

EAPLI
Teórico-Prática

EAPLI Framework

What it is?

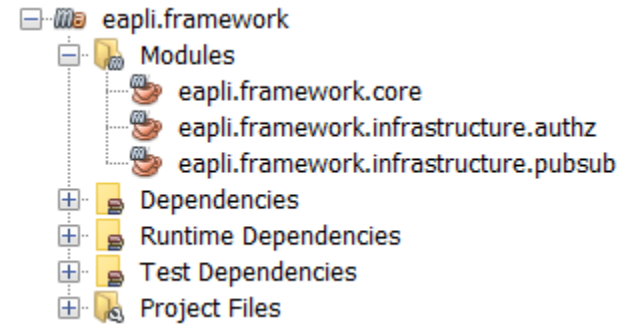
Standard application framework developed within the scope of the “*Engenharia de Aplicações*” course at Instituto Superior de Engenharia do Porto to teach Domain Driven Design and JPA Persistence.

Disclaimer:

The framework has more functionality than what is needed for EAPLI.
Read documentation in the Git repo

Modules

- **Core**
 - Domain driven design
 - Persistence
 - Utilities
 - Simple Console Presentation framework
- **Infrastructure Authz**
- **Infrastructure Pub/Sub**
 - Event publishing and dispatching



← For EAPLI it might not be needed

pom.xml (eapli.framework)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="...">
  <modelVersion>4.0.0</modelVersion>
  <packaging>pom</packaging>
  <version>9.1.1</version>
  <name>eapli.framework</name>
  <artifactId>eapli.framework</artifactId>
  <version>1.3.0-SNAPSHOT</version>

  <parent>...</parent>

  <properties>...</properties>

  <modules>
    <module>eapli.framework.core</module>
    <module>eapli.framework.infrastructure.authz</module>
    <module>eapli.framework.infrastructure.pubsub</module>
  </modules>

  <dependencies>...</dependencies>

  <build>...</build>

  <distributionManagement>
    <repository>
      <id>bintray-pagsousa-eapli</id>
      <name>pagsousa-eapli</name>
      <url>https://api.bintray.com/maven/pagsousa/eapli/eapli.framework/;publish=1</url>
    </repository>
  </distributionManagement>
</project>
```

EAPLI framework

Build

Build

- **Maven**

- Dependency manager
- Artifact (jar) repository
- Build automation
- other tasks:
 - deploy
 - run
 - ...

- **Works for Eclipse, IntelliJ, Netbeans**

- **Pom.xml**

pom.xml (ecafeteria.base)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="...">
  <modelVersion>4.0.0</modelVersion>
  <groupId>eapli</groupId>
  <artifactId>ecafeteria</artifactId>
  <version>1.3.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <properties> ... </properties>
  <modules> ... </modules>
  <dependencies>
    <dependency>
      <groupId>eapli</groupId>
      <artifactId>eapli.framework.core</artifactId>
      <version>9.1.1</version>
    </dependency>
    <dependency>
      <groupId>eapli</groupId>
      <artifactId>eapli.framework.infrastructure.authz</artifactId>
      <version>9.1.1</version>
    </dependency>
    <dependency>
      <groupId>eapli</groupId>
      <artifactId>eapli.framework.infrastructure.pubsub</artifactId>
      <version>9.1.1</version>
    </dependency>
    <dependency> ... </dependency>
    <dependency> ... </dependency>
  </dependencies>
  <repositories>
    <repository>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>bintray-pagsousa-eapli</id>
      <url>http://dl.bintray.com/pagsousa/eapli</url>
    </repository>
  </repositories>
</project>
```

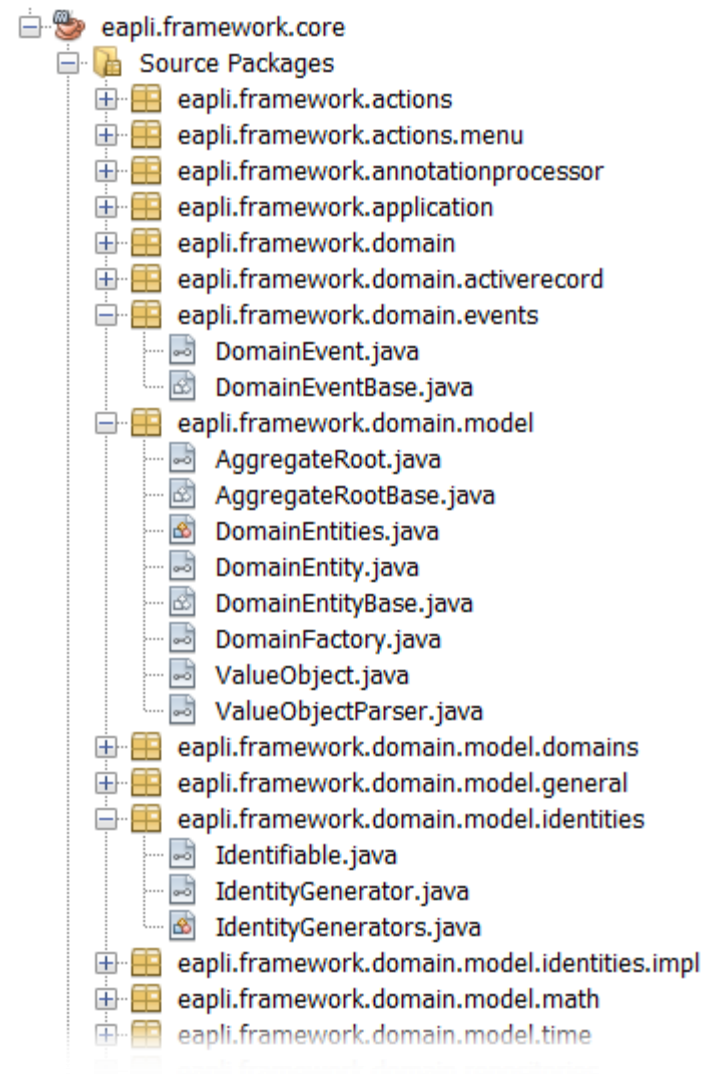
EAPLI framework

Core
Domain Driven Design

Domain Driven Design

■ DDD pattern interfaces

- ValueObject
- DomainEntity
- AggregateRoot



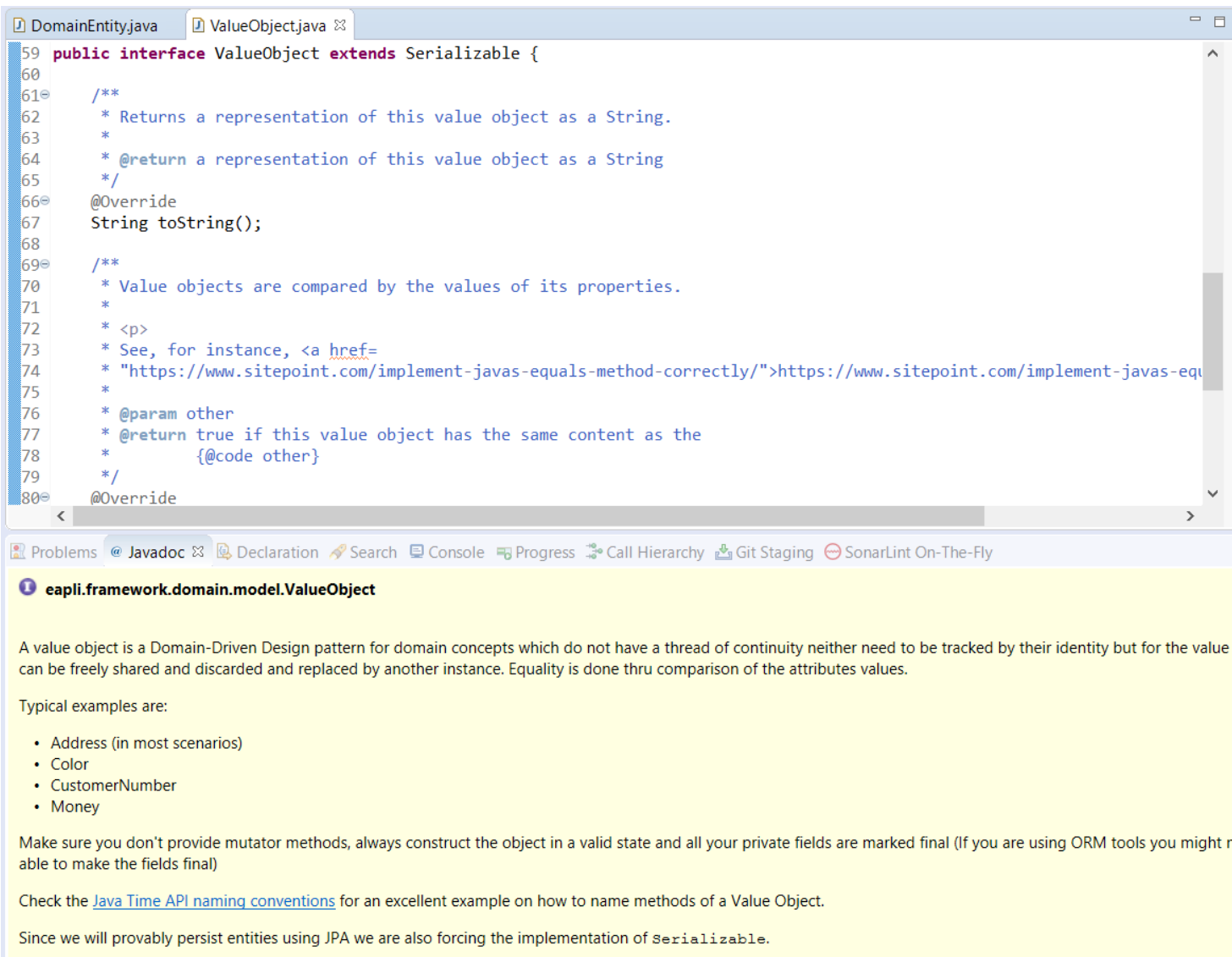
DomainEntity.java

```
DomainEntity.java
20 * Copyright (c) 2013-2019 the original author or authors.
21 package eapli.framework.domain.model;
22
23 import java.io.Serializable;
24
25 /**
26  * An entity is a Domain-Driven Design pattern for concepts in the domain which
27  * have a thread of continuity which needs to be tracked.
28  *
29  * <p>
30  * These concepts matter by their identity and we need to track them
31  * continuously. Instance equality must be done thru the identity of the objects
32  * and we cannot loose track or allow duplication of an entity. By definition a
33  * domain entity always has a business identity, so it should be impossible to
34  * create new instances without identity.
35  *
36  * <p>
37  * Typical examples are:
38  * <ul>
39  * <li>Product
40  * <li>Person
41  * <li>Account
42  * </ul>
43  *
44  * <p>
45  * Since we will provably persist entities using JPA we are also forcing the
46  * implementation of {@code Serializable}.
47  *
48  * @param <I>
49  *     the type of the primary <b>business</b> id of the entity
50  * @author Paulo Gandra Sousa
51  */
52 public interface DomainEntity<I> extends Comparable<I> /* & ValueObject */
53     extends Identifiable<I>, Comparable<I>, Serializable {
54
55     /**
56      * Entities are compared by identity only. No need to compare all fields of
57      * the object. you can use the
58      * {@link DomainEntity#equals(DomainEntity, Object)} method as default
59      */
60 }
```

Problems Javadoc Declaration Search Console Progress Call Hierarchy Git Staging SonarLint On-The-Fly

eapli.framework.domain.model.DomainEntity<I> extends Comparable<I>>

ValueObject.java



The screenshot shows an IDE window with two tabs: `DomainEntity.java` and `ValueObject.java`. The `ValueObject.java` tab is active, displaying the following code:

```
59 public interface ValueObject extends Serializable {
60
61     /**
62      * Returns a representation of this value object as a String.
63      *
64      * @return a representation of this value object as a String
65      */
66     @Override
67     String toString();
68
69     /**
70      * Value objects are compared by the values of its properties.
71      *
72      * <p>
73      * See, for instance, <a href=
74      * "https://www.sitepoint.com/implement-javas-equals-method-correctly/">https://www.sitepoint.com/implement-javas-eq
75      *
76      * @param other
77      * @return true if this value object has the same content as the
78      *         {@code other}
79      */
80     @Override
```

Below the code editor, the Javadoc for `eapli.framework.domain.model.ValueObject` is displayed. It includes a description of a value object, typical examples, and implementation guidelines.

eapli.framework.domain.model.ValueObject

A value object is a Domain-Driven Design pattern for domain concepts which do not have a thread of continuity neither need to be tracked by their identity but for the value o can be freely shared and discarded and replaced by another instance. Equality is done thru comparison of the attributes values.

Typical examples are:

- Address (in most scenarios)
- Color
- CustomerNumber
- Money

Make sure you don't provide mutator methods, always construct the object in a valid state and all your private fields are marked final (If you are using ORM tools you might ne able to make the fields final)

Check the [Java Time API naming conventions](#) for an excellent example on how to name methods of a Value Object.

Since we will provably persist entities using JPA we are also forcing the implementation of `Serializable`.

AggregateRoot.java

```
DomainEntity.java ValueObject.java Money.java MoneyAddTest.java AggregateRoot.java
24 | * Copyright (c) 2013-2019 the original author or authors.
25 | package eapli.framework.domain.model;
26 | /**
27 |  * An aggregate root is a Domain-Driven Design pattern for defining scopes of
28 |  * change and "whole-objects". some entities cluster other objects (entities and
29 |  * value objects) which don't make sense to exist outside of that entity (e.g.,
30 |  * OrderLine and Order).
31 |  *
32 |  * the aggregate root is the entity serving as a root for that cluster of
33 |  * objects. these are the only objects that client code can interact directly
34 |  * and that is managed by Repositories. "inside" objects cannot be directly
35 |  * manipulated by client code, instead they need to be manipulated thru the
36 |  * aggregate root.
37 |  *
38 |  * Typical examples are:
39 |  * <ol>
40 |  * <li>Order {OrderLine, Billing Address, Shipping Address}
41 |  * <li>Customer {Home Address}
42 |  * <li>Product
43 |  * </ol>
44 |  *
45 |  * @param <I> the type of the primary <b>business</b> identity of the entity
46 |  *
47 |  * @author Paulo Gandra Sousa
48 |  */
49 | public interface AggregateRoot<I> extends Comparable<I> /* & ValueObject */ extends DomainEntity<I> {
50 |     // empty
51 | }
52 |
```

Problems @ Javadoc Declaration Search Console Progress Call Hierarchy Git Staging SonarLint On-The-Fly

eapli.framework.domain.model.AggregateRoot<I> extends Comparable<I>>

Identifiable.java

```
DomainEntity.java ValueObject.java Money.java MoneyAddTest.java AggregateRoot.java Identifiable.java x
2* | * Copyright (c) 2013-2019 the original author or authors.
24 package eapli.framework.domain.model.identities;
25
26 import com.fasterxml.jackson.annotation.JsonGetter;
27
28 /**
29  * An object that has an identity.
30  *
31  * @author Paulo Gandra Sousa
32  * @param <T>
33  *         the type of the entity's identity. e.g., if an object Person is
34  *         identified by an IdCardNumber, the class Person should implement
35  *         interface Identifiable<IdCardNumber>
36  */
37 public interface Identifiable<T> {
38
39     /**
40      * Returns the primary <b>business</b> id of the entity.
41      *
42      * <p>
43      * This method is marked as a getter for Jackson serialization
44      *
45      * @return the primary <b>business</b> id of the entity
46      */
47     @JsonGetter
48     T identity();
49
50     /**
51      * checks if the object has a certain business identity
52      *
53      * @param otherId
54      *         the identity to compare
55      * @return true if the object has that identity
56      */
57     default boolean hasIdentity(final T otherId) {
58         return identity().equals(otherId);
59     }
60 }
61
```

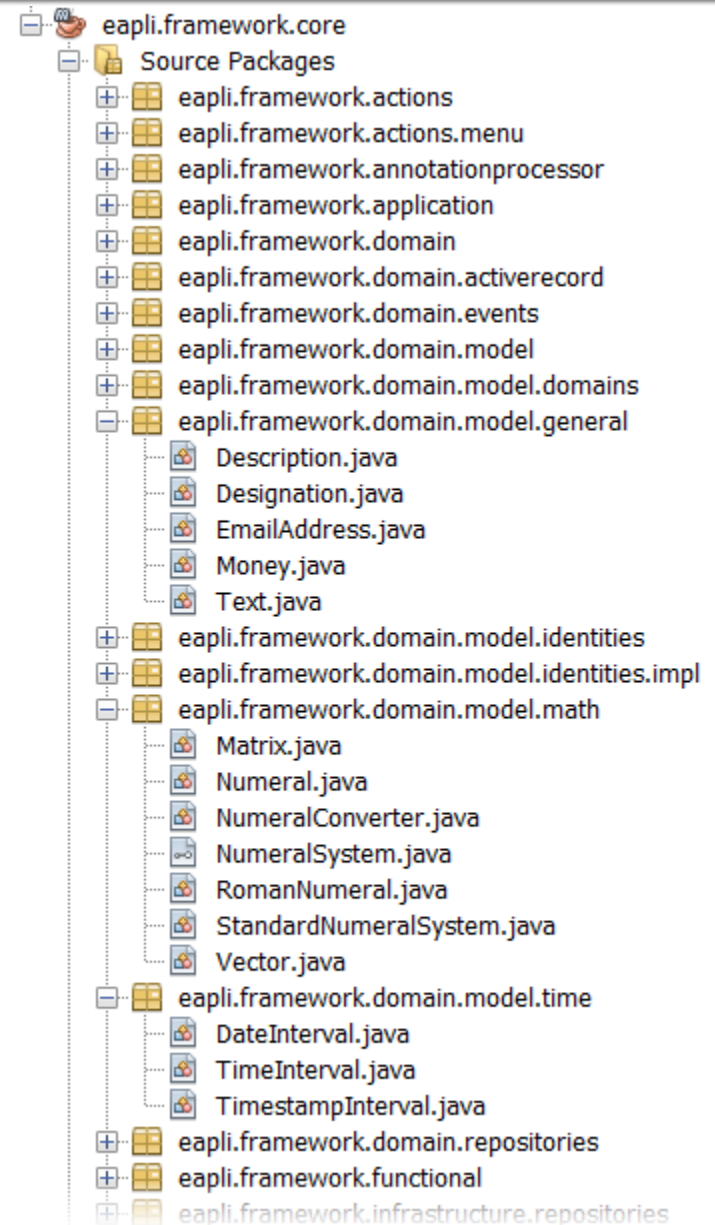
Problems @ Javadoc Declaration Search Console Progress Call Hierarchy Git Staging SonarLint On-The-Fly

eapli.framework.domain.model.identities.Identifiable<T>

Domain Driven Design

■ Domain objects

- Money
- Range
- Time period
- ...



Money.java

The screenshot shows an IDE window with three tabs: `DomainEntity.java`, `ValueObject.java`, and `Money.java`. The `Money.java` tab is active, displaying the following code:

```
89  *
90  */
91  @Embeddable
92  @JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY)
93  public class Money implements Comparable<Money>, Serializable, ValueObject {
94
95      private static final long serialVersionUID = 1L;
96
97      private static final transient int HUNDRED = 100;
98      // @Convert(converter = MoneyAmountConverter.class)
99      private final BigInteger amount;
100     private final Currency currency;
101
102     /**
103      * For ORM tool only
104      *
105      * @author Paulo Gandra de Sousa
106      */
107     protected Money() {
108         amount = null;
109         currency = null;
110     }
```

Below the code editor, the Javadoc for `Money` is displayed:

eaapl.framework.domain.model.general.Money

[@Embeddable](#)
[@JsonAutoDetect\(fieldVisibility=ANY\)](#)

Represents money values.

For a full description see the book *Patterns of Enterprise Application Architecture* (Martin Fowler) page 488 or [PoEAA Money](#). See also a [good discussion](#) about pros and cons.

A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming languages, the most obvious surrounding currencies. If all your calculations are done in a single `m_currency`, this isn't a huge problem, but once you involve multiple currencies to your yen without taking the `m_currency` differences into account. The more subtle problem is with rounding. Monetary calculations are often rounded to the smallest unit (lose pennies (or your local equivalent) because of rounding errors.

The good thing about object-oriented programming is that you can fix these problems by creating a `Money` class that handles them. Of course, it's still surprising that no one actually does this.

- Martin Fowler

MoneyAddTest.java

Package E... Type Hier... JUnit

24 * Copyright (c) 2013-2019 the original author or authors.

```
24 package eapli.framework.domain.model.general;
25
26 import static org.junit.Assert.assertEquals;
27
28 /**
29  *
30  * @author Paulo Gandra de Sousa
31  *
32  */
33 public class MoneyAddTest {
34
35     @Test
36     public void ensureAdd0() {
37         final Money subject = Money.euros(1);
38         final Money other = Money.euros(0);
39         final Money expected = Money.euros(1);
40         assertEquals(expected, subject.add(other));
41     }
42
43     @Test
44     public void ensureAdd1() {
45         final Money subject = Money.euros(1);
46         final Money other = Money.euros(1);
47         final Money expected = Money.euros(2);
48         assertEquals(expected, subject.add(other));
49     }
50
51     @Test
52     public void ensureAddCents() {
53         final Money subject = Money.euros(0.99);
54         final Money other = Money.euros(0.01);
55         final Money expected = Money.euros(1);
56         assertEquals(expected, subject.add(other));
57     }
58
59     @Test
60     public void ensureAddCents2() {
61         final Money subject = Money.euros(1.65);
62         final Money other = Money.euros(0.01);
63         final Money expected = Money.euros(1.66);
64         assertEquals(expected, subject.add(other));
65     }
66 }
```

Problems Javadoc Declaration Search Console Progress Call Hierarchy Git Staging SonarLint On-The-Fly

eapli.framework.domain.model.general.MoneyAddTest

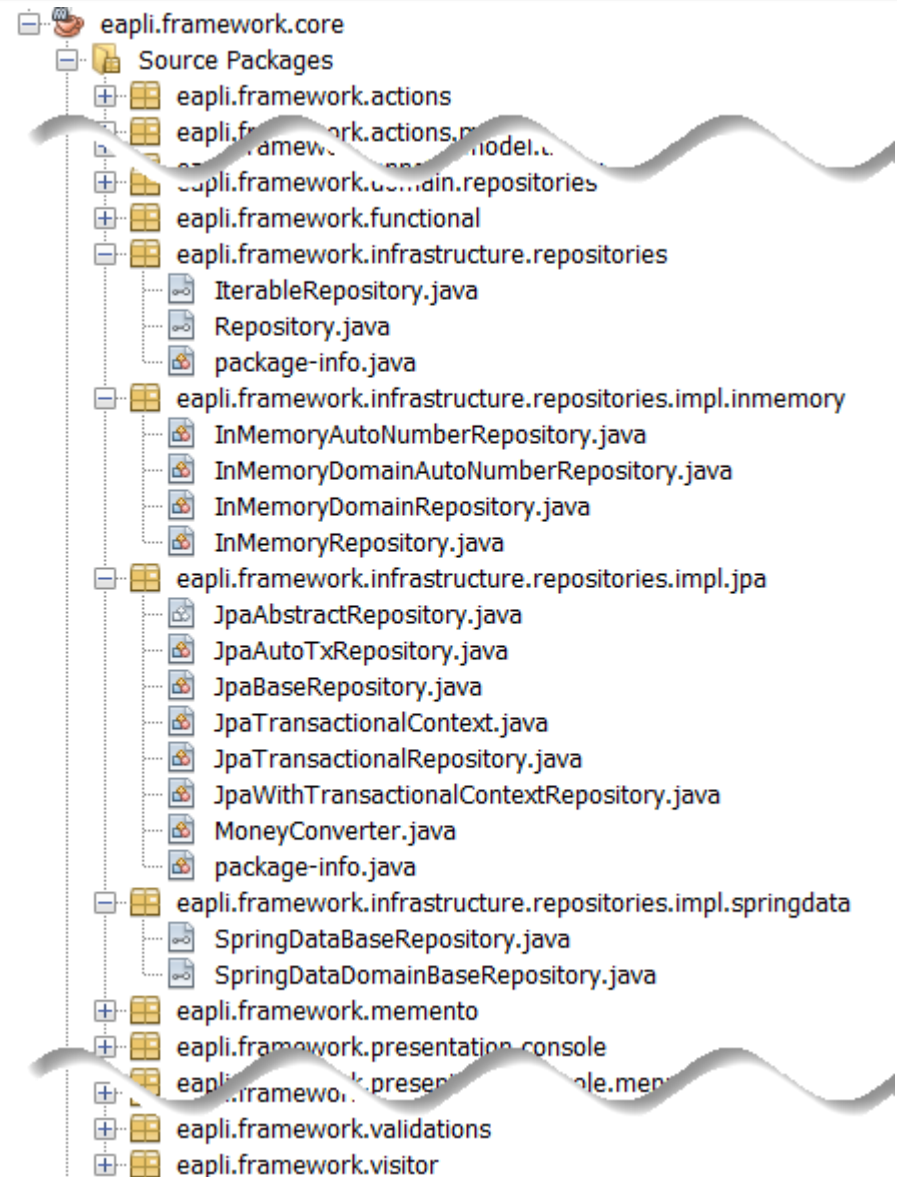
EAPLI framework

Core
Persistence

Persistence

- **Persistence**
 - Repository interfaces
- **Implementations**
 - JPA
 - InMemory
 - Spring Data JPA

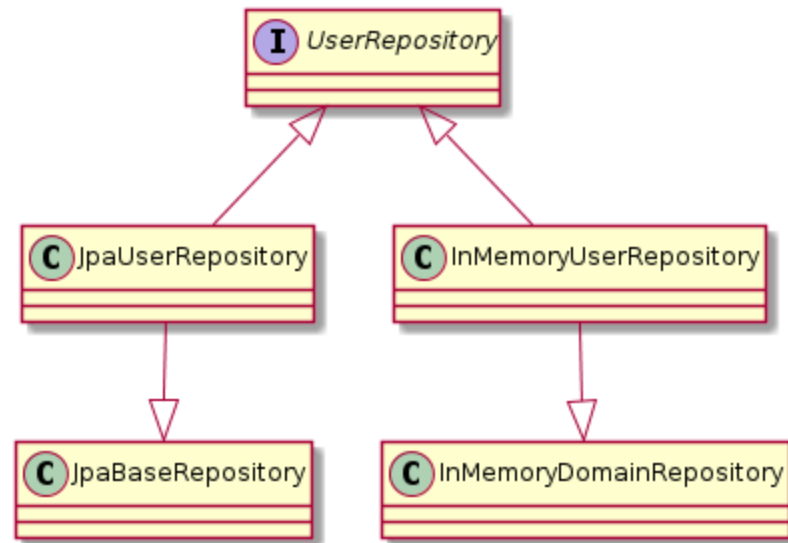
For EAPLI, spring and spring data are out of scope



Repositories

- **Repository**
- **TransactionalContext**
- **Interfaces for describing repository functionalities and transactions**

Example: User Repository

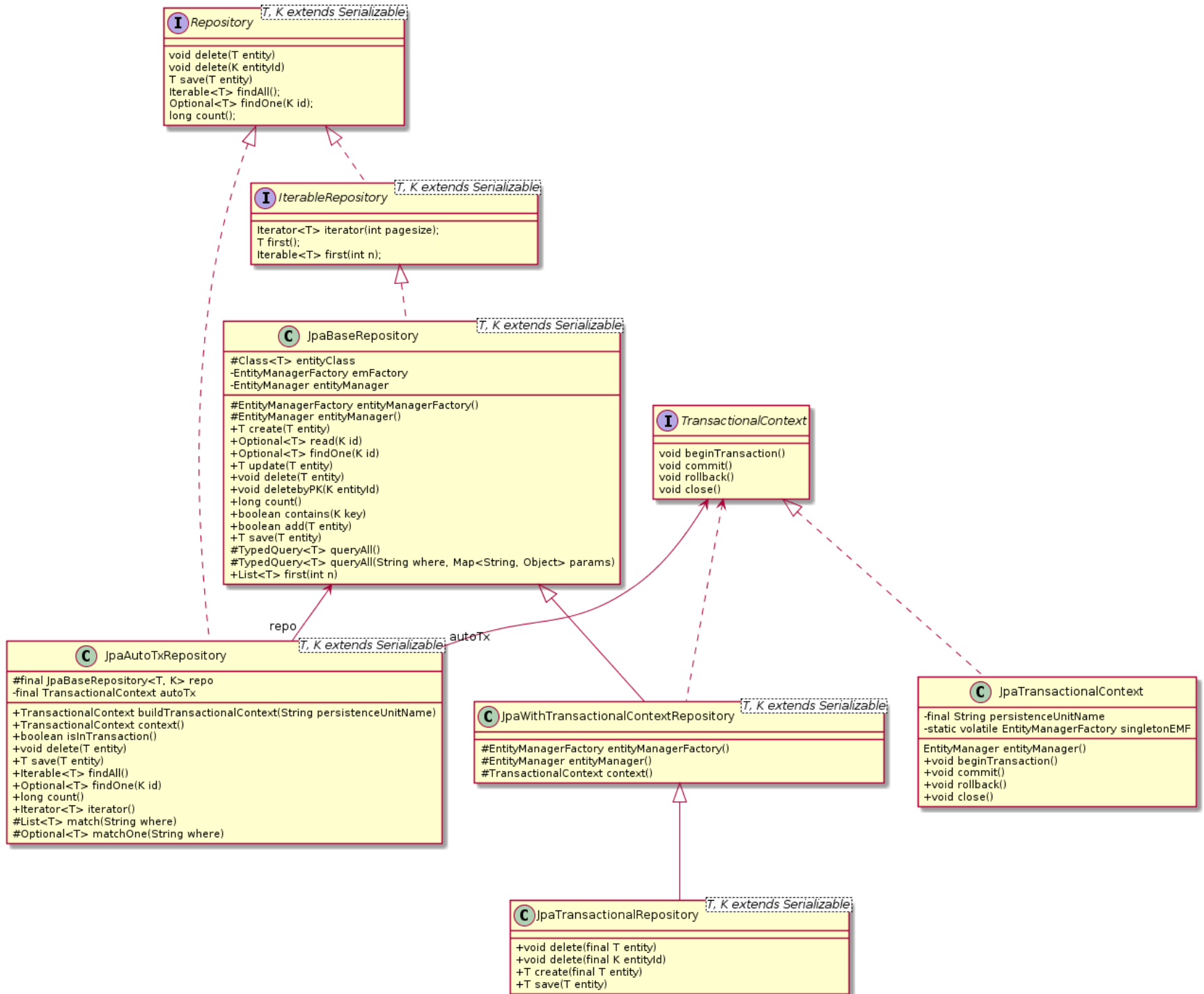


Example: User Repository

```
public interface UserRepository extends Repository<SystemUser,Username> {  
    Iterable<SystemUser> findByActive(boolean active);  
}
```

```
public class InMemoryUserRepository  
extends InMemoryDomainRepository<SystemUser,Username>  
implements UserRepository{  
    @Override  
    public Iterable<SystemUser> findByActive(final boolean active) {  
        return match(e -> e.isActive() == active);  
    }  
}
```

```
public class JpaUserRepository  
extends JpaBaseRepository<SystemUser,Username>  
implements UserRepository{  
    @Override  
    public Iterable<SystemUser> findByActive(final boolean active) {  
        return match(UserRepositoryConstants.FIND_BY_ACTIVE, "active", active);  
    }  
}
```



JPA Repositories

- **JpaRepository**

- Generic repository implementation that expects the entity manager factory to be injected by a container, e.g., web server

- **JpaNotRunningInContainerBaseRepository**

- For scenarios where the code is not running in a container but transaction is managed by the outside, e.g., controller

- **JpaTransactionalBaseRepository**

- For scenarios not running in a container but transactions are created and committed by each repository method;
- The connection is also closed automatically in each method.

- **JpaAutoTxRepository**

- Dual behaviour to either have outside transactional control or explicit transaction in each method

Transaction control

- **Control the transaction yourself using helper classes in the persistence package**



**Next lesson with a
concrete example**

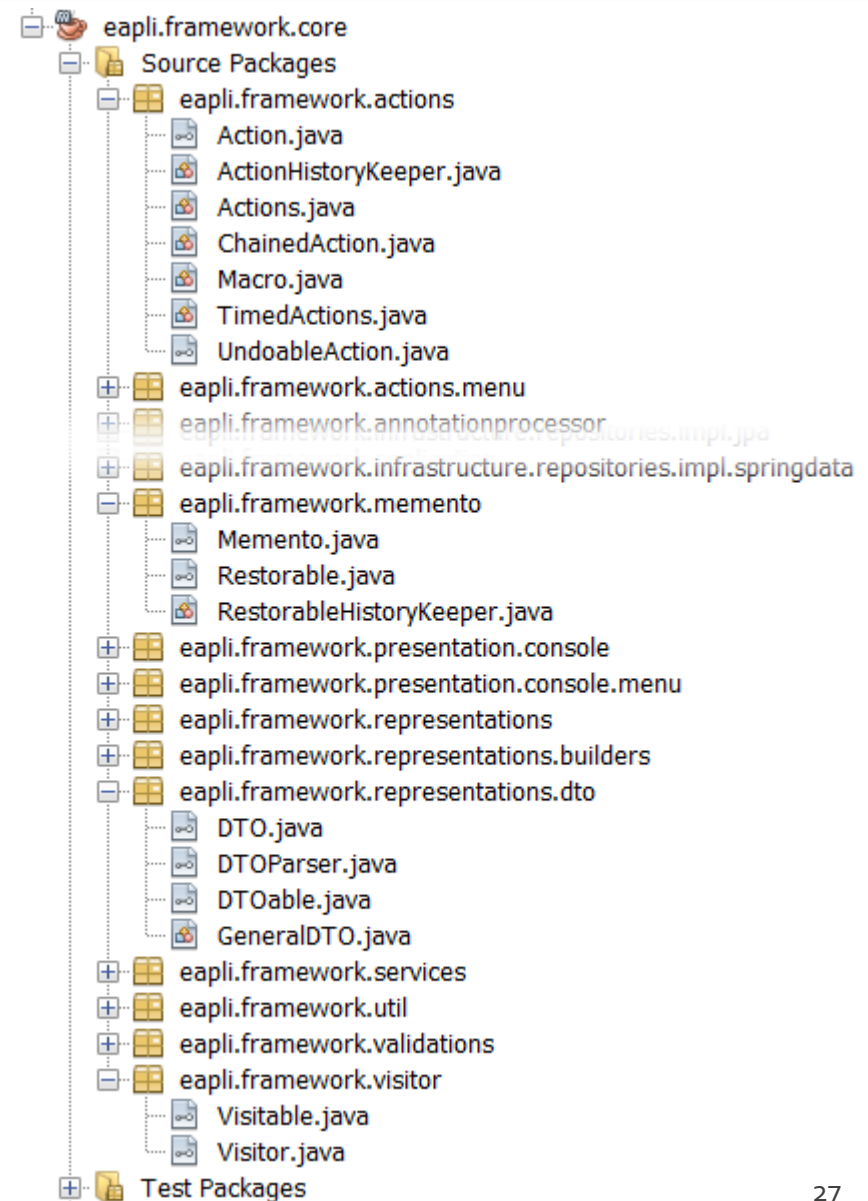
EAPLI framework

Core

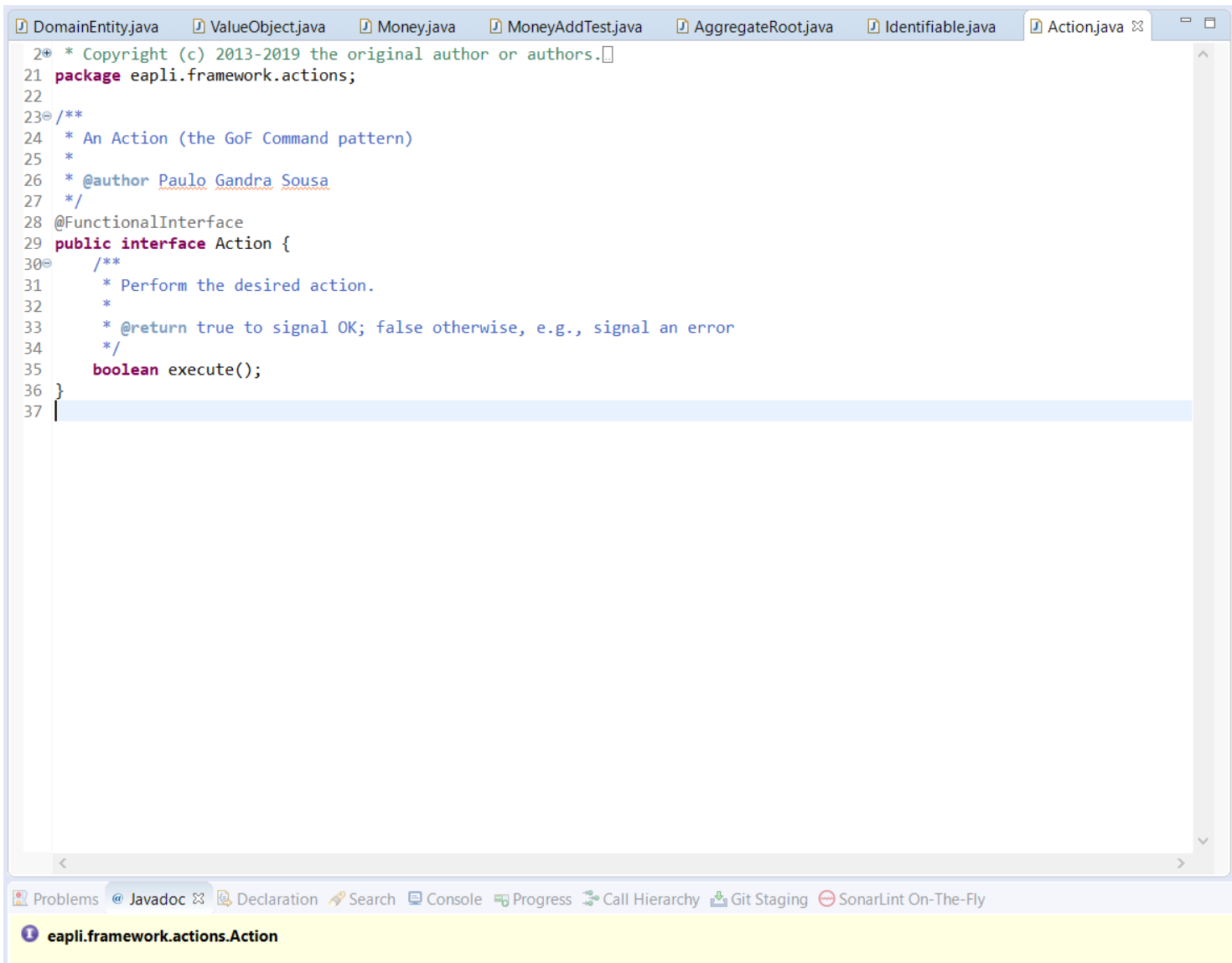
Other parts of the framework

Common patterns

- **Actions**
 - a.k.a Command
- **Memento**
- **Visitor**
- **DTO**



Action.java



```
20 * Copyright (c) 2013-2019 the original author or authors.
21 package eapli.framework.actions;
22
23 /**
24  * An Action (the GoF Command pattern)
25  *
26  * @author Paulo Gandra Sousa
27  */
28 @FunctionalInterface
29 public interface Action {
30     /**
31      * Perform the desired action.
32      *
33      * @return true to signal OK; false otherwise, e.g., signal an error
34      */
35     boolean execute();
36 }
37
```

Problems @ Javadoc Declaration Search Console Progress Call Hierarchy Git Staging SonarLint On-The-Fly

eapli.framework.actions.Action

Actions.java

The screenshot shows an IDE with the `Actions.java` file open. The code is a Java utility class for working with actions. It includes a copyright notice, package declaration, import, and a Javadoc comment. The class `Actions` is a final utility class containing three abstract static classes: `ThrowAction`, `ThrowStateAction`, and `ThrowArgumentAction`. `ThrowAction` has a `message` attribute and an `execute()` method that throws `IllegalStateException`. `ThrowStateAction` and `ThrowArgumentAction` extend `ThrowAction`.

```
20 * Copyright (c) 2013-2019 the original author or authors.
21 package eapli.framework.actions;
22
23 import java.util.function.BooleanSupplier;
24
25 /**
26  * Utility class for working with {@link Action}
27  *
28  * @author Paulo Gandra de Sousa
29  */
30 @Utility
31 public final class Actions {
32
33     private abstract static class ThrowAction implements Action {
34
35         protected final String message;
36
37         public ThrowAction(final String m) {
38             message = m;
39         }
40     }
41
42     private static class ThrowStateAction extends ThrowAction {
43
44         public ThrowStateAction(final String m) {
45             super(m);
46         }
47
48         @Override
49         public boolean execute() {
50             throw new IllegalStateException(message);
51         }
52     }
53
54     private static class ThrowArgumentAction extends ThrowAction {
55
56         public ThrowArgumentAction(final String m) {
```

The Outline view on the right shows the structure of the `eapli.framework.actions` package, including the `Actions` class and its nested classes and methods.

- eapli.framework.actions
 - Actions
 - ThrowAction
 - ThrowArgumentAction
 - ThrowStateAction
 - FAIL : Action
 - SUCCESS : Action
 - THROW_ARGUMENT : Action
 - THROW_STATE : Action
 - dof(Action, boolean) : boolean
 - dof(Action, BooleanSupplier) : boolean
 - dofNot(Action, boolean) : boolean
 - dofNot(Action, BooleanSupplier) : boolean
 - repeat(Action, int) : void
 - retry(Action, int, int) : boolean
 - retry(Action, int, int, boolean) : boolean
 - throwArgument(String) : Action
 - throwState(String) : Action

ActionTest.java

The screenshot shows an IDE with the `ActionsTest.java` file open. The file contains several JUnit tests for the `Actions` class. The tests are as follows:

```
37  */
38  public class ActionsTest {
39
40      @Test
41      public void testSucess() {
42          assertTrue(Actions.SUCCESS.execute());
43      }
44
45      @Test
46      public void testFail() {
47          assertFalse(Actions.FAIL.execute());
48      }
49
50      @Test(expected = IllegalStateException.class)
51      public void testThrowState() {
52          Actions.THROW_STATE.execute();
53      }
54
55      @Test(expected = IllegalStateException.class)
56      public void testThrowStateWithMsg() {
57          Actions.throwState("Test").execute();
58      }
59
60      @Test
61      public void testThrowStateWithMsgHasTheRightMessage() {
62          try {
63              Actions.throwState("Test").execute();
64          } catch (final IllegalStateException e) {
65              assertEquals("Test", e.getMessage());
66              return;
67          }
68          fail("somethign terrible just happened...");
69      }
70
71      @Test(expected = IllegalArgumentException.class)
72      public void testThrowArgument() {
73          Actions.THROW_ARGUMENT.execute();
74      }
75  }
```

The Outline pane on the right shows the structure of the `ActionsTest` class, listing all the test methods and their return types:

- `ensureDolf() : void`
- `ensureDolfDoesnotExecutelfFalse() : void`
- `ensureDolfNot() : void`
- `ensureDolfNotDoesnotExecutelfTrue() : void`
- `ensureRepeat0Times() : void`
- `ensureRepeat10Times() : void`
- `ensureRepeatMinus1Times() : void`
- `ensureRetryFailsIfActionFails() : void`
- `ensureRetryRunsNTimesIfActionFails() : void`
- `ensureRetryWorksAtFirst() : void`
- `ensureRetryWorksIfActionSucceeds() : void`
- `testFail() : void`
- `testSucess() : void`
- `testThrowArgument() : void`
- `testThrowArgumentWithMsg() : void`
- `testThrowArgumentWithMsgHasTheRightMessage() : void`
- `testThrowState() : void`
- `testThrowStateWithMsg() : void`
- `testThrowStateWithMsgHasTheRightMessage() : void`

The bottom status bar shows the current file path: `eapli.framework.actions.ActionsTest`.

Utilities

- **Console**

- Helper console reading functions

- **DateTime**

- Simplifies manipulation of dates and times through java Calendar

- **Files**

- File manipulation helper

- **Arrays**

- **Collections**

- **Comparables**

Utilities

- **HashCode**

- Simplifies generation of a hash code class
- Helper console reading functions

- **NumberGenerator**

- Simple API for Java Random class

- **Strings**

- String manipulation

- **StringMixin**

- **Math**

- Sample math utility

Utilities

- **StringPredicates**

- Predicates to test strings

- **NumberPredicates**

- Predicates to test primitive numbers

- **PreConditions**

- Validates preconditions on methods, usually the method's parameters
- Throws `InvalidArgumentException`

- **Invariants**

- Validates internal consistency of an object
- Throws `InvalidStateException`

Invariants.java

The screenshot shows an IDE with the `Invariants.java` file open. The code defines a utility class for expressing and ensuring class/method invariants. It includes a package declaration, an import, a Javadoc comment, an `@author` tag, an `@Utility` annotation, a private constructor, and two public static methods: `areEqual(long a, long b)` and `areEqual(Object a, Object b, String msg)`. The IDE also displays an outline of the class on the right side.

```
20 * Copyright (c) 2013-2019 the original author or authors.
21 package eapli.framework.validations;
22
23 import java.util.Collection;
24
25 /**
26  * Utility class for expressing and ensuring class/method's invariants. If the
27  * invariant does not hold, an IllegalStateException is thrown.
28  *
29  * @author Paulo Gandra de Sousa
30  */
31 @Utility
32 public final class Invariants {
33     private Invariants() {
34         // ensure utility
35     }
36
37     /**
38      * Ensures two values are equal.
39      *
40      * @param a
41      * @param b
42      * @throws IllegalStateException
43      *         if the assertion is unfulfilled
44      */
45     public static void areEqual(final long a, final long b) {
46         Validations.ensureAreEqual(a, b, Actions.THROW_STATE);
47     }
48
49     /**
50      * Ensures two objects are equal.
51      *
52      * @param a
53      * @param b
54      * @param msg
55      *         the message text to include in the throw exception
56      * @throws IllegalStateException
57      */
58     public static void areEqual(Object a, Object b, String msg) {
59         Validations.ensureAreEqual(a, b, msg, Actions.THROW_STATE);
60     }
61 }
```

Outline:

- eapli.framework.validations
 - Invariants
 - areEqual(long, long) : void
 - areEqual(Object, Object, String) : void
 - ensure(boolean) : void
 - ensure(boolean, String) : void
 - isPositive(long) : void
 - isPositive(long, String) : void
 - matches(Pattern, String, String) : void
 - notEmpty(Collection<?>) : void
 - notEmpty(Collection<?>, String) : void
 - notEmpty(String) : void
 - notEmpty(String, String) : void
 - noneNull(Object...) : void
 - nonNegative(long) : void
 - nonNegative(long, String) : void
 - nonNull(Object) : void
 - nonNull(Object, String) : void
 - Invariants()

Preconditions.java

The screenshot shows an IDE with the `Preconditions.java` file open. The code is a utility class for validating preconditions. It includes a private constructor, a `noneNull` method, and a `nonNegative` method. The `noneNull` method uses `Validations.ensureNonNull` to throw an `IllegalArgumentException` if any argument is null. The `nonNegative` method uses `Validations.ensureNonNegative` to throw an `IllegalArgumentException` if any argument is negative.

```
32 * Utility class for validating preconditions, usually on function parameters.
33 * if the precondition is not met, an IllegalArgumentException is thrown.
34 *
35 * @author Paulo Gandra de Sousa
36 *
37 */
38 @Utility
39 public final class Preconditions {
40     private Preconditions() {
41         // ensure utility
42     }
43
44     /**
45      * Ensures all object references are not null.
46      *
47      * @param arg
48      * @throws IllegalArgumentException
49      *         if the assertion is unfulfilled
50      */
51     public static void noneNull(final Object... arg) {
52         Validations.ensureNonNull(Actions.throwArgument("At least one of the
53             arg);
54     }
55
56     /**
57      * ensures a value is non negative
58      *
59      * @param arg
60      * @throws IllegalArgumentException
61      *         if the assertion is unfulfilled
62      */
63     public static void nonNegative(final long arg) {
64         Validations.ensureNonNegative(arg, Actions.THROW_ARGUMENT);
65     }
66
67     /**
```

The right sidebar shows the Outline view with the following members:

- `eapli.framework.validations`
 - `Preconditions`
 - `areEqual(long, long) : void`
 - `areEqual(Object, Object, String) : void`
 - `ensure(boolean) : void`
 - `ensure(boolean, String) : void`
 - `isPositive(long) : void`
 - `isPositive(long, String) : void`
 - `matches(Pattern, String, String) : void`
 - `notEmpty(Collection<?>) : void`
 - `notEmpty(Collection<?>, String) : void`
 - `notEmpty(String) : void`
 - `notEmpty(String, String) : void`
 - `noneNull(Object...) : void`
 - `nonNegative(long) : void`
 - `nonNegative(long, String) : void`
 - `nonNull(Object) : void`
 - `nonNull(Object, String) : void`
 - `Preconditions()`

The bottom status bar shows the text: "Members calling 'notEmpty(Collection<?>)' - in workspace".

Example

The screenshot shows an IDE with the following components:

- Editors:** Invariants.java, InvariantsTe..., Preconditio..., Dish.java (active), and SonarLint Rule Descripti... are open.
- Code (Dish.java):**

```
60
61 @EmbeddedId
62 private Designation name;
63 /**
64  * cascade = CascadeType.NONE as the dishType is part of another aggregate
65  */
66 @ManyToOne()
67 private DishType dishType;
68 private NutricionalInfo nutricionalInfo;
69 private Money price;
70 private boolean active;
71
72 public Dish(final DishType dishType, final Designation name,
73             final NutricionalInfo nutricionalInfo, final Money price) {
74     Preconditions.checkNotNull(dishType, name, nutricionalInfo);
75
76     this.dishType = dishType;
77     this.name = name;
78     this.nutricionalInfo = nutricionalInfo;
79     this.setPrice(price);
80     this.active = true;
81 }
82
83 public Dish(final DishType dishType, final Designation name, final Money price) {
84     Preconditions.checkNotNull(dishType, name, price);
85
86     this.dishType = dishType;
87     this.name = name;
88     this.nutricionalInfo = null;
89     this.price = price;
```
- Outline:** Shows the package `eapli.ecafeteria.dishmanagement.domain` and the `Dish` class with its fields and methods.
- Members calling 'noneNull(Object...)' - in workspace:**

Member	Line	Call
<code>CardMovement(MovementType, Money, CafeteriaUser, Calendar)</code> - <code>eapli.e...</code>		
<code>ChainedAction(Action, Action)</code> - <code>eapli.framework.actions.ChainedAction</code>		
<code>CreditRecharge(CafeteriaUser, Money, Calendar)</code> - <code>eapli.ecafeteria.sales.d...</code>		
<code>Dish(DishType, Designation, Money)</code> - <code>eapli.ecafeteria.dishmanagement.d...</code>		
<code>Dish(DishType, Designation, NutricionalInfo, Money)</code> - <code>eapli.ecafeteria.dish...</code>	74	<code>noneNull(dishType, name, nutricionalInfo)</code>
<code>encodedAndValid(String, PasswordPolicy, PasswordEncoder)</code> - <code>Optional<Pa...</code>		
<code>init(Username, Password, Name, EmailAddress, RoleSet, Calendar)</code> - <code>void</code>		