1 Technical report: Packing program

This document is meant to specify what exactly happens in the current program. Here we can see what the opportunities, limitations, upsides and downsides are. Often will details be explained with either code blocks or class diagrams.

1.1 Minimal requirements

Every program has the minimum amount of features to function properly. Making a program decently requires a lot of time, extensive testing and a lot of communication with the client and other developers.

Therefore, I'll be honest and say that the program is, in my opinion, not yet completely finished and ready to be used in a commercial environment due to mostly lack of time and experience. Though it is ready to be used as a reference for proving theoretical problems, packing columns in boxes (with possible human intervention), etc...

Must-have features

- ✓Algorithmic approach: Pack columns in boxes.
- ✓Algorithmic approach: Calculate best combination
- ✓Compatibility: other front-ends
- ✓Basic visualization
- ✗ Extensive testing
- ✗ Error handling

Optional features

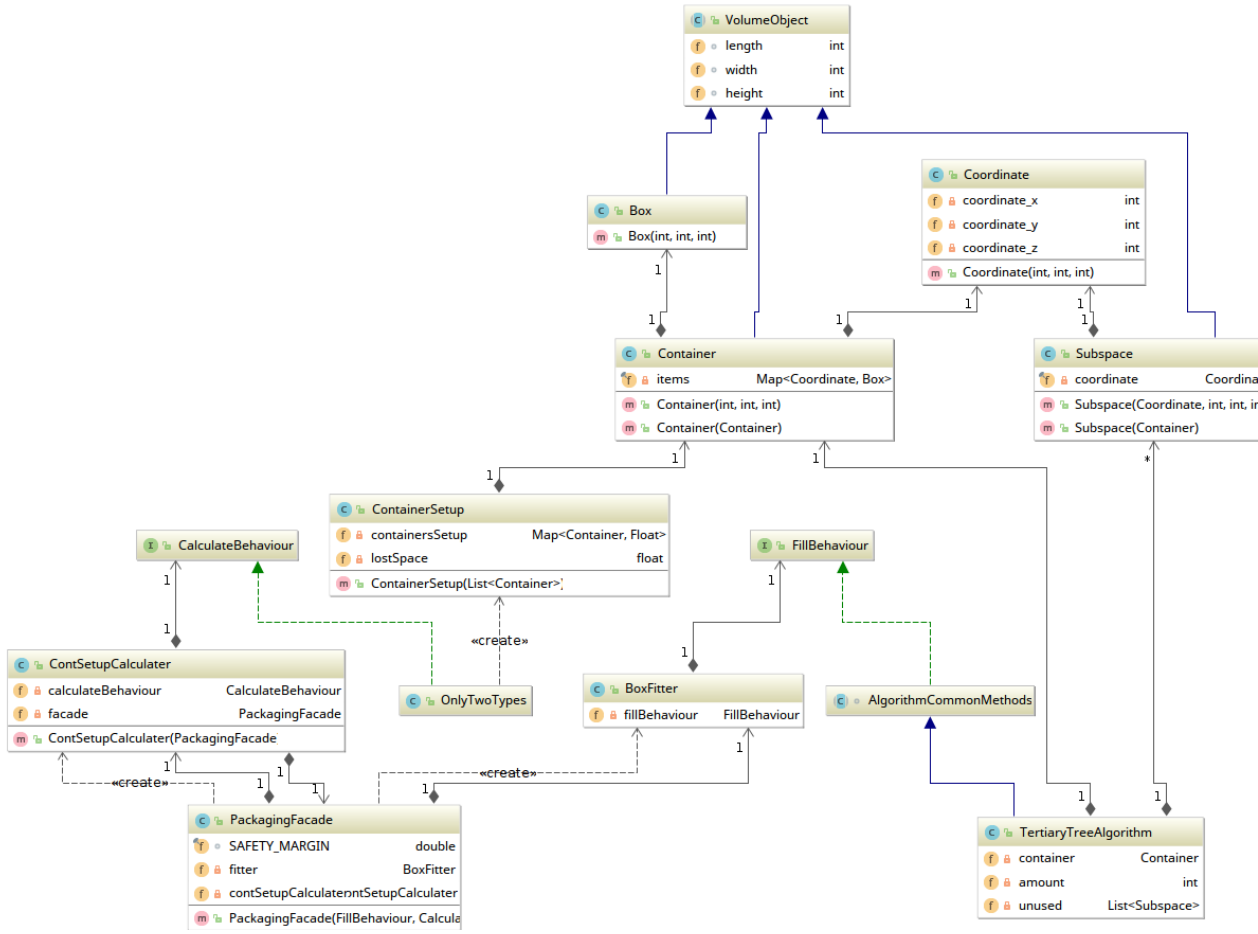
- Web based configuration: boxes management (update sizes, etc...)
- Integrate cost regression and link the projects
- visualization: short & detailed calculation (no layer schematics)
- dependency injection

Todo list: There are still some small changes to be made. The newest version is updated on our git repository*.
Please do keep in mind that different students worked at this project. Not every commit is as it should be, or is professionally written.

- ✓Update class names (Box = Column, Container = Box)
- ✓Review constructors usage
- ✓No calculations in Facade
- ✓Correctly implement Strategy pattern
- ✗ Copy constructor sort keys based on coordinates

1.2 Class diagram: General overview

Here is a short overview. This diagram is generated through IntelliJ IDEA.



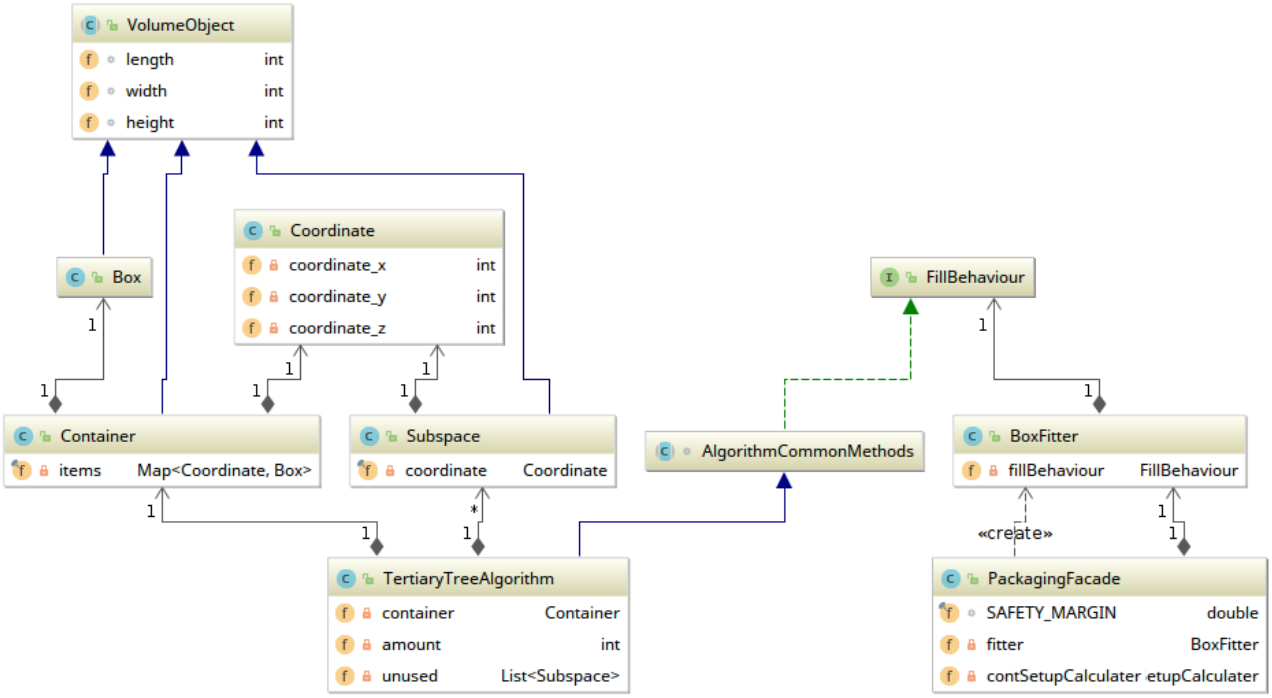
There are 3 parts in this program with their respective classes. There are more classes, but those are referenced in either a general context (Factory classes) or are entities (Column, Coordinate, Subspace, etc...).

1. The Facade: which is responsible for all communication with other programs. The rest service will communicate entirely through this class, but the class is not allowed to have any logic behind it.

2. The packing algorithm classes: Boxfitter, FillBehaviour, TertiaryTreeAlgorithm.

3. The optimal setup algorithm classes: ContSetupCalculator, OnlyTwoTypes, ContainerSetup.

1.3 Class diagram: Packing algorithm



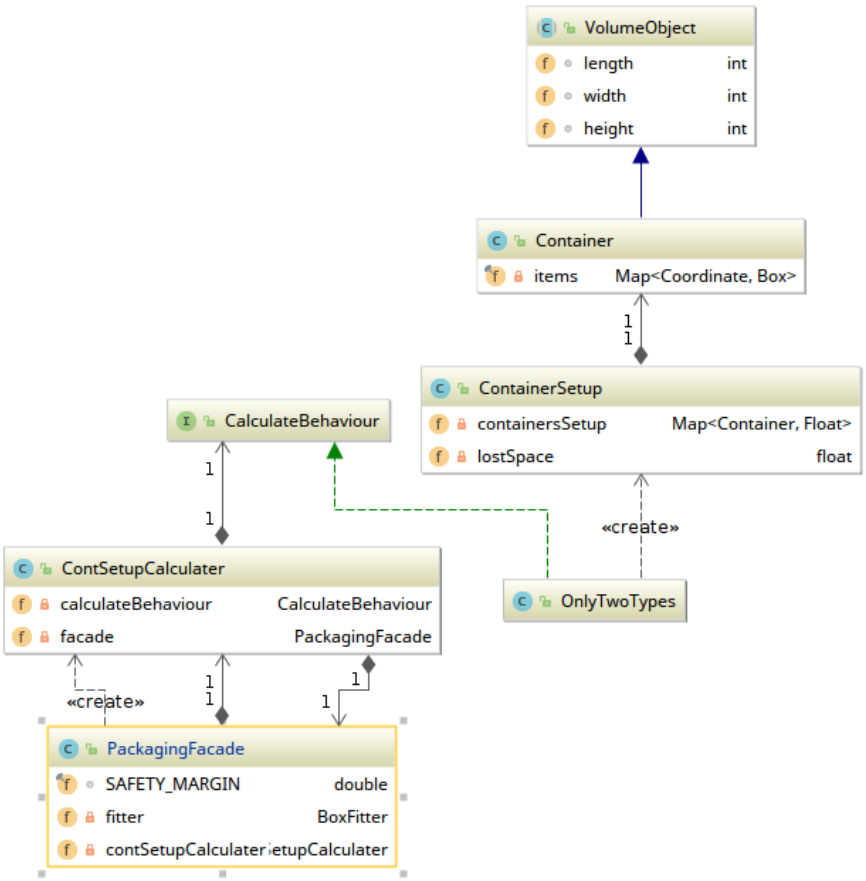
In the packing algorithm, this is the first step that happens. The data entered in the front-end is passed through the rest-service, then the Facade, and finally arrives here.

A Strategy pattern is implemented for future algorithms. Right now, the Tertiary Tree Algorithm works only with 1 homogeneous object. In case you'd want to try a different algorithm or approach, you can just make a new class, implement FillBehaviour (or extend AlgorithmCommonMethods) and start coding.

The TertairyTreeAlgorithm will place Columns in subspaces, then divide that subspace in 3 other subspaces. This will keep going until the column no longer fits in the subspace. Whenever a column is placed, it will be mapped with a coordinate.

Keep in mind that the creation of objects is entirely done through factories.

1.4 Class diagram: Setup algorithm



These classes will determine what the best setup of boxes is. The packing algorithm will determine how much space is lost and how many columns can fit into a box. Afterwards, "ContSetupCalculator" will determine (according to the previous data) how many boxes will be necessary.

This algorithm is not the most performant, as it brute-forces all possibilities and then chooses the one setup with the least volume loss.

There are 2 major drawbacks:

1. It is a brute force algorithm.
2. Only 2 containers are supported in the "OnlyTwoTypes" class. A better algorithm can easily be implemented by making a new class, and implementing "CalculateBehaviour".

After choosing the best "ContainerSetup", you'll return this to the Facade. Before returning it to the rest-service, it will be converted in another format. "TranslatorContainersetup" does this for you. After translating, it is sent to the rest-service. There it will be converted into JSON format.

2 Maven configurations

2.1 back-end

Please do remember that whenever you build this for the very first time, you must first "build & install" it with maven so it is in your local repository. This is easily done with an IDE that supports maven Projects.

I'd like to focus on 2 parts of the "pom.xml" of the back-end project:

```
<groupId>com.bin</groupId>
<artifactId>3dbinpackaging</artifactId>
<version>1.4</version>

<dependencies>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
        </dependency>
</dependencies>
```

The first part are the identification details about our back-end project. We'll need these later on when we import it in our rest-service. The second part is simply a JUnit dependency that you import.

2.2 rest-service

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.2.RELEASE</version>
</parent>

<dependencies>
        <dependency>
                <groupId>com.bin</groupId>
                <artifactId>3dbinpackaging</artifactId>
                <version>1.4</version>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.google.code.gso
        <dependency>
                <groupId>com.google.code.gson</groupId>
                <artifactId>gson</artifactId>
                <version>2.8.0</version>
        </dependency>
</dependencies>
```

Most of this is from the official spring rest service website. This is to use Spring annotations, etc...

The other parts that are important, is first of all the dependency where you import the previously build project (the back-end). Make sure the versions match.

There is also the "gson" dependency. That's because we are mapping complex objects, and gson is (in my experience) easier to use.

3 CORS and GSON complex serialization

The default setting refuses JSON request that's not from the same origin. To disable this for specific routes, we have the following configuration in our "Application.java" file in the rest-service.

```
@SpringBootApplication
public class Application {
        public static void main(String[] args) {SpringApplication.run(Appli

                @Bean
                public WebMvcConfigurer corsConfigurer() {
                        return new WebMvcConfigurerAdapter() {
                                @Override
                                public void addCorsMappings(CorsRegistry re
                                        registry.addMapping("/api/**");
                                }
                };
        }
}
```

Now all sorts of front-ends can send JSON requests to our application.

Because we work with complex objects (Maps in Maps, objects in objects), we'll have to configure GSON. This is done in your Controller class in the rest-service.

```
@RestController
public class ContainerSetupController {
        GsonBuilder builder = new GsonBuilder().enableComplexMapKeySerializ
        Gson gson = builder.create();
```

4 Nice implementations & extensions

This project is just meant as a stepping stone for a more automated and easier approach to calculate data. Of course, it does not stop there. With the second part, cost regression, programmed and implemented, a lot of steps can be automated.

Several ideas:

- Implement cost regression in the rest service, and adjust the front-end.
  this will encourage a complete, smooth, web-based approach. There are 3 advantages over Excel:
  1. Right now (if I'm not mistaken) the Cost Regression that's made in Excel, uses Windows-only code and makes it OS dependent.
  2. Time is lost when copy-pasting data from the web application into the excel
  3. More futuristic opportunities, such as linking calculations to a database. Either to keep a history of orders, or for statistical analysis.
- Perhaps the generation of invoices?
- 3D visualization on the website / front-end
- Current boxes (dimensions) are an "Enum" file. This can either be kept in a configuration file, or database.

Enum file as a reference:

```
public enum ContainerType {
        SMALLEST(120,80,100),
        BIGGEST(150,100,120);

        private final int length;
        private final int width;
        private final int height;

        ContainerType(int length, int width, int height) {
                this.length = length;
                this.width = width;
                this.height = height;
        }
```