Qi4j - Abstract Composites

Rickard Öberg

Agenda

Agenda

- What is an Abstract Composite?
- Abstract Composite structure
- Abstract Composite examples
  - Default Constraints
  - Generic Mixins
  - Caching
  - Aggregation
  - Validation

What is an Abstract Composite?

What is an Abstract Composite?

:: Qi4j contains reusable Fragments

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
    :: Constraints

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
  :: Constraints
  :: Assertions

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
  :: Constraints
  :: Assertions
  :: SideEffects

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
  :: Constraints
  :: Assertions
  :: SideEffects
  :: Mixins

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
  :: Constraints
  :: Assertions
  :: SideEffects
  :: Mixins
:: Can be grouped together in extendable Abstract Composite interfaces for easy reuse

What is an Abstract Composite?

:: Qi4j contains reusable Fragments
  :: Constraints
  :: Assertions
  :: SideEffects
  :: Mixins
:: Can be grouped together in extendable Abstract Composite interfaces for easy reuse
  :: Examples: Caching, Default constraints, Aggregation

Abstract Composite structure

Abstract Composite structure

:: Abstract Composites are regular Composite interfaces which declare a number of Fragments

Abstract Composite structure

:: Abstract Composites are regular Composite interfaces which declare a number of Fragments
:: Useless on their own

Abstract Composite structure

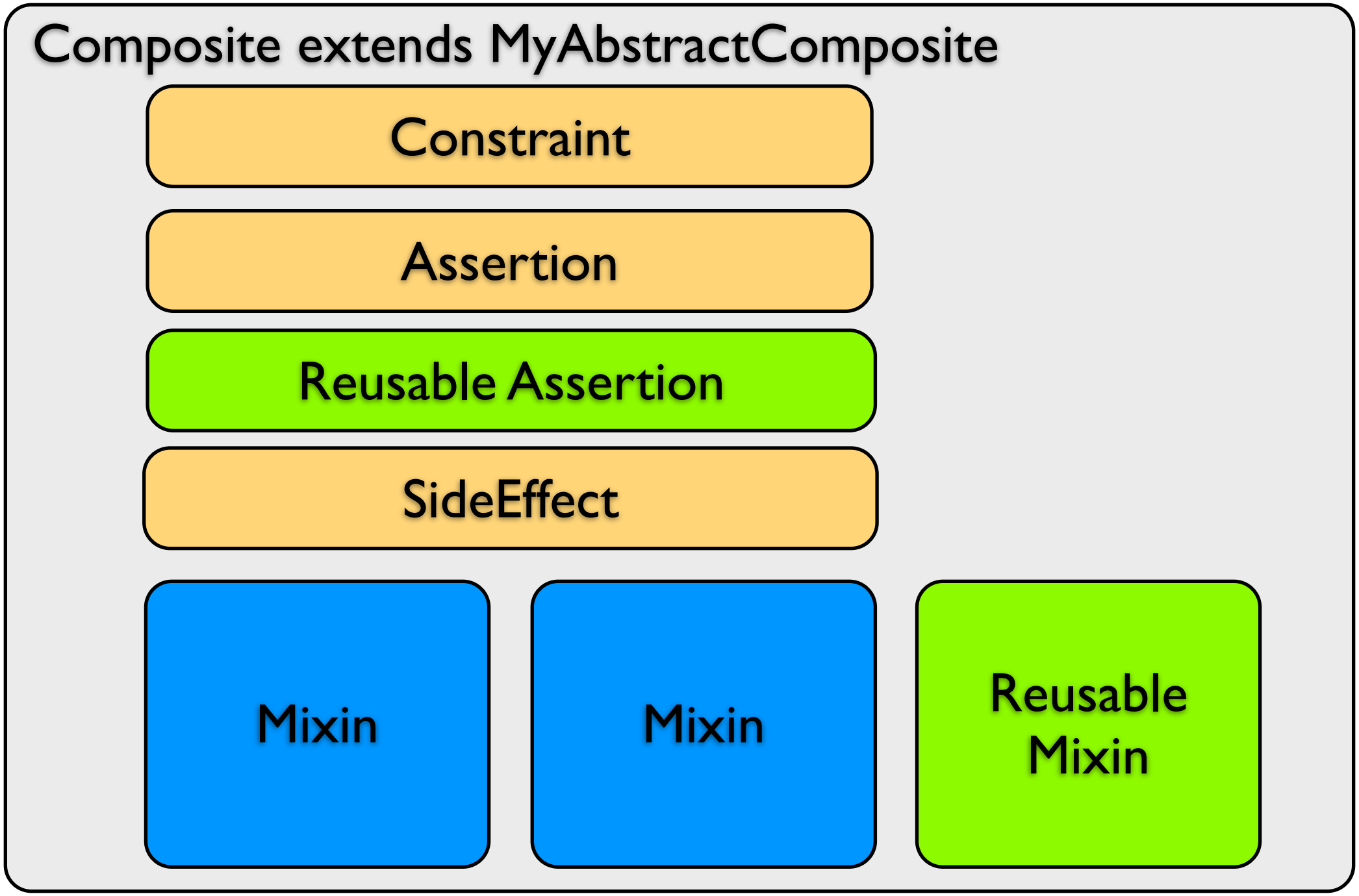
:: Abstract Composites are regular Composite interfaces which declare a number of Fragments
:: Useless on their own
:: Extend it to use

Abstract Composite structure

:: Abstract Composites are regular Composite interfaces which declare a number of Fragments
:: Useless on their own
:: Extend it to use
:: A Composite can extend many Abstract Composites at the same time

Abstract Composite structure

:: Abstract Composites are regular Composite interfaces which declare a number of Fragments
:: Useless on their own
:: Extend it to use
:: A Composite can extend many Abstract Composites at the same time
:: Check what annotations are used by the Fragments in the Abstract Composite


Composite extends MyAbstractComposite
Constraint
Assertion
Reusable Assertion
SideEffect
Mixin
Mixin
Reusable Mixin

Standard Abstract Composites

Standard Abstract Composites

:: Some Abstract Composites provide library functionality

Standard Abstract Composites

:: Some Abstract Composites provide library functionality
:: It can be useful for a project to have a "StandardAbstractComposite" base interface that all domain Composites extend from, and which extends other Abstract Composites

Standard Abstract Composites

- Some Abstract Composites provide library functionality
- It can be useful for a project to have a "StandardAbstractComposite" base interface that all domain Composites extend from, and which extends other Abstract Composites
- Makes it easy to add/remove standard functionality later on

Default constraints

HelloWorldComposite.java:

```
public interface HelloWorldComposite
    extends StandardComposite
{
    @Cached String say();

    void setPhrase( @NonEmptyString @Contains( "H" )String phrase )
        throws ValidationException;

    String getPhrase();

    void setName( @NotNull @Matches( "Universe|World" )String name )
        throws ValidationException;

    String getName();
}
```

HelloWorldComposite.say.js:

```
function say()
{
    return phrase + " " + name;
}
```

HelloWorldComposite.properties:

```
phrase.not.null=Phrase may not be null
phrase.contains=Phrase "{0}" must contain the string "{1}"
phrase.min.length=Length of phrase "{0}" is too short. It must be at least {1} characters
name.not.null=Name may not be null
name.matches=Name must be either Universe or World
```

Default constraints

```
StandardComposite.java:
@Assertions( ReturnCachedValueAssertion.class )
public interface StandardAbstractComposite
    extends InvocationCacheAbstractComposite, DefaultConstraintsAbstractComposite,
ValidatableAbstractComposite, GenericMixinsAbstractComposite
{
}
```

Abstract Composites in Qi4j

Abstract Composites in Qi4j

:: A number of Abstract Composites are available in Qi4j

Abstract Composites in Qi4j

:: A number of Abstract Composites are available in Qi4j
:: Implements a number of common usecases

Default Constraints

Default Constraints

:: Defines and declares a number of commonly useful annotations and Constraints

Default Constraints

:: Defines and declares a number of commonly useful annotations and Constraints
:: Extend DefaultConstraintsAbstractComposite to use

Default Constraints

```
@Constraints( { NotNullConstraint.class,
    MinLengthConstraint.class,
    MaxLengthConstraint.class,
    GreaterThanConstraint.class,
    LessThanConstraint.class,
    ContainsConstraint.class,
    MatchesConstraint.class } )
public interface DefaultConstraintsAbstractComposite
{
}
```

Generic Mixins

Generic Mixins

:: Qi4j contains a number of Generic Mixins

Generic Mixins

:: Qi4j contains a number of Generic Mixins
:: Provides implementations based on conventions

Generic Mixins

- Qi4j contains a number of Generic Mixins
- Provides implementations based on conventions
  - PropertiesMixin - get/set/add/remove

Generic Mixins

- Qi4j contains a number of Generic Mixins
- Provides implementations based on conventions
  - PropertiesMixin - get/set/add/remove
  - FinderMixin - findByNameOrEmail

Generic Mixins

:: Qi4j contains a number of Generic Mixins
:: Provides implementations based on conventions
  :: PropertiesMixin - get/set/add/remove
  :: FinderMixin - findByNameOrEmail
:: Provides implementations for scripting

Generic Mixins

:: Qi4j contains a number of Generic Mixins
:: Provides implementations based on conventions
  :: PropertiesMixin - get/set/add/remove
  :: FinderMixin - findByNameOrEmail
:: Provides implementations for scripting
  :: JavaScriptMixin

Generic Mixins

:: Qi4j contains a number of Generic Mixins
:: Provides implementations based on conventions
  :: PropertiesMixin - get/set/add/remove
  :: FinderMixin - findByNameOrEmail
:: Provides implementations for scripting
  :: JavaScriptMixin
:: Provides implementations for infrastructure delegation

Generic Mixins

- Qi4j contains a number of Generic Mixins
- Provides implementations based on conventions
  - PropertiesMixin - get/set/add/remove
  - FinderMixin - findByNameOrEmail
- Provides implementations for scripting
  - JavaScriptMixin
- Provides implementations for infrastructure delegation
  - RMIMixin

Generic Mixins

:: Qi4j contains a number of Generic Mixins
:: Provides implementations based on conventions
  :: PropertiesMixin - get/set/add/remove
  :: FinderMixin - findByNameOrEmail
:: Provides implementations for scripting
  :: JavaScriptMixin
:: Provides implementations for infrastructure delegation
  :: RMIMixin
:: NoopMixin - can handle anything!

PropertiesMixin example

```
public interface Foo
{
  void setBar(String name);
  String getBar();

  void addXYZ(Xyz xyzzy);
  void removeXYZ(Xyz xyzzy);
  Iterable<Xyz> getXyzzy();
}
```

FinderMixin example

```
public interface PeopleFinder
{
  PersonComposite findByName(String name);

  Iterable<PersonComposite> findByZipCode(String ZipCode);
}
```

JavaScriptMixin example

```
public interface HelloWorldComposite
    extends StandardAbstractComposite
{
    @Cached String say();

    void setPhrase( @NonEmptyString @Contains( "H" )String phrase )
        throws ValidationException;

    String getPhrase();

    void setName( @NotNull @Matches( "Universe|World" )String name )
        throws ValidationException;

    String getName();
}

HelloWorldComposite.say.js:
function say()
{
    return phrase + " " + name;
}
```

RMIMixin example

```
@Mixins( RMIMixin.class )
public interface RemoteInterfaceComposite
    extends RemoteInterface, InvocationCacheAbstractComposite, Composite
{
}

@Cached
public interface RemoteInterface
    extends Remote
{
    String foo( String aBar )
        throws IOException;
}

RemoteInterfaceImpl remoteObject = new RemoteInterfaceImpl();
RemoteInterface stub = (RemoteInterface) UnicastRemoteObject.exportObject( remoteObject,
0 );
Registry registry = LocateRegistry.createRegistry( 1099 );
registry.rebind( RemoteInterface.class.getSimpleName(), stub );

CompositeBuilderFactory factory = new CompositeBuilderFactoryImpl();
RemoteInterface remote = factory.newCompositeBuilder
( RemoteInterfaceComposite.class ).newInstance();

System.out.println( remote.foo( "Bar" ) );
```

DecoratorMixin example

```
public class DecoratorMixin
    implements InvocationHandler
{
    Object delegate;
    public DecoratorMixin( @Decorate Object delegate )
    {
        this.delegate = delegate;
    }
    public Object invoke( Object object, Method method, Object[] args ) throws Throwable
    {
        if( delegate instanceof InvocationHandler )
        {
            InvocationHandler handler = (InvocationHandler) delegate;
            return handler.invoke( object, method, args );
        }
        else
        {
            try
            {
                return method.invoke( delegate, args );
            }
            catch( InvocationTargetException e )
            {
                throw e.getCause();
            }
        }
    }
}
```

Invocation Caching

Invocation Caching

:: If a method call is expensive, use result caching to avoid unnecessary computation

Invocation Caching

- If a method call is expensive, use result caching to avoid unnecessary computation
- Can be used in distributed setups to mask connection errors

Invocation Caching

- If a method call is expensive, use result caching to avoid unnecessary computation
- Can be used in distributed setups to mask connection errors
- InvocationCacheAbstractComposite provides a number of Mixins, Assertions and SideEffects that implement caching for you

Invocation Caching

- If a method call is expensive, use result caching to avoid unnecessary computation
- Can be used in distributed setups to mask connection errors
- InvocationCacheAbstractComposite provides a number of Mixins, Assertions and SideEffects that implement caching for you
- Trigger using @Cached annotation on methods, in domain interface or Mixin

Invocation Caching

Invocation Caching

:: Trigger using @Cached annotation on methods

Invocation Caching

- Trigger using @Cached annotation on methods
- Method results are cached with method name+arguments as key as a SideEffect

Invocation Caching

- Trigger using @Cached annotation on methods
- Method results are cached with method name+arguments as key as a SideEffect
- Cache-aware Assertions return cached values always or only on exception

Invocation Caching

- Trigger using @Cached annotation on methods
- Method results are cached with method name+arguments as key as a SideEffect
- Cache-aware Assertions return cached values always or only on exception
- Invalidate cache explicitly using InvocationCache interface or implicitly using SideEffects on setters

InvocationCaching Example

```
public interface HelloWorldComposite
    extends InvocationCacheAbstractComposite
{
    @Cached String say();

    void setPhrase( String phrase );

    String getPhrase();

    void setName( String name );

    String getName();
}
```

Aggregation

Aggregation

:: When working with persistent objects (Entities) it is often useful to express Aggregates

Aggregation

:: When working with persistent objects (Entities) it is often useful to express Aggregates
  :: Common lifecycle

Aggregation

- When working with persistent objects (Entities) it is often useful to express Aggregates
  - Common lifecycle
  - Common validation rules

Aggregation

- When working with persistent objects (Entities) it is often useful to express Aggregates
  - Common lifecycle
  - Common validation rules
- Aggregated objects implements the interface Aggregated

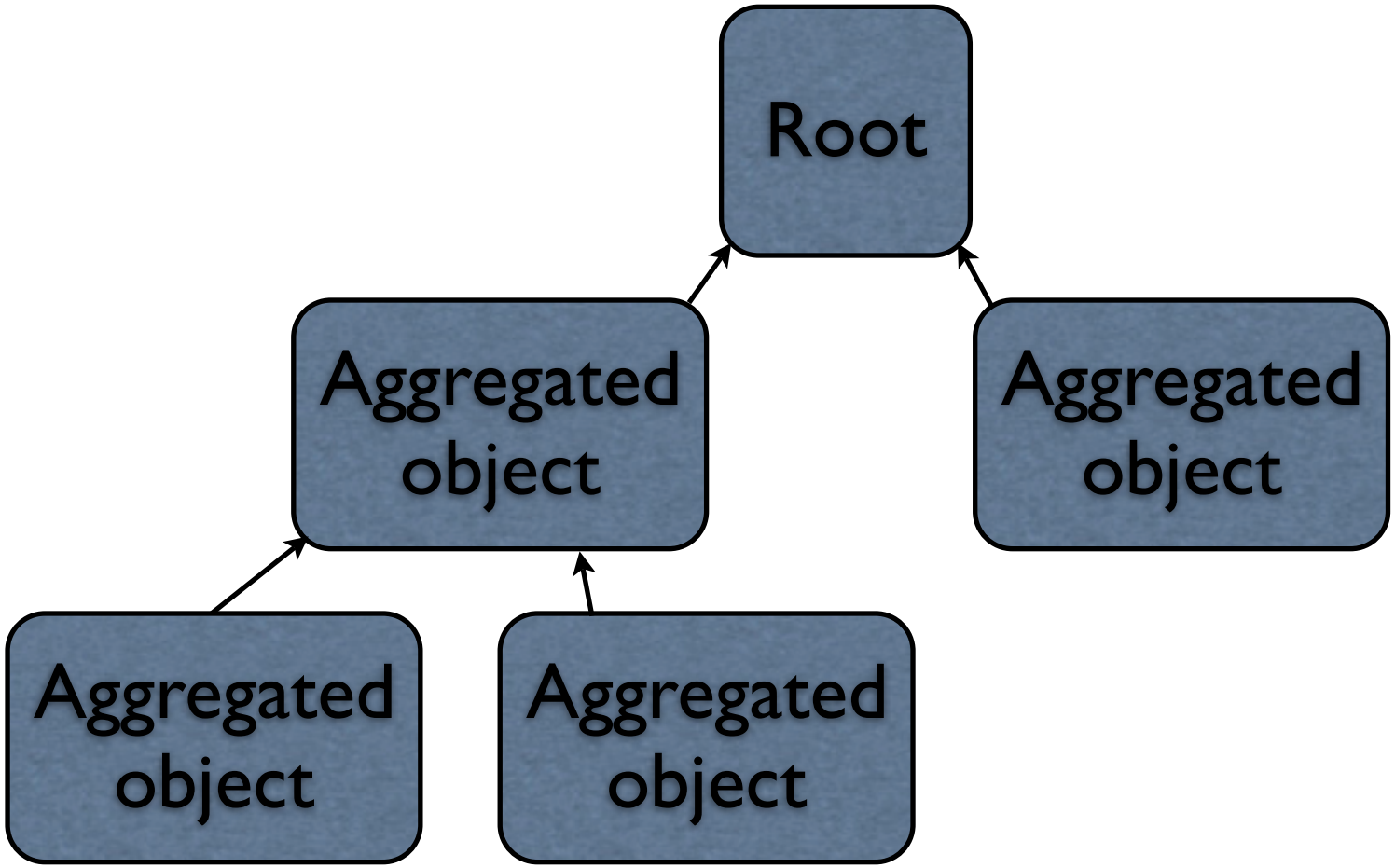
Aggregation

- When working with persistent objects (Entities) it is often useful to express Aggregates
  - Common lifecycle
  - Common validation rules
- Aggregated objects implements the interface Aggregated
  - AggregatedMixin contains reference to "owning" object

Aggregation

- When working with persistent objects (Entities) it is often useful to express Aggregates
  - Common lifecycle
  - Common validation rules
- Aggregated objects implements the interface Aggregated
  - AggregatedMixin contains reference to “owning” object
  - Owning objects may be Aggregated themselves

Aggregation tree



Aggregation Example

```
public abstract class AggregateValidationAssertion
    implements Validatable
{
    @ThisAs Aggregated aggregated;
    @AssertionFor Validatable next;

    public List<ValidationMessage> validate()
    {
        List<ValidationMessage> messages = next.validate();

        Object aggregator = aggregated.getAggregate();
        if( aggregator instanceof Validatable )
        {
            Validatable aggregatorValidation = (Validatable) aggregator;
            messages.addAll( aggregatorValidation.validate() );

        }
        return messages;
    }
}
```

Validation

Validation

:: Often there are business rules that needs to be checked to verify that an object is in a valid state

Validation

- Often there are business rules that needs to be checked to verify that an object is in a valid state
- Check either statically (isValid) or dynamically (Assertion)

Validation

- Often there are business rules that needs to be checked to verify that an object is in a valid state
- Check either statically (isValid) or dynamically (Assertion)
- Many validation problems may need to be collected

Validation

- Often there are business rules that needs to be checked to verify that an object is in a valid state
- Check either statically (isValid) or dynamically (Assertion)
- Many validation problems may need to be collected
  - Web form

ValidatableAbstractComposite

```
@Assertions( { ValidatableMessagesAssertion.class, ConstraintValidationAssertion.class,
ChangeValidationAssertion.class } )
@Mixins( { ValidatableMixin.class, ValidationMessagesMixin.class } )
public interface ValidatableAbstractComposite
    extends Validatable, Composite
{
}
```

Validatable Example

```
public final class NoDuplicateNamesValidatableAssertion
    extends AbstractValidatableAssertion
{
    @ThisAs Folder folder;

    @Override protected void isValid( Validator validator )
    {
        Set<String> names = new HashSet<String>();
        for(Nameable nameableChild : folder.getChildren())
        {
            validator.error(names.contains(nameableChild.getName()), “duplicate.name”);
            names.add(nameableChild.getName());
        }
    }
}
```

Questions?