTrue( logger.recordedAdd() );
    assertTrue( foo.contains( testObject ) );
  }

  @After
  public void cleanup() {
    fooProducer.cleanUp();
  }
…
}
```

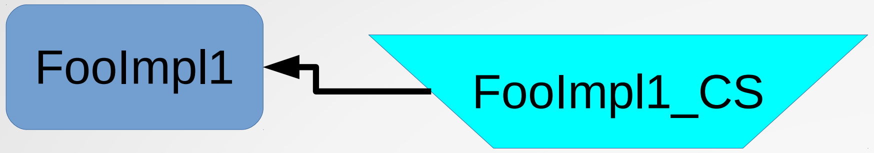


@ContractImpl(FooImpl1.class)

```
@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)
public class FooImpl1_CS {

  private IProducer<FooImpl1> fooProducer;

  public FooImpl1_CS() {

    fooProducer = new IProducer<FooImpl1>() {

      @Override
      public FooImpl1 newInstance() {
        return new FooImpl1();
      }

      @Override
      public void cleanUp() {
        // nothing to do
      }

    };
  }

  @Contract.Inject
  public final IProducer<FooImpl1> getTestProducer() {
    return fooProducer;
  }
}
```



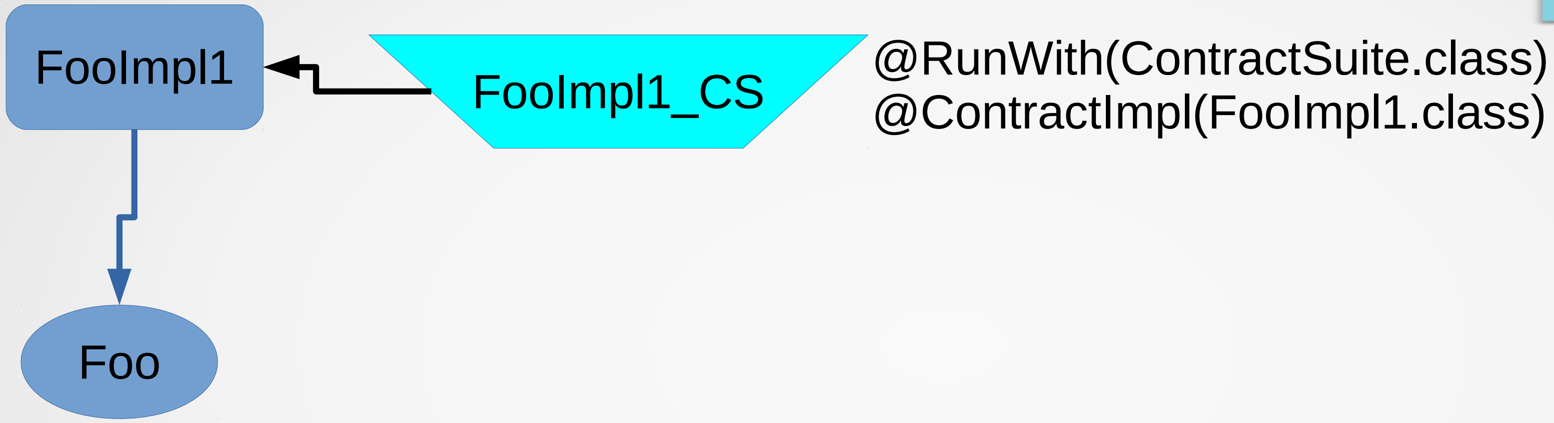
Runtime Result



@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)

1. Find class specified in ContractImpl

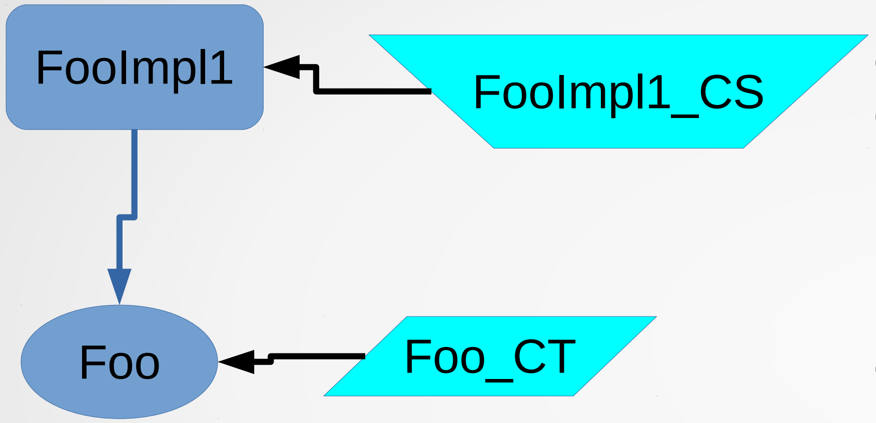
Runtime Result



@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)

1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.

Runtime Result

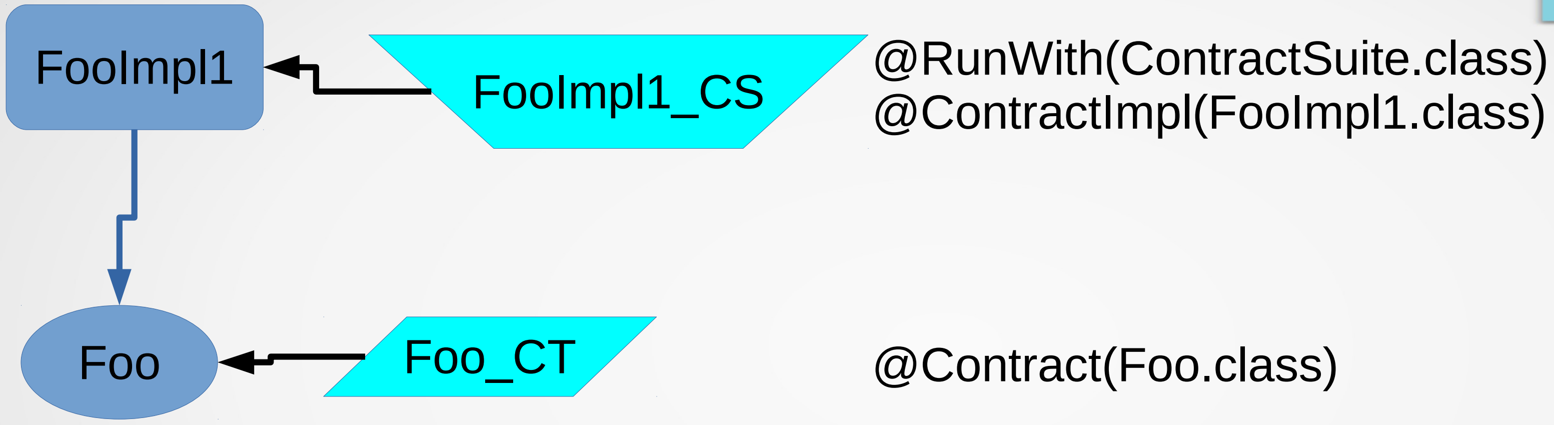


@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)

@Contract(Foo.class)

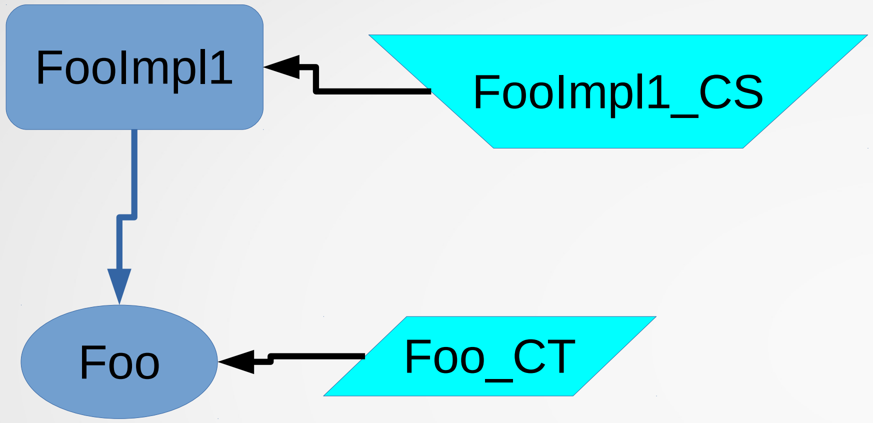
1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.
3. Find all classes annotated with Contract and which test an ancestor interface.

Runtime Result



1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.
3. Find all classes annotated with Contract and which test an ancestor interface.
4. Instantiate each class found in step 3.

Runtime Result

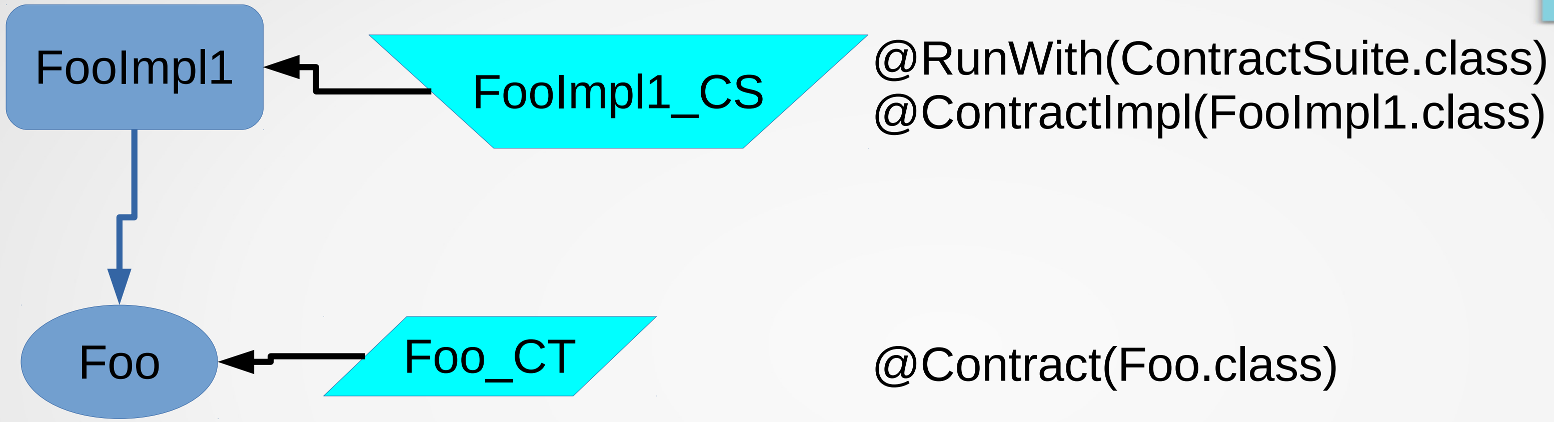


```
@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)
```

```
@Contract(Foo.class)
```

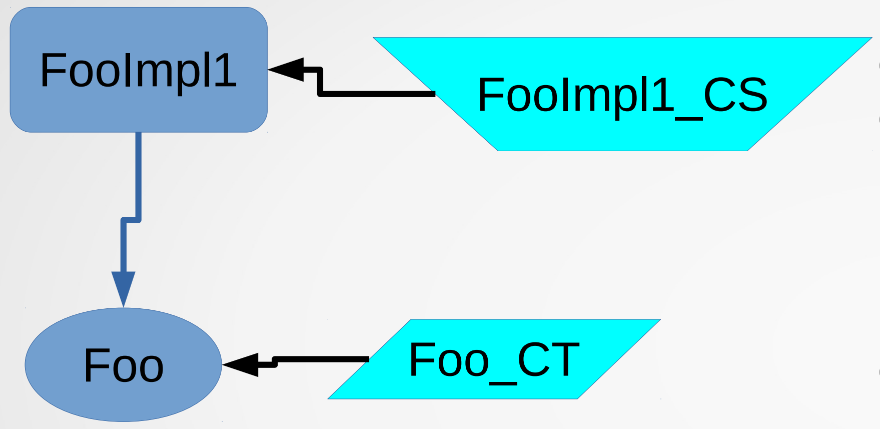
1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.
3. Find all classes annotated with Contract and which test an ancestor interface.
4. Instantiate each class found in step 3.
5. Create a jUnit suite comprising all ContractTest annotated methods found in the classes from step 3..

Runtime Result



1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.
3. Find all classes annotated with Contract and which test an ancestor interface.
4. Instantiate each class found in step 3.
5. Create a jUnit suite comprising all ContractTest annotated methods found in the classes from step 3.
6. Get the Producer object from the contract suite and insert in classes instantiated in step 4.

Runtime Result

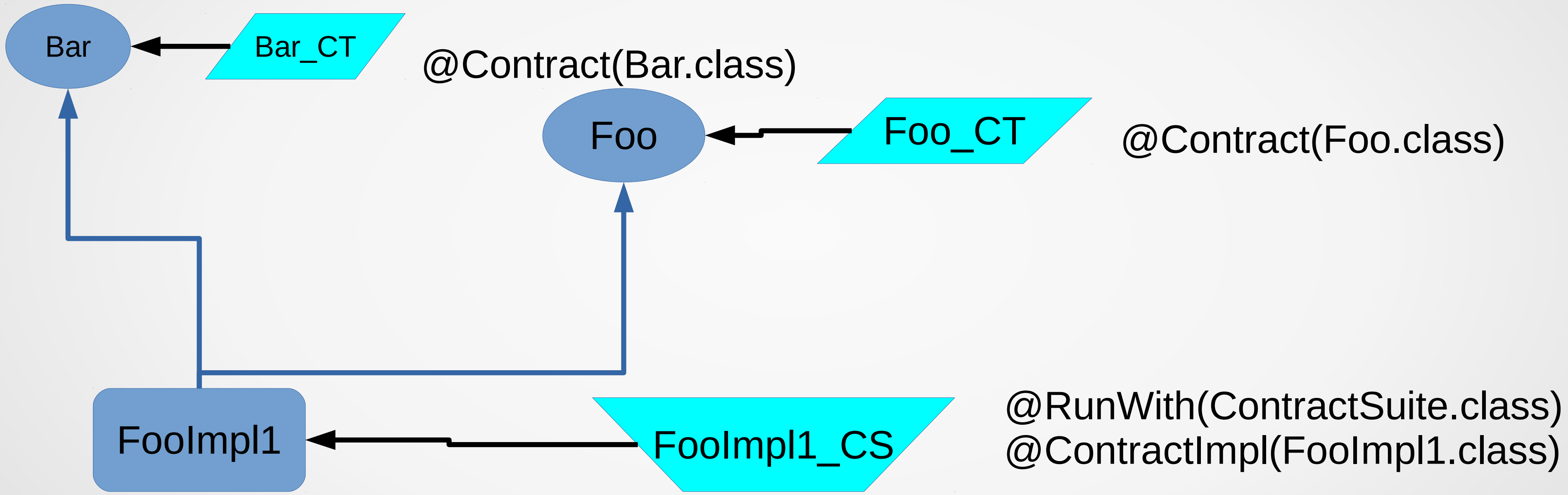


@RunWith(ContractSuite.class)
@ContractImpl(FooImpl1.class)

@Contract(Foo.class)

1. Find class specified in ContractImpl
2. Find all ancestors of the class that are interfaces.
3. Find all classes annotated with Contract and which test an ancestor interface.
4. Instantiate each class found in step 3.
5. Create a jUnit suite comprising all ContractTest annotated methods found in the classes from step 3.
6. Get the Producer object from the contract suite and insert in classes instantiated in step 4.
7. Execute the suite.

Complex Solution Diagram



Bits and Bobs

- There can be more than one ContractImpl for a single concrete class.
- ContactImpl has a skip property to ignore specific interface tests (e.g bar.class).
- Coverage reporting:
  - Unimplemented Tests
  - Untested Interfaces
- Maven reporting plugin.
- Provides a Dynamic interface which triggers testing of the classes returned from methods.

More Info

- https://github.com/Claudenw/junit-contracts
- Maven:

```
<dependency>
    <groupId>org.xenei</groupId>
    <artifactId>junit-contracts</artifactId>
    <version>0.1.5</version>
</dependency>
```

- Simplifying Contract Testing, Dr. Dobb's Journal, May 2014. http://www.drdobbs.com/testing/simplifying-contract-testing/240167128

QA

PHILOSOPHISING GEEKS
geek & poke
SOMETIMES I ASK MYSELF: "BUT WHAT IF THIS TEST FAILS?"
assert(true);