Software Engineering 2

Deadline - Report

Work Group 304

01504922 SIGMUND Daniel
01426012 NISSLMÜLLER Utz
01363460 RAMHARTER Alexander

Technology Stack

Programming languages	Java 1.8
IDE	IntelliJ
Build Tool	Maven
Unit Testing	JUnit Jupiter - JUnit5
Network Communication	Spring Boot Framework v2.0.5
VCS	GitLab
Client-Side Programming	Vue.js Framework (HTML, CSS, Javascript)
Additional Frameworks	Batik Transcoder v1.6.1 (file format conversion for download) ApiDocjs Framework for creating API specification (apidocjs.com)
Additional tools	“Analyze” - in Intellij “SpotBugs” - Plugin for Maven

Design Patterns

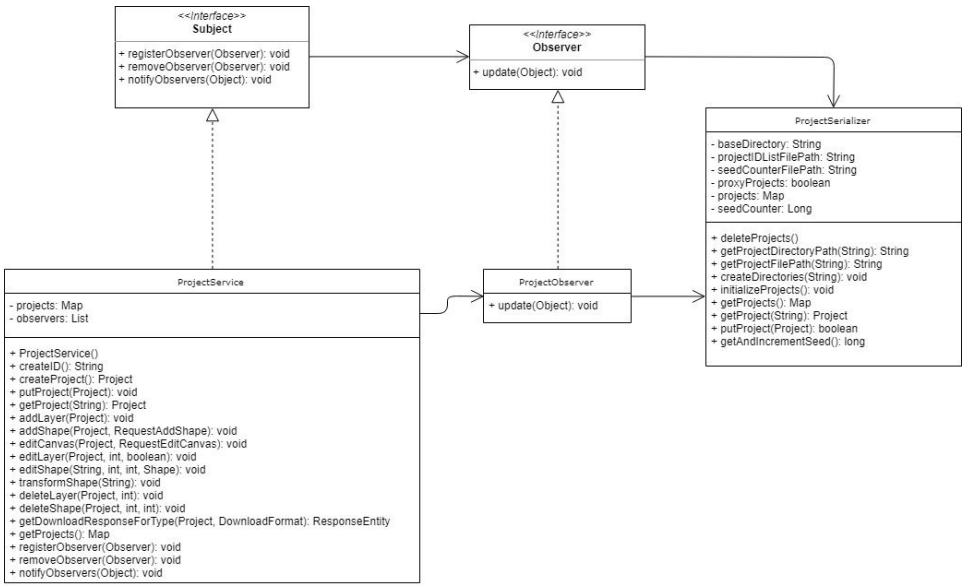
Note: All UML-Diagrams are also available in higher resolution in: “designs/patterns/”

Observer Pattern

We have created an interface named `Subject` and an interface named `Observer` in the `observer` package. They define the basic functionalities needed by our observers and subjects. The concrete classes implementing `Subject` and `Observer` are `main.input.ProjectService` and `persistence.ProjectObserver`, respectively.

`ProjectService` is the class that implements all the functionality the API offers, and is thus very closely linked to `RESTHandler`. It stores the projectIDs and the `Project` instances linked to them in a `Map` and is able to modify a `Project` with its various functions. We have decided to make it observable so we could monitor changes to each `Project` without having to worry about which classes actually need to know about said changes (other than having to register them as observers).

The only concrete `Observer` we implemented is `ProjectObserver`, a class that has the simple task of sending any updated `Project` to the `ProjectSerializer` class, which is responsible for serializing and deserializing `Projects`. Naturally, the persistent data needs to be updated whenever a `Project` managed in `ProjectService` changes, which is why we have decided to use the observer pattern for it. Despite currently only having one concrete observer, we think that this is a valid use of the pattern, in case we decide to extend our functionality.



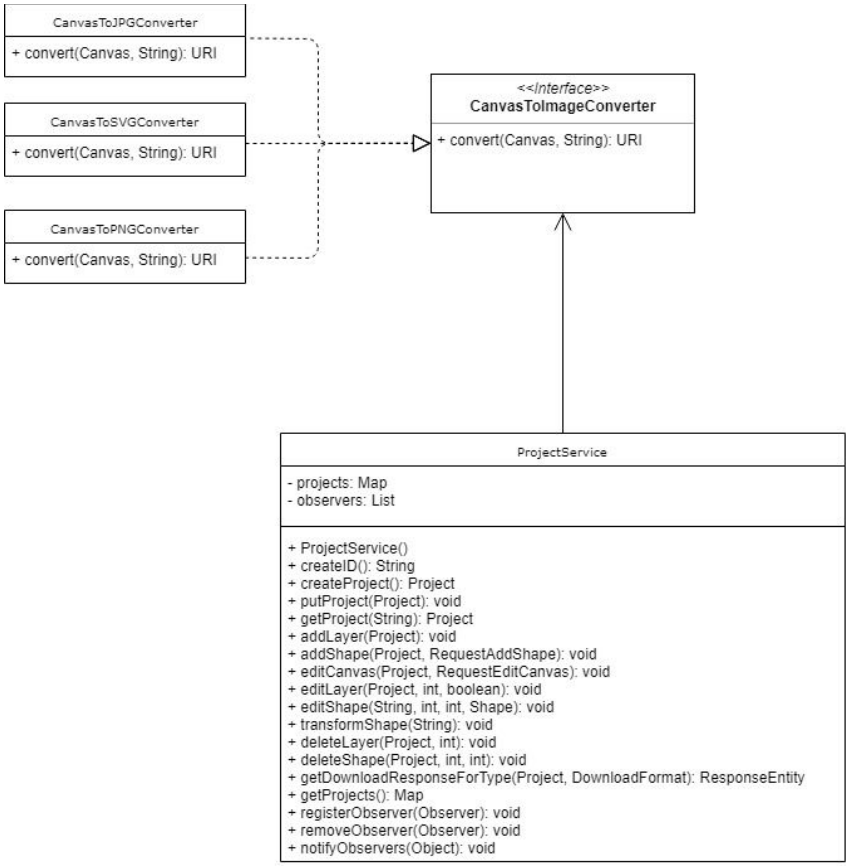
Strategy Pattern

Being one of the functional requirements of the implementation, the ability to download the canvas in SVG format proved to be extensible by using the strategy pattern.

We created a functional interface `CanvasToImageConverter` with the function `java.net.URI convert(Canvas canvas, String projectID)` and three classes implementing the interface: `CanvasToSVGConverter`, `CanvasToPNGConverter` and `CanvasToJPGConverter`.

Each of the classes does exactly what its name implies. Each of them extracts the SVG code from the `Canvas` and stores it in the `projects/<projectID>` directory. In the case of `CanvasToPNGConverter` or `CanvasToJPGConverter`, it then uses the Batik API to convert the SVG file to a PNG or JPG file respectively. It then returns the URI of the converted file to be processed for download.

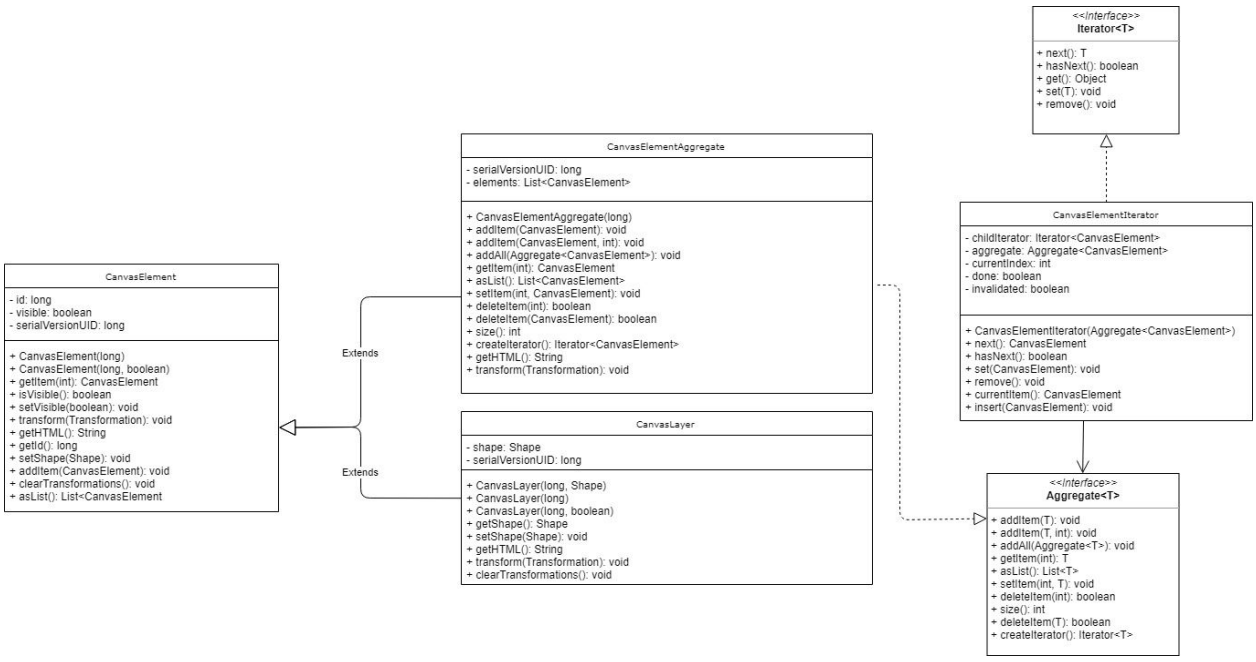
The strategy is determined dynamically, depending on the format the API receives in the download URL (see API specification).



Iterator Pattern

The implementation we chose for the iterator pattern is closely tied to the one for the composite pattern. Since the composite pattern, with its polymorphic lists, created a structure that would be difficult to iterate through, we decided that we needed an iterator to do so more easily. Furthermore, the iterator would enforce a key programming principle: program to an interface, not to a specific class! It will also allow us to leave the dependent classes intact, should we decide to store our layers and shapes differently.

As the “composite” part of `CanvasElement` can hold many `CanvasElements` itself, we created the generic `Aggregate` interface, which is implemented by `CanvasElementAggregate`. We then made another generic interface, `Iterator`, to traverse an `Aggregate` of its type. For the concrete case, we named it `CanvasElementIterator` and used a structure of parent and child iterators to traverse the entire `CanvasElementAggregate`. Our iterator has several additional functions in addition to those, which are mentioned in most textbook depictions of this pattern. This includes the ability to read, set and delete the current item - the first two of which will trigger a boolean flag which invalidates the iterator, since integrity is not guaranteed after these operations.We found it useful to implement these methods directly in our `CanvasElementIterator`, since they are called fairly often in tandem.



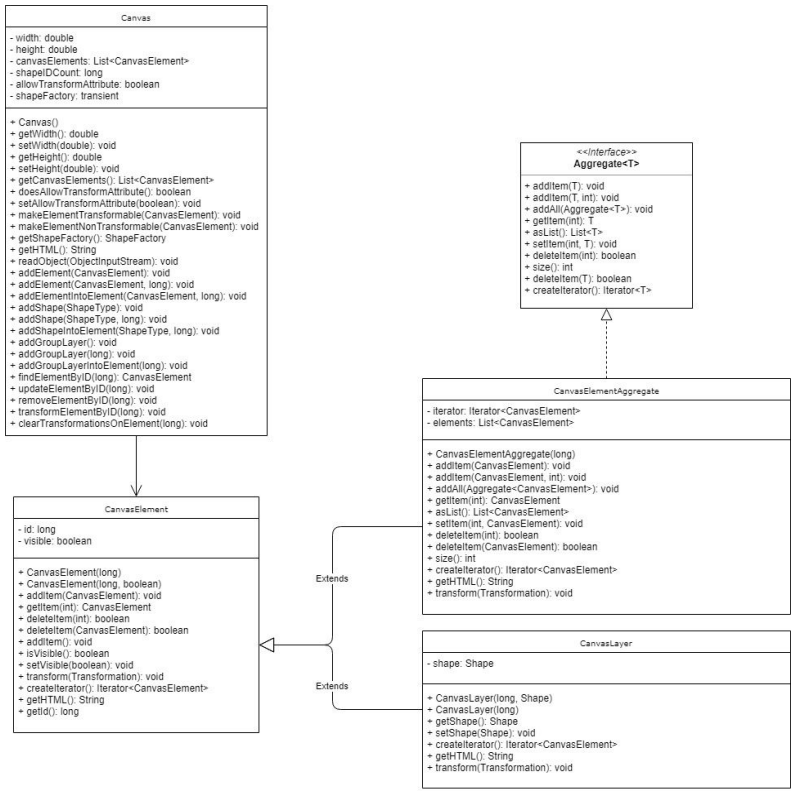
Composite Pattern

Since one of the requirements was to implement the composite pattern, we thought it would be a good idea to use the composite pattern to redo the one-level canvas–layer–shape hierarchy we implemented at first.

Hence, we deleted the old `Layer` class and created an abstract one called `CanvasElement`, with the concrete `CanvasElementAggregate` and `CanvasLayer` subclasses. The `Canvas` has a `List` of `CanvasElements`.

`CanvasElementAggregate` implements the interface `Aggregate<CanvasElement>` and holds a `List` of `CanvasElements`, which can again be instances of either of the subclasses. `CanvasLayer` holds the actual `Shape`. This allows us to create a nested structure of layers and layer groups with arbitrary depth.

We used `CanvasElement` in most of the use cases in order to make use of the polymorphic possibilities, which this pattern provides. It provides a tree structure for the iterator to iterate through, and to dynamically make decisions, based on which class he is iterating through. To make full use of this, we created several methods in `CanvasElement` which are only implemented in one subclass - the call of the non-overridden method will throw an `UnsupportedActionException`.

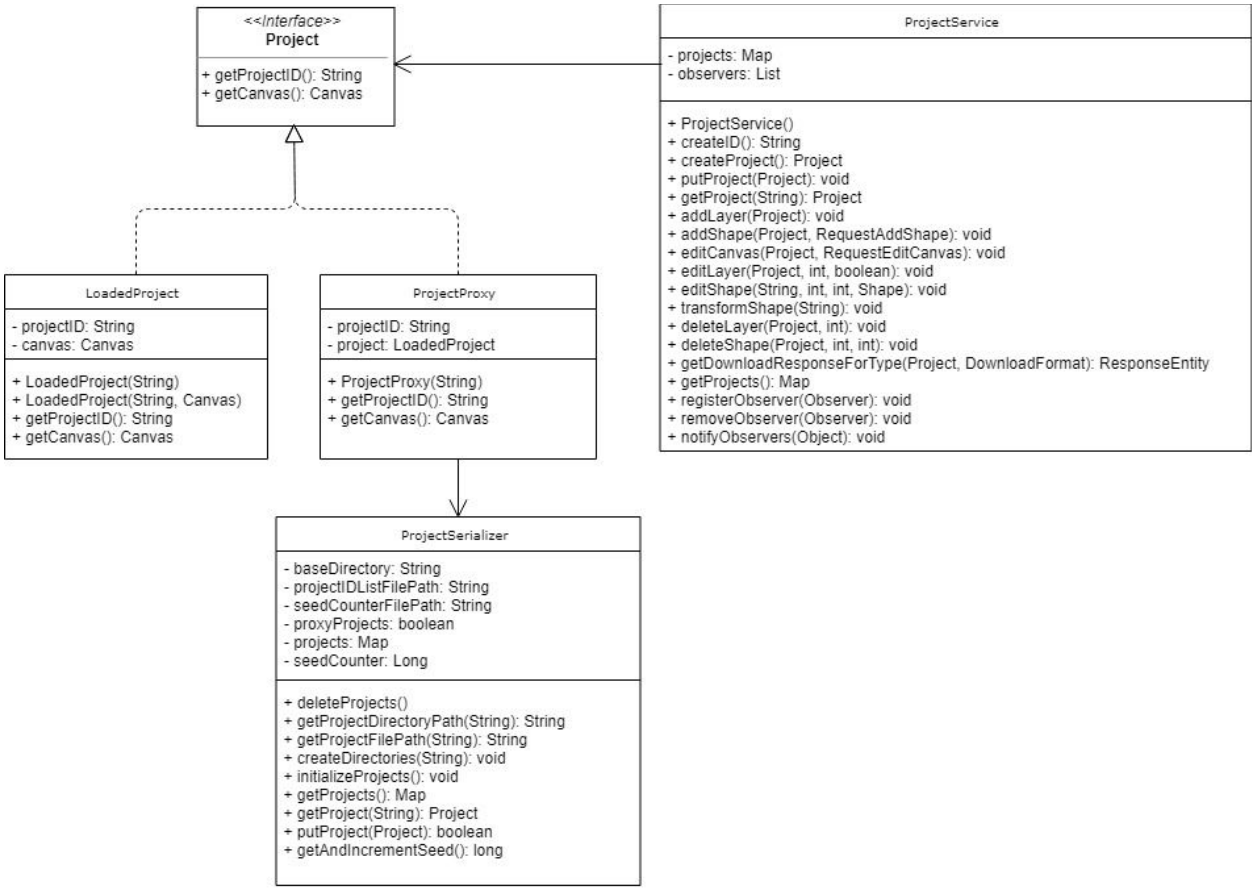


Proxy Pattern

Because we decided to persistently store all projects, we decided that having all of them in memory at all times would make it scale very badly as the number of created projects increases. For that reason, we applied the proxy pattern in a way that would decrease that overhead.

We implemented a `ProjectProxy` class that acts like a normal `Project` does, but does not actually hold any information besides the project ID. If the canvas information is requested during runtime, it fetches the corresponding `LoadedProject` (that does hold all the information) from the `ProjectSerializer` and delegates the function call to it.

Currently, fetching only happens once and the project stays loaded after that. Future improvements could deal with unloading projects using timeouts.



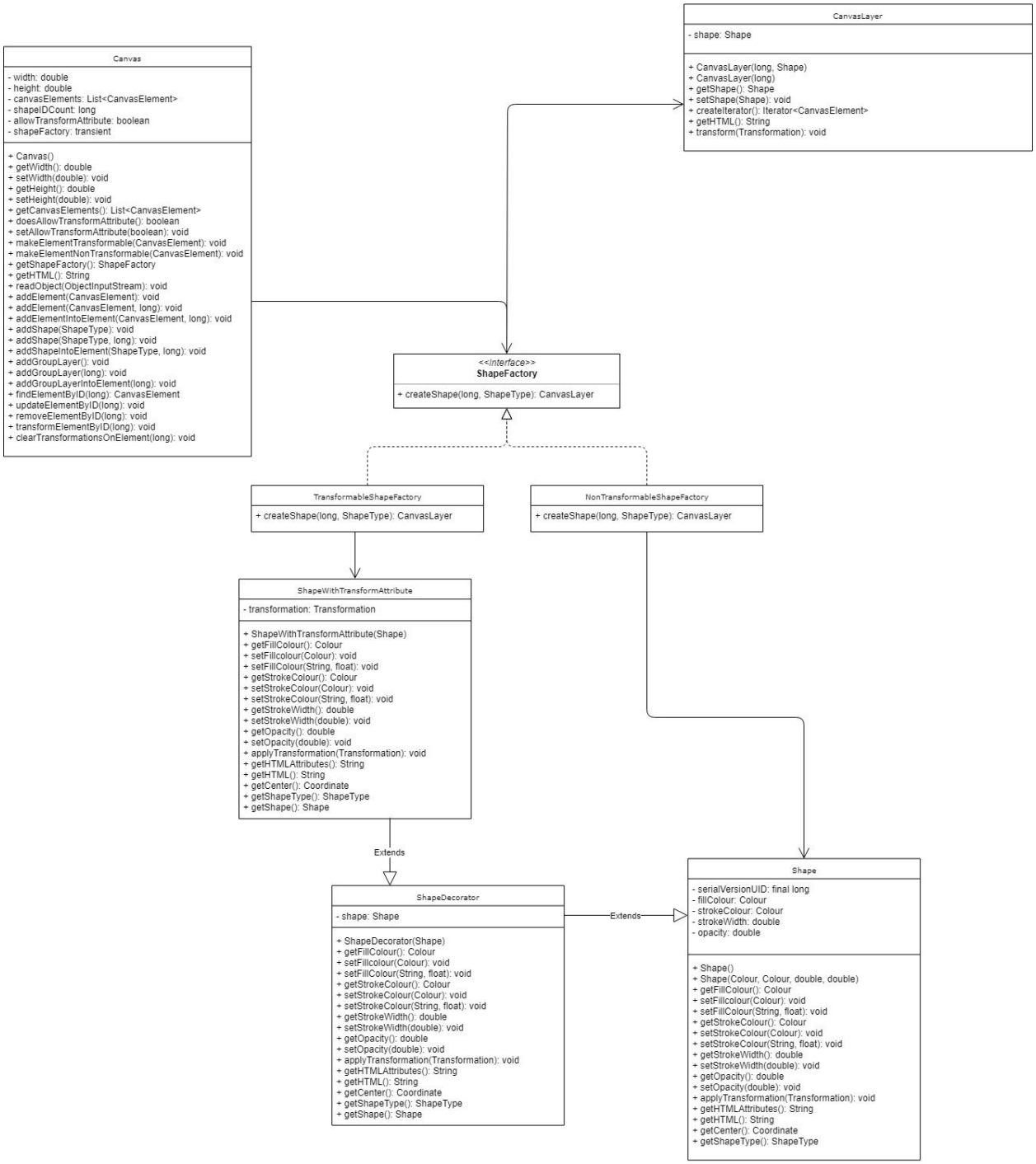
Factory / Abstract Factory Pattern

We decided that a factory was most suitable to creating the various `Shape` instances. For that reason, we created an abstract interface `ShapeFactory`, that, when called, directly returns a `CanvasLayer` containing a `Shape` of the desired type rather than the `Shape` itself..

When the `ProjectService` receives a request to add a new `Shape` of a certain type to the canvas, it calls the `addShape` method of the project's `Canvas` instance. `Canvas` then uses its `ShapeFactory` instance to create a `CanvasLayer` with a `Shape` of the requested type.

Arguably the most arbitrary design decision throughout our project happened here, as we were looking for a way to extend the idea to an abstract factory. We decided to base the concrete implementations of `ShapeFactory` on whether to allow the use of the SVG `transform` attribute in the project or not (since we were not sure about browser support). The result were the concrete classes `TransformableShapeFactory` and `NonTransformableShapeFactory`.

The difference between the two concrete factories is hopefully mostly obvious by their names, although "non-transformable" can be misleading in a global context, since the way we implemented them does still allow them to "transform", but only to the extent that their attributes allow without having to append a `transform` attribute in the SVG. For example, a `Circle` has a `center` attribute, which makes it easy to move around, but skewing it requires a `transform` attribute.


Canvas
- width: double
- height: double
- canvasElements: List<CanvasElement>
- shapeIDCount: long
- allowTransformAttribute: boolean
- shapeFactory: transient
+ Canvas()
+ getWidth(): double
+ setWidth(double): void
+ getHeight(): double
+ setHeight(double): void
+ getCanvasElements(): List<CanvasElement>
+ doesAllowTransformAttribute(): boolean
+ setAllowTransformAttribute(boolean): void
+ makeElementTransformable(CanvasElement): void
+ makeElementNonTransformable(CanvasElement): void
+ getShapeFactory(): ShapeFactory
+ getHTML(): String
+ readObject(ObjectInputStream): void
+ addElement(CanvasElement): void
+ addElement(CanvasElement, long): void
+ addElementIntoElement(CanvasElement, long): void
+ addShape(ShapeType): void
+ addShape(ShapeType, long): void
+ addShapeIntoElement(ShapeType, long): void
+ addGroupLayer(): void
+ addGroupLayer(long): void
+ addGroupLayerIntoElement(long): void
+ findElementByID(long): CanvasElement
+ updateElementByID(long): void
+ removeElementByID(long): void
+ transformElementByID(long): void
+ clearTransformationsOnElement(long): void
CanvasLayer
- shape: Shape
+ CanvasLayer(long, Shape)
+ CanvasLayer(long)
+ getShape(): Shape
+ setShape(Shape): void
+ createIterator(): Iterator<CanvasElement>
+ getHTML(): String
+ transform(Transformation): void
<<Interface>>
ShapeFactory
+ createShape(long, ShapeType): CanvasLayer
TransformableShapeFactory
+ createShape(long, ShapeType): CanvasLayer
NonTransformableShapeFactory
+ createShape(long, ShapeType): CanvasLayer
ShapeWithTransformAttribute
- transformation: Transformation
+ ShapeWithTransformAttribute(Shape)
+ getFillColour(): Colour
+ setFillcolour(Colour): void
+ setFillColour(String, float): void
+ getStrokeColour(): Colour
+ setStrokeColour(Colour): void
+ setStrokeColour(String, float): void
+ getStrokeWidth(): double
+ setStrokeWidth(double): void
+ getOpacity(): double
+ setOpacity(double): void
+ applyTransformation(Transformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ getCenter(): Coordinate
+ getShapeType(): ShapeType
+ getShape(): Shape
Extends
ShapeDecorator
- shape: Shape
+ ShapeDecorator(Shape)
+ getFillColour(): Colour
+ setFillcolour(Colour): void
+ setFillColour(String, float): void
+ getStrokeColour(): Colour
+ setStrokeColour(Colour): void
+ setStrokeColour(String, float): void
+ getStrokeWidth(): double
+ setStrokeWidth(double): void
+ getOpacity(): double
+ setOpacity(double): void
+ applyTransformation(Transformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ getCenter(): Coordinate
+ getShapeType(): ShapeType
+ getShape(): Shape
Extends
Shape
- serialVersionUID: final long
- fillColour: Colour
- strokeColour: Colour
- strokeWidth: double
- opacity: double
+ Shape()
+ Shape(Colour, Colour, double, double)
+ getFillColour(): Colour
+ setFillcolour(Colour): void
+ setFillColour(String, float): void
+ getStrokeColour(): Colour
+ setStrokeColour(Colour): void
+ setStrokeColour(String, float): void
+ getStrokeWidth(): double
+ setStrokeWidth(double): void
+ getOpacity(): double
+ setOpacity(double): void
+ applyTransformation(Transformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ getCenter(): Coordinate
+ getShapeType(): ShapeType

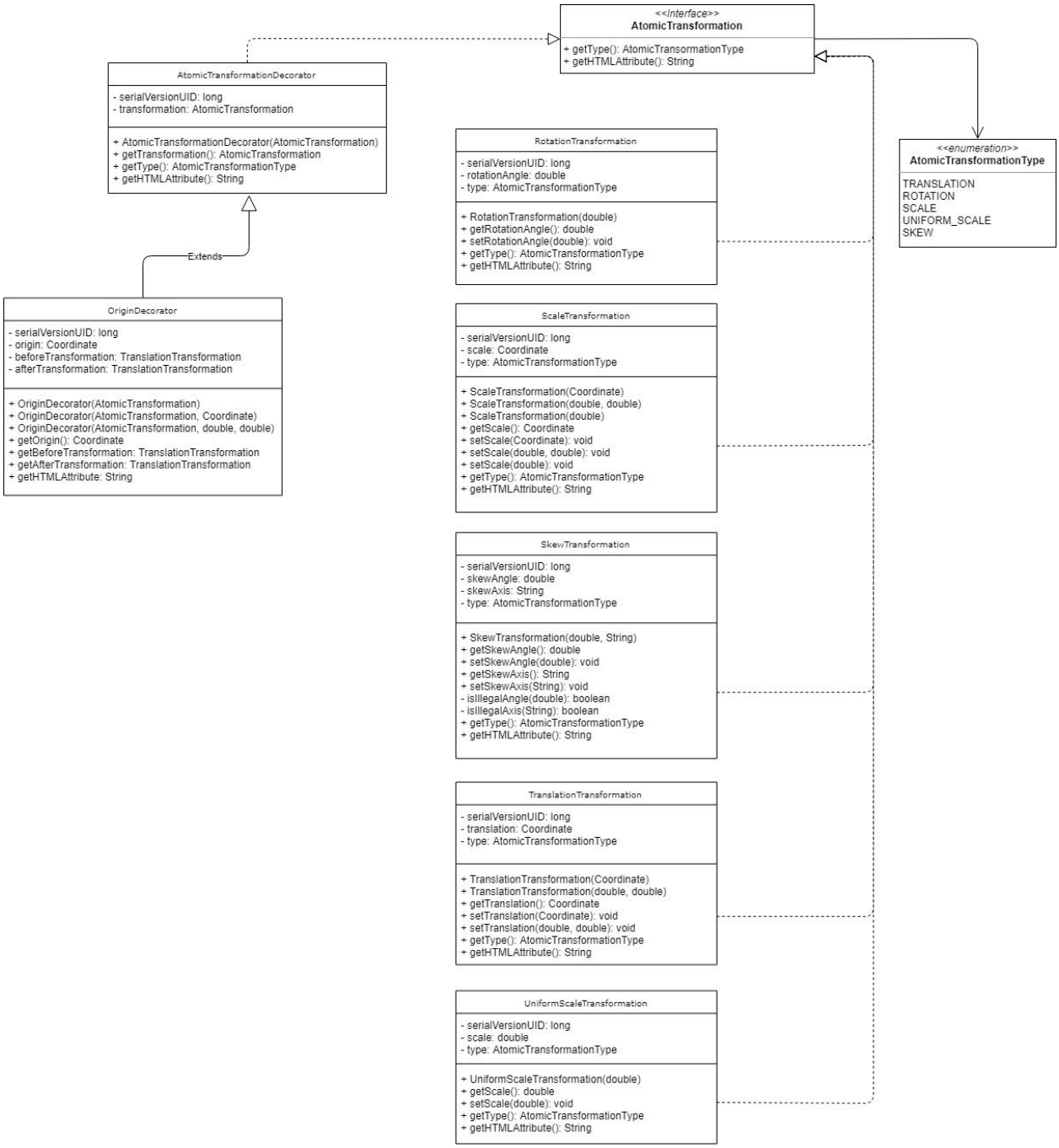
Decorator Pattern 1

We came across our first application for the decorator pattern by circumstance as we were modeling the different kinds of transformations that could be applied to a `Shape` using the SVG `transform` attribute.

As it turned out, the attribute performs all transformations that require a point of reference by using the point (0 | 0) as default, and not all transformations allow changing that point. Since the order of transformations is fixed, we came up with a workaround (in order to be able to e.g. rotate a shape around its own center).

We created the interface `AtomicTransformation`, which represents any transformation that should be executed completely or not at all. Besides all the transformations supported by SVG, we also created an `AtomicTransformationDecorator` implementing it. The only “real” decorator we implemented is the workaround described above, `OriginDecorator`.

`OriginDecorator` wraps any `AtomicTransformation` and sandwiches it between two translate (move) transformations. The result is the same as changing the point of reference for the original transformation. Similar chaining of transformations could be achieved by other decorators.

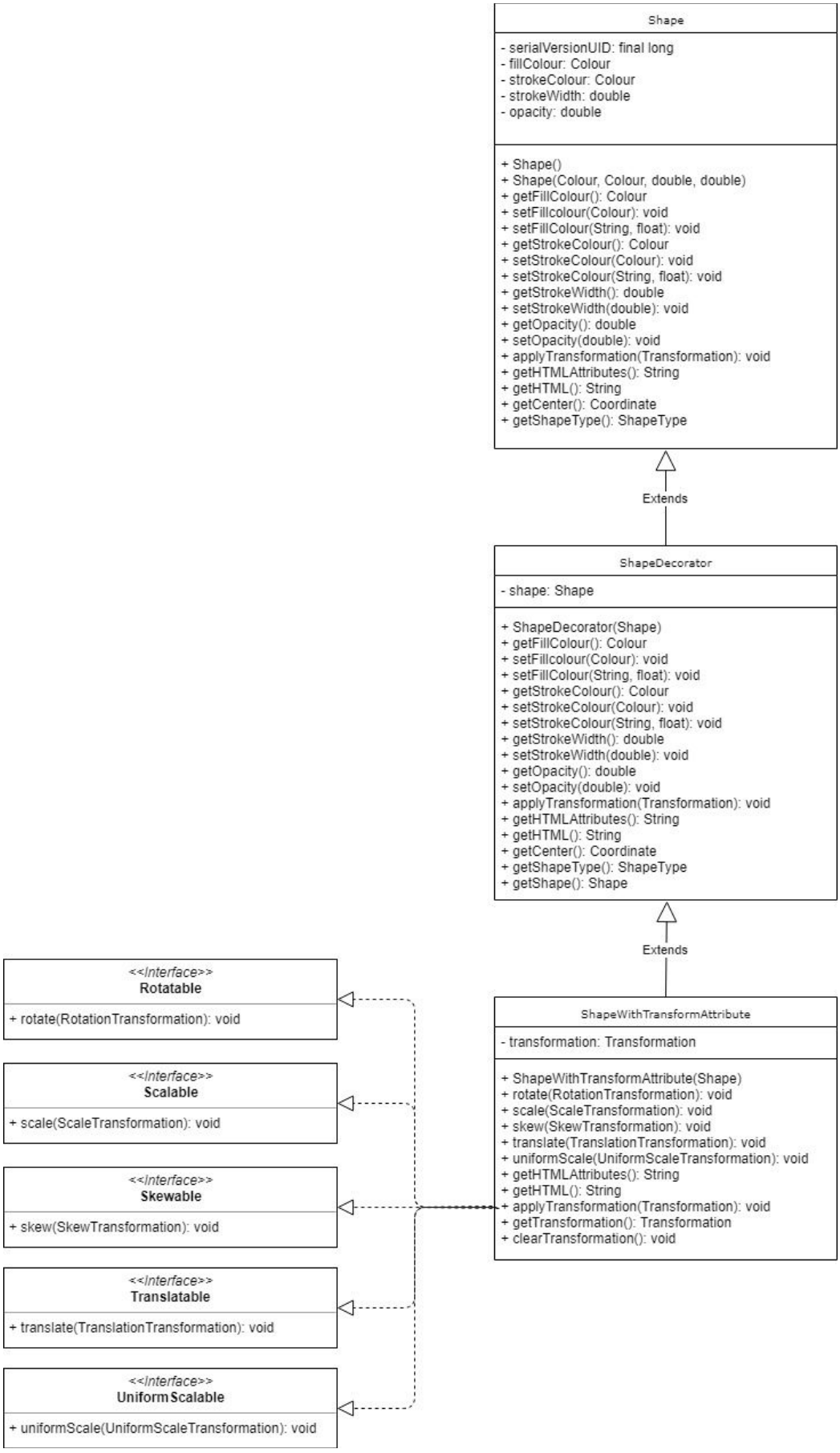

<<Interface>>
AtomicTransformation
+ getType(): AtomicTransormationType
+ getHTMLAttribute(): String
AtomicTransformationDecorator
- serialVersionUID: long
- transformation: AtomicTransformation
+ AtomicTransformationDecorator(AtomicTransformation)
+ getTransformation(): AtomicTransformation
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
Extends
OriginDecorator
- serialVersionUID: long
- origin: Coordinate
- beforeTransformation: TranslationTransformation
- afterTransformation: TranslationTransformation
+ OriginDecorator(AtomicTransformation)
+ OriginDecorator(AtomicTransformation, Coordinate)
+ OriginDecorator(AtomicTransformation, double, double)
+ getOrigin(): Coordinate
+ getBeforeTransformation: TranslationTransformation
+ getAfterTransformation: TranslationTransformation
+ getHTMLAttribute: String
RotationTransformation
- serialVersionUID: long
- rotationAngle: double
- type: AtomicTransformationType
+ RotationTransformation(double)
+ getRotationAngle(): double
+ setRotationAngle(double): void
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
ScaleTransformation
- serialVersionUID: long
- scale: Coordinate
- type: AtomicTransformationType
+ ScaleTransformation(Coordinate)
+ ScaleTransformation(double, double)
+ ScaleTransformation(double)
+ getScale(): Coordinate
+ setScale(Coordinate): void
+ setScale(double, double): void
+ setScale(double): void
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
SkewTransformation
- serialVersionUID: long
- skewAngle: double
- skewAxis: String
- type: AtomicTransformationType
+ SkewTransformation(double, String)
+ getSkewAngle(): double
+ setSkewAngle(double): void
+ getSkewAxis(): String
+ setSkewAxis(String): void
- isIllegalAngle(double): boolean
- isIllegalAxis(String): boolean
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
TranslationTransformation
- serialVersionUID: long
- translation: Coordinate
- type: AtomicTransformationType
+ TranslationTransformation(Coordinate)
+ TranslationTransformation(double, double)
+ getTranslation(): Coordinate
+ setTranslation(Coordinate): void
+ setTranslation(double, double): void
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
UniformScaleTransformation
- serialVersionUID: long
- scale: double
- type: AtomicTransformationType
+ UniformScaleTransformation(double)
+ getScale(): double
+ setScale(double): void
+ getType(): AtomicTransformationType
+ getHTMLAttribute(): String
<<enumeration>>
AtomicTransformationType
TRANSLATION
ROTATION
SCALE
UNIFORM_SCALE
SKEW

Decorator Pattern 2

Our second implementation of the decorator pattern is closely tied to the factory pattern. At first, we gave each (non-transformable) `Shape` a transformable counterpart. What we ended up with was a lot of duplicate code and close to no difference from one class to the other. Adding to that, we knew that at runtime we would have to (in the worst case) convert shapes back and forth between transformable and non-transformable as the project properties were being changed. This lead to our rather unconventional abstract factory that uses a decorator instead of inheritance.

We created the class `ShapeDecorator` extending `Shape`. `ShapeWithTransformAttribute` extends the decorator and also implements all five transformation interfaces: `Rotatable`, `Scalable`, `Skewable`, `Translatable` and `UniformScalable` (these basically signal whether a certain kind of transformation can be applied to the shape).

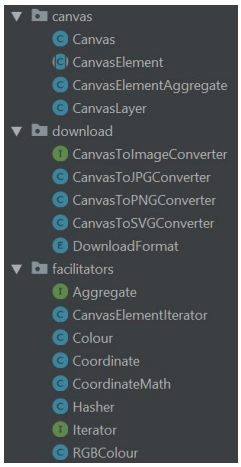
`ShapeWithTransformAttribute` adds a `Transformation` instance, which can hold any number of `AtomicTransformations`, to a regular shape. If a transformation cannot be applied directly to the shape, it is instead stored in that variable and put into a `transform` attribute when parsing to SVG.


Shape
- serialVersionUID: final long
- fillColour: Colour
- strokeColour: Colour
- strokeWidth: double
- opacity: double
+ Shape()
+ Shape(Colour, Colour, double, double)
+ getFillColour(): Colour
+ setFillcolour(Colour): void
+ setFillColour(String, float): void
+ getStrokeColour(): Colour
+ setStrokeColour(Colour): void
+ setStrokeColour(String, float): void
+ getStrokeWidth(): double
+ setStrokeWidth(double): void
+ getOpacity(): double
+ setOpacity(double): void
+ applyTransformation(Transformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ getCenter(): Coordinate
+ getShapeType(): ShapeType
Extends
ShapeDecorator
- shape: Shape
+ ShapeDecorator(Shape)
+ getFillColour(): Colour
+ setFillcolour(Colour): void
+ setFillColour(String, float): void
+ getStrokeColour(): Colour
+ setStrokeColour(Colour): void
+ setStrokeColour(String, float): void
+ getStrokeWidth(): double
+ setStrokeWidth(double): void
+ getOpacity(): double
+ setOpacity(double): void
+ applyTransformation(Transformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ getCenter(): Coordinate
+ getShapeType(): ShapeType
+ getShape(): Shape
Extends
ShapeWithTransformAttribute
- transformation: Transformation
+ ShapeWithTransformAttribute(Shape)
+ rotate(RotationTransformation): void
+ scale(ScaleTransformation): void
+ skew(SkewTransformation): void
+ translate(TranslationTransformation): void
+ uniformScale(UniformScaleTransformation): void
+ getHTMLAttributes(): String
+ getHTML(): String
+ applyTransformation(Transformation): void
+ getTransformation(): Transformation
+ clearTransformation(): void
<<Interface>>
Rotatable
+ rotate(RotationTransformation): void
<<Interface>>
Scalable
+ scale(ScaleTransformation): void
<<Interface>>
Skewable
+ skew(SkewTransformation): void
<<Interface>>
Translatable
+ translate(TranslationTransformation): void
<<Interface>>
Uniform Scalable
+ uniformScale(UniformScaleTransformation): void

Coding Practices

Naming Convention

We think that we did a pretty good job with naming in our project. First of all we persistently used “CamelCase” for naming variables, methods and classes throughout the whole project.



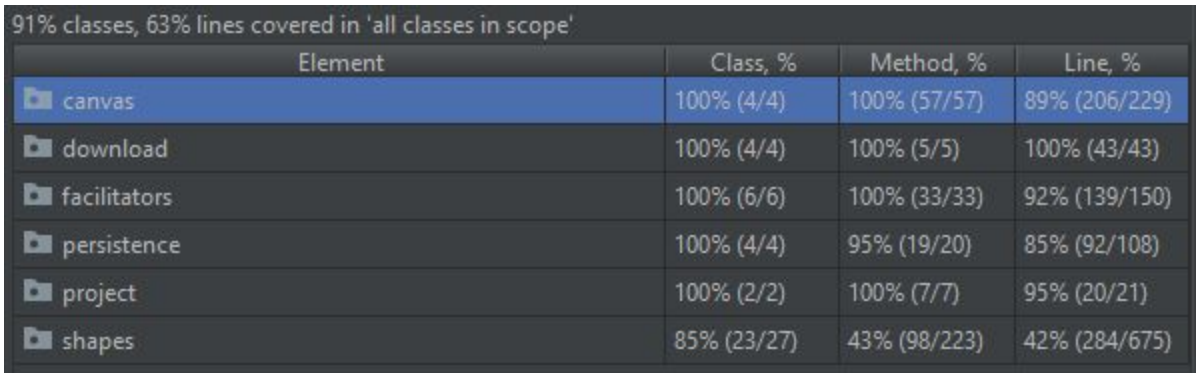

```
public CanvasElement findElementByID(long id) {
  for (CanvasElement element : canvasElements) {
    if (element.getId() == id) {
      return element;
    } else if (element instanceof CanvasElementAggregate) {
      Iterator<CanvasElement> iterator = ((CanvasElementAggregate) element).createIterator();
      while (iterator.hasNext()) {
        CanvasElement currentElement = iterator.next();
        if (currentElement.getId() == id) {
          return currentElement;
        }
      }
    }
  }
  return null;
}
```

We knew that in a project like ours, where multiple developers work on the same code and more importantly have to read each other’s classes and methods, it is very important to have meaningful names, so that everybody knows the exact purpose of a variable or function.

Fortunately, we believe that all of us adhered as strictly as possible to naming conventions to avoid confusion. The names we chose were descriptive, pronounceable and easy to understand. From time to time, we did some refactoring of names to be more precise where we saw fit, but we never really ran into the situation where we became confused by the naming of a variable or function in the project.

Unit testing

As part of the quality requirements, we used the JUnit 5 testing framework to assure the correctness of our classes and functions. We achieved a high coverage of classes and methods throughout our project, testing methods with multiple parameter variations in order to weed out bugs, which were not clearly apparent to us from the beginning. This proved to be especially useful for our Iterator pattern, which needed several rounds of refinement, until the tests passed successfully and we could start using it within our client. We didn't focus too much on shape testing, since this was done in the web client directly by Alexander.



91% classes, 63% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
canvas	100% (4/4)	100% (57/57)	89% (206/229)
download	100% (4/4)	100% (5/5)	100% (43/43)
facilitators	100% (6/6)	100% (33/33)	92% (139/150)
persistence	100% (4/4)	95% (19/20)	85% (92/108)
project	100% (2/2)	100% (7/7)	95% (20/21)
shapes	85% (23/27)	43% (98/223)	42% (284/675)

Comments

Even though our code stand on its own we commented every class and method using javadoc style, trying to give additional information about how and when to use each function. Where necessary, we also added comments within functions to explain the reasoning for certain ways of doing what we were doing.

```
/**
 * Adds a {@link Shape} of the specified type to the specified {@link Project}.
 *
 * @param project The Project to add the Shape to.
 * @param request The {@link RequestAddShape} object containing the values for the modification.
 * @throws IllegalArgumentException If the shape type could not be resolved or the element ID
 * after which to insert the Shape does not exist.
 */
public void addShape(Project project, RequestAddShape request)
    throws IllegalArgumentException, IndexOutOfBoundsException {
```

Formatting

Throughout the whole project we used the GoogleStyle - Code Style. Some of the commits throughout the project were purely reformatting ones.

Operations

In every case where it made sense, we created a separate operation to reduce complexity and increase the simplicity of finding and fixing bugs. Wherever we encountered code that could be separated, we built a function for it. Cohesion was one of the key aspects we paid attention to, as we made sure that each function does the thing it is supposed to do entirely and exclusively.

```
public CanvasElement findElementByID(long id) {
  for (CanvasElement element : canvasElements) {
    if (element.getId() == id) {
      return element;
    } else if (element instanceof CanvasElementAggregate) {
      Iterator<CanvasElement> iterator = ((CanvasElementAggregate) element).createIterator();
      while (iterator.hasNext()) {
        CanvasElement currentElement = iterator.next();
        if (currentElement.getId() == id) {
          return currentElement;
        }
      }
    }
  }
  return null;
}
```

```
public void transformElementByID(long id, Transformation transformation) {
  findElementByID(id).transform(transformation);
}
```

Defensive Programming

Assertions

We used assertions rarely, but there were some places where we decided that we needed to assert that we did not make any mistakes in our assumptions. On the other hand, we did use a lot of them in our JUnit tests (as to be expected).

```
case ROTATION:
  assert (shape instanceof Rotatable
      && atomicTransformation instanceof RotationTransformation);
  ((Rotatable) shape).rotate((RotationTransformation) atomicTransformation);
  break;
```

```
testIDList.setIdList(idStringList);
assertEquals(idStringList, testIDList.getIdList());
persistenceTestLogger.info("IDList setting test passed! Starting IDList serialization test.");
```

Logging

The “main” logging classes of our project are the ones that deal with processing I/O data and events (`RESTHandler`, `ProjectService` and `ProjectSerializer`). The most structured approach to logging is in `ProjectService`, where most functions have logging events for both success and failure.

```
public void editElement(Project project, RequestEditElement request) {

  String operationToLog = "Project " + project.getProjectID() + ": Set Element "
      + request.getElementID() + " " + (request.isVisible() ? "visible" : "invisible");

  Canvas projectCanvas;
  try {
    projectCanvas = project.getCanvas();
  } catch (Exception e) {
    projectServiceLogger.error("Operation failed: " + operationToLog);
    throw new RuntimeException(e);
  }

  projectCanvas.findElementByID(request.getElementID()).setVisible(request.isVisible());

  putProject(project);
  projectServiceLogger.info("Operation successful: " + operationToLog);

}
```

Exceptions

In general we followed the principle “garbage in - error message out”. Depending on the level of abstraction, there were however some cases in which we found “garbage in - nothing out” to be the better approach. In addition, we tried to limit the amount of garbage allowed in wherever possible.

As for the garbage that did get in, we tried to cover every imaginable input and throw suitable exceptions if necessary. We consider most of the exceptions that we do throw important for the user of our API to know about, which is why we propagate most of them all the way to the `RESTHandler`, where we wrap them in an `ErrorResponse` to send to the user.

```
if (!projects.containsKey(projectID)) {
  projectServiceLogger.error("Operation failed: " + operationToLog);
  throw new IllegalArgumentException("Project ID " + projectID + " does not exist!");
}
```

Code Metrics

Facts

	Packages/Directories	Classes/Files	Total Lines	Source Code Lines	Comment Lines	Blank Lines
Java	17	92	8132	4377	2538	1217
vue	3	19	2003	1553	84	366
Sum	**20**	**111**	**10135**	**5930**	**2622**	**1583**

Used: “Statistics” - Plugin for IntelliJ

Bugs 1

Warning	**Nr of occurences**	**Package/Class/Method**
Unchecked cast	3	CanvasElementIterator, PersistenceTest, Serializer
Declaration access can be weaker	29	Package canvas, Package shape, Package facilitators
Declaration can have final modifier	10	Package canvas, Package main, Package project
Unused declaration	100	All packages
Probable bugs - Unused assignments	6	CanvasTest, FacilitatorTest, IteratorTest, PersistenceTest

Used: “Analyze” - Tool included in IntelliJ

Bugs 2

Bug Category	Bug	Nr of occurences	Occurence
Bad Practice	Method may fail to close stream	3	Package download
Internationalization	Consider using Locale parameterized version of invoked method (String.toUpperCase( ))	4	Package shape
Internationalization	Reliance on default encoding	3	Package download
Multithreaded correctness	Incorrect lazy initialization of static field	1	ProjectSerializer.getAndIncrementSeed()
Dodgy Code	Dead store to local variable	5	RESTHandler, Package download, ProjectSerializer
Dodgy Code	Redundant nullcheck of value known to be null	1	CanvasElementIterator
Dodgy Code	Exception is caught when Exception is not thrown	7	RESTHandler, ProjectSerializer
Dodgy Code	Test for floating point equality	2	CoordinateMath.skewX, CoordinateMath.skewY

Used: “SpotBugs” - Plugin for Maven

Discussion

After using analyzing tools such as the integrated IntelliJ tool or SpotBugs. Some of the Bugs which the IntelliJ tool showed us were already marked in the code as we programmed, but we did not recognize them or forgot to resolve them, while for others we simply do not have a good workaround since we deem them necessary for the server to work as intended.

SpotBugs gave us a more in-depth look at our project. It reminded us of the issues, which we had forgotten about and highlighted new problems, which were hidden from us up until that point. It also provided us with a certain peace of mind to see a definitive list of bugs and issues that were prevalent within our project, which we then would be able to work on and tick off one item after the other from the list.

Team Contribution

Daniel Sigmund

- Web client implementation
- Work on mostly the `shapes` and especially `shapes.transform` packages
- Jackson parsing annotations and parsing classes in `message` package
- Persistence classes & proxy pattern
- Composite & Iterator pattern (with Utz)
- Abstract factory & decorator pattern (with Alexander)
- API specification

Utz Nisslmüller

- Unit testing
- Maven & Spring integration
- Java foundation for server communication
- Composite & Iterator pattern (with Daniel)
- Observer pattern
- Design drafts (Composite, Iterator, Strategy, Factory patterns)

Alexander Ramharter

- Implementing design drafts
- Implementing download functionality with Batik
- Strategy pattern
- Abstract factory & decorator pattern (with Daniel)
- Client testing
- JavaDoc & UML diagrams

HowTo

Note: The API Specification can be found in `./implementation/apidoc/index.html`.
Note: The JavaDoc can be found in `./implementation/javadoc/index.html`.

Start the Server

1) From the root directory, navigate to `./implementation`.
2) Start the server with `java -jar se2-server-0304.jar`

Open the Web Client

1) From the root directory, navigate to `./implementation/web/dist`.
2) Open `index.html` in a browser.