Contents

User requirements

People, saved in the system, have a name, family name, name of his/her father, a number representing wealth and sex (male, female). Each person can be a Human or Elf and at the same time either a ruler of some kingdom, a governor of some region or landlord, whose family owns a construction, and it is passed by generations. For humans, we want to store a list of diseases that he/she had experienced, whereas for elves the type of magic is stored (healing, offensive, defensive or scholar). For rulers, we need to know the date of the beginning of his reign and a ruling time that gets calculated from the aforementioned start ruling date. As the governor needs to have some managerial education, we need to store his/her skills. The landlord needs to remember the number of generations that his family owns a specific place.

People can join fractions against the ruler. For each fraction, we want to store reasons that should be at least one, and the support of this fraction by the populations or participants that should be less than 100. When this number reaches 100, a rebellion starts and a ruler is either changed or a fraction is suppressed and it disappears. A ruler cannot join a fraction against himself. In each fraction, there should be at least one person and a leader that is one of the rebels.

A ruler employs a governor to govern a particular region or to be a councillor in the ruler's council. For the employment, we need to store the date of signing the contract, and a tax value that the government pays the ruler from the region he governs. That value cannot be increased by more than 10% at once but can be decreased by any amount. If the contract is temporary, then the termination date should be stored. Moreover, a ruler rules the kingdom, that must always have a ruler and he/she can be a regent in some kingdom, but only if a ruler is underage. A governor, as mentioned before, either is a councillor in a kingdom or governs a region. A region must have a governor though and a kingdom can have up to 7 councillors. A landlord can manage many constructions and each construction must be managed by some landlord.

All the kingdoms, regions, and constructions must have a possibility to be displayed on the map, thus they need a name, a colour that would represent these places on the map and approximate coordinates that would point somewhere to the centre of the place (i.e. 46.65N; 74.25W). For each kingdom, we need to be able to get the power that is several regions in the kingdom times the base power per region that is 50. In each region, we store a climate, and there is at least one construction for which we store a construction date, optionally destruction date, a type of construction that can be a City, Castle or a Village and age, based on the date of construction.

Each construction has a status that is either (from best to worst) prosperous, inspired, standard, raided, starving and destructed. Every 3 months a status worsens from prosperous to standard and from raided to destroyed. To transfer a status to raided the construction must be raided. After a year of being destroyed a construction is removed from the system. The status can get better by helping locals in construction.

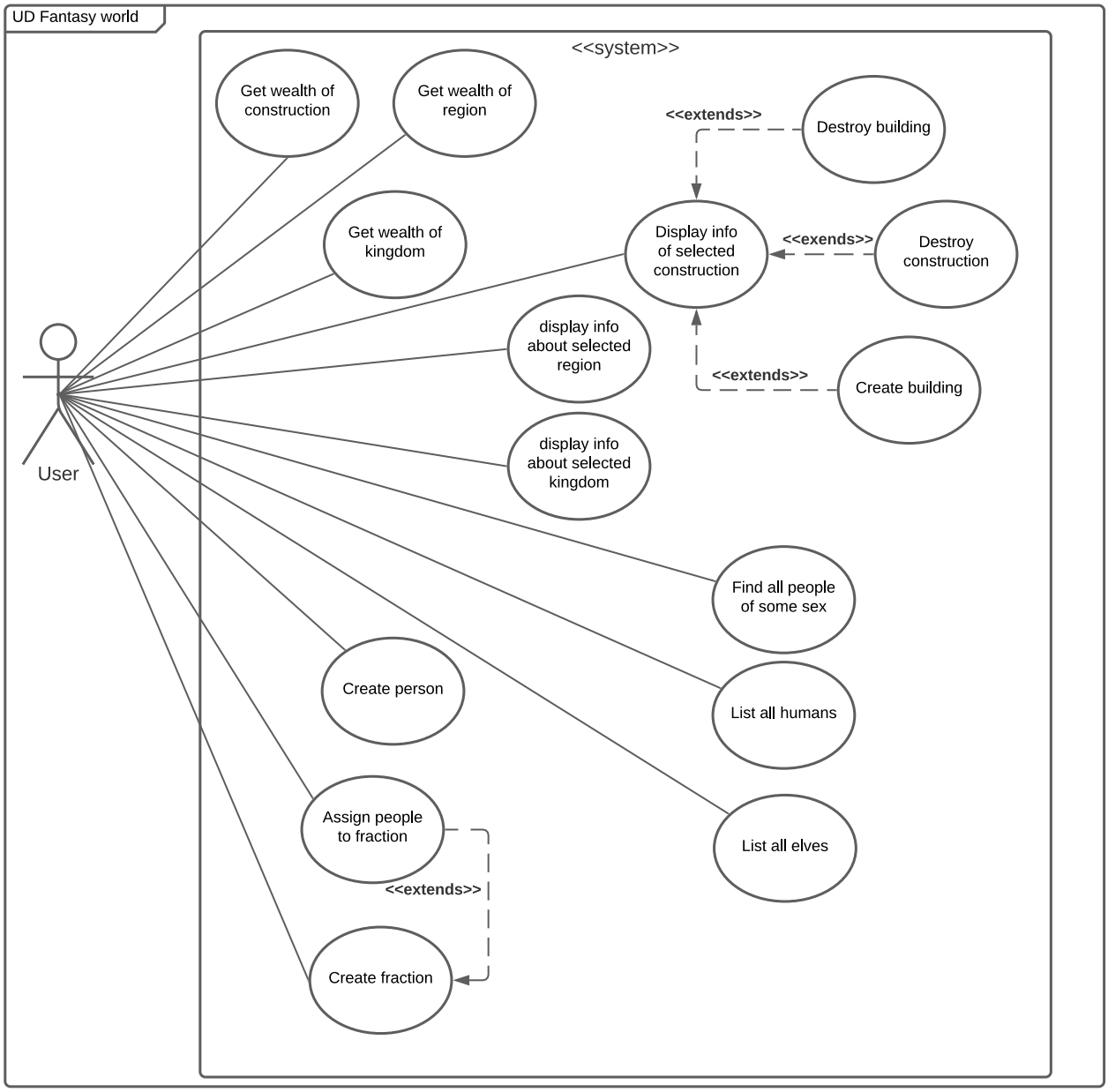
There is at least one building in each construction that cannot exist without it. For each building, we store the type(forge, farm, house, storage, inn, lumbermill) and optionally a name.

A king lives in a capital that can be a centre of several religions. A Capital is represented on a map as a region and has all properties of the construction.

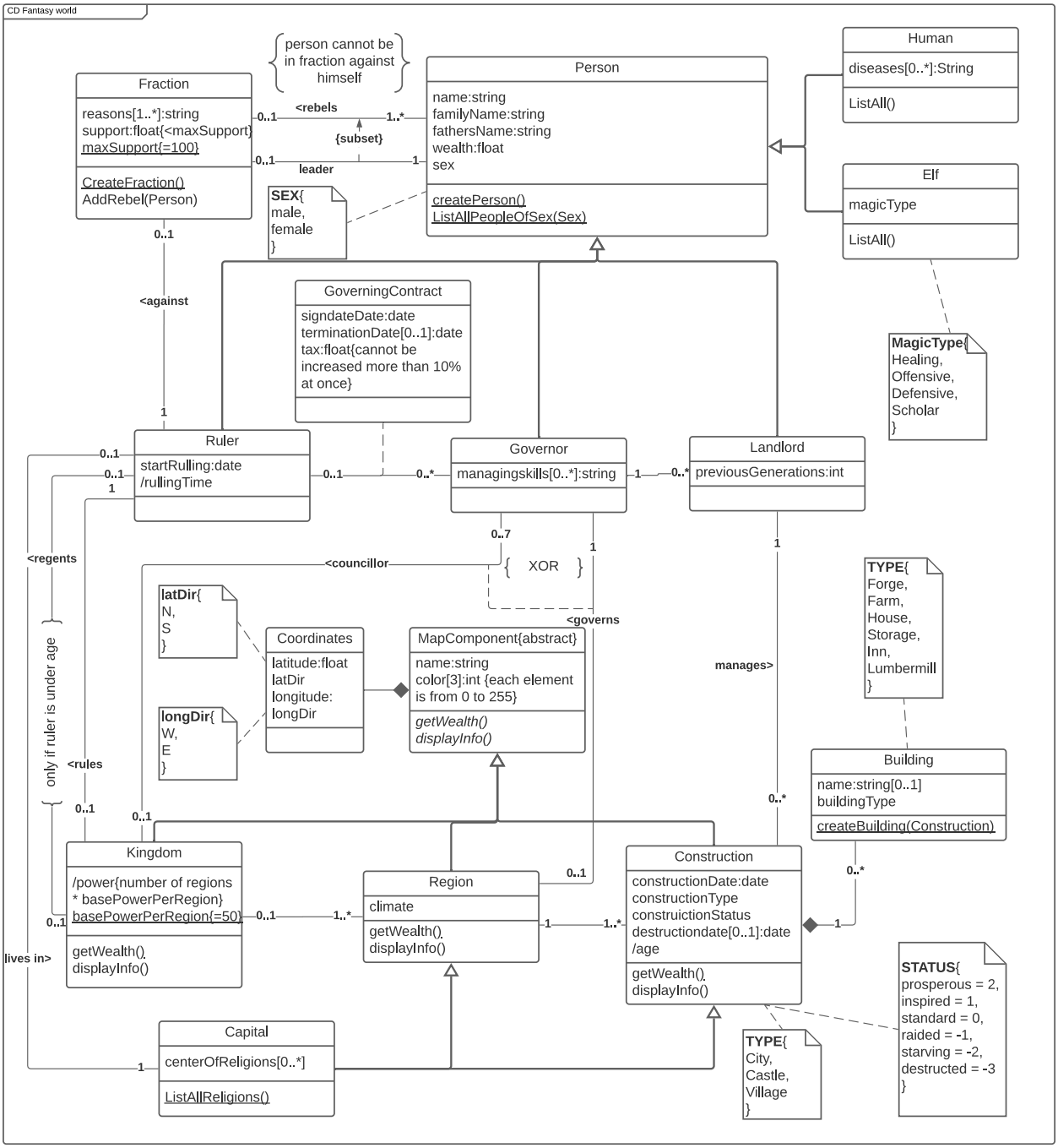
For the system described above a user should be able to:

- Get the wealth of each construction that should be 50 if prosperous, 0 if raided and –50 if destroyed.
- Get the wealth of each region that is a total wealth of constructions belonging to it.
- Get the wealth of each kingdom that is a total wealth of regions in the kingdom.
- Create a new person.
- **Display info of selected construction**
- Destroy construction
- Create buildings in constructions.
- Destroy buildings in constructions
- Display info about created region
- Display info about kingdom
- Find all people of some sex.
- Find all elves.
- Find all humans.
- Create a fraction.
- Assign people to a fraction.

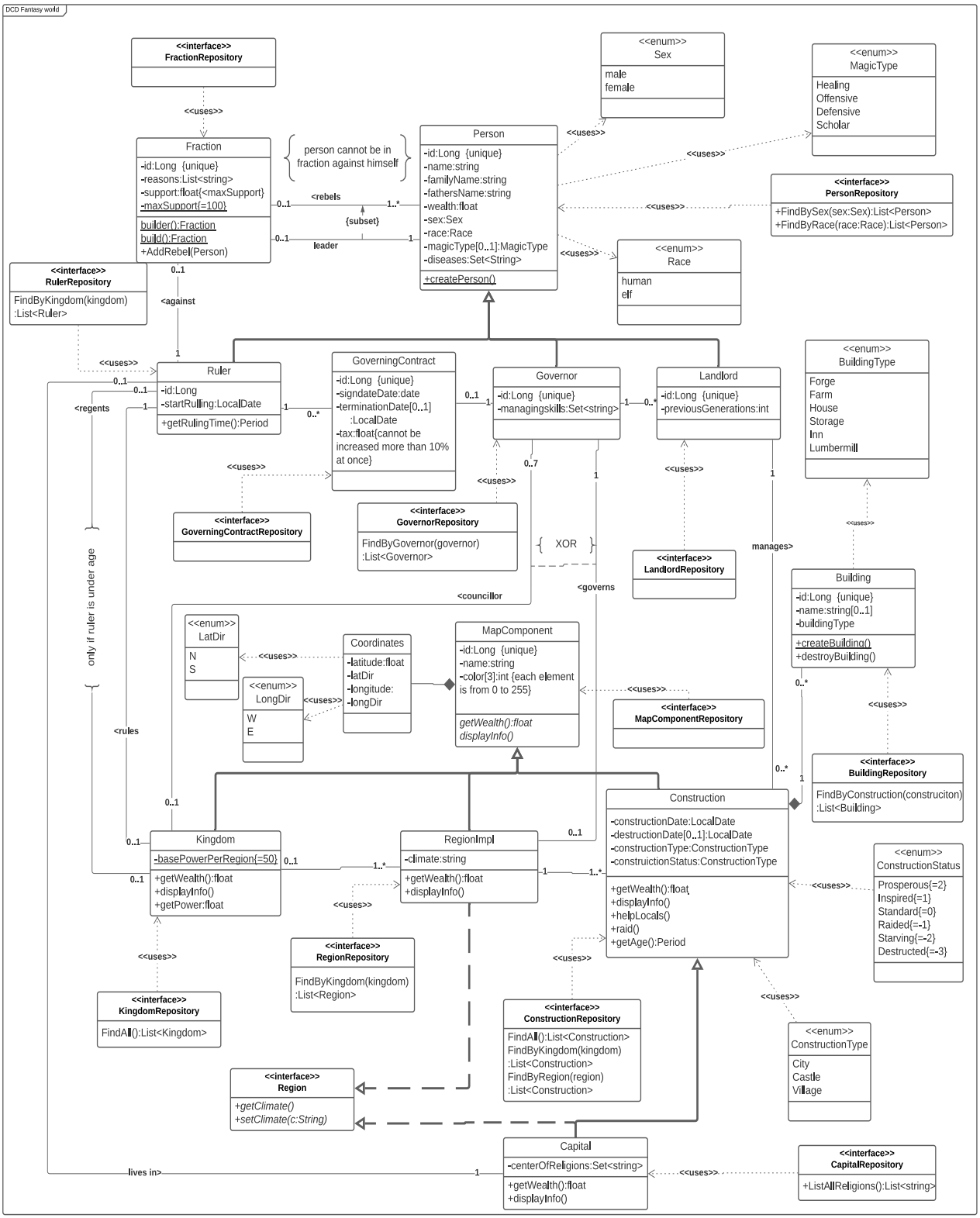
The use case diagram



The class diagram - analytical



The class diagram – design

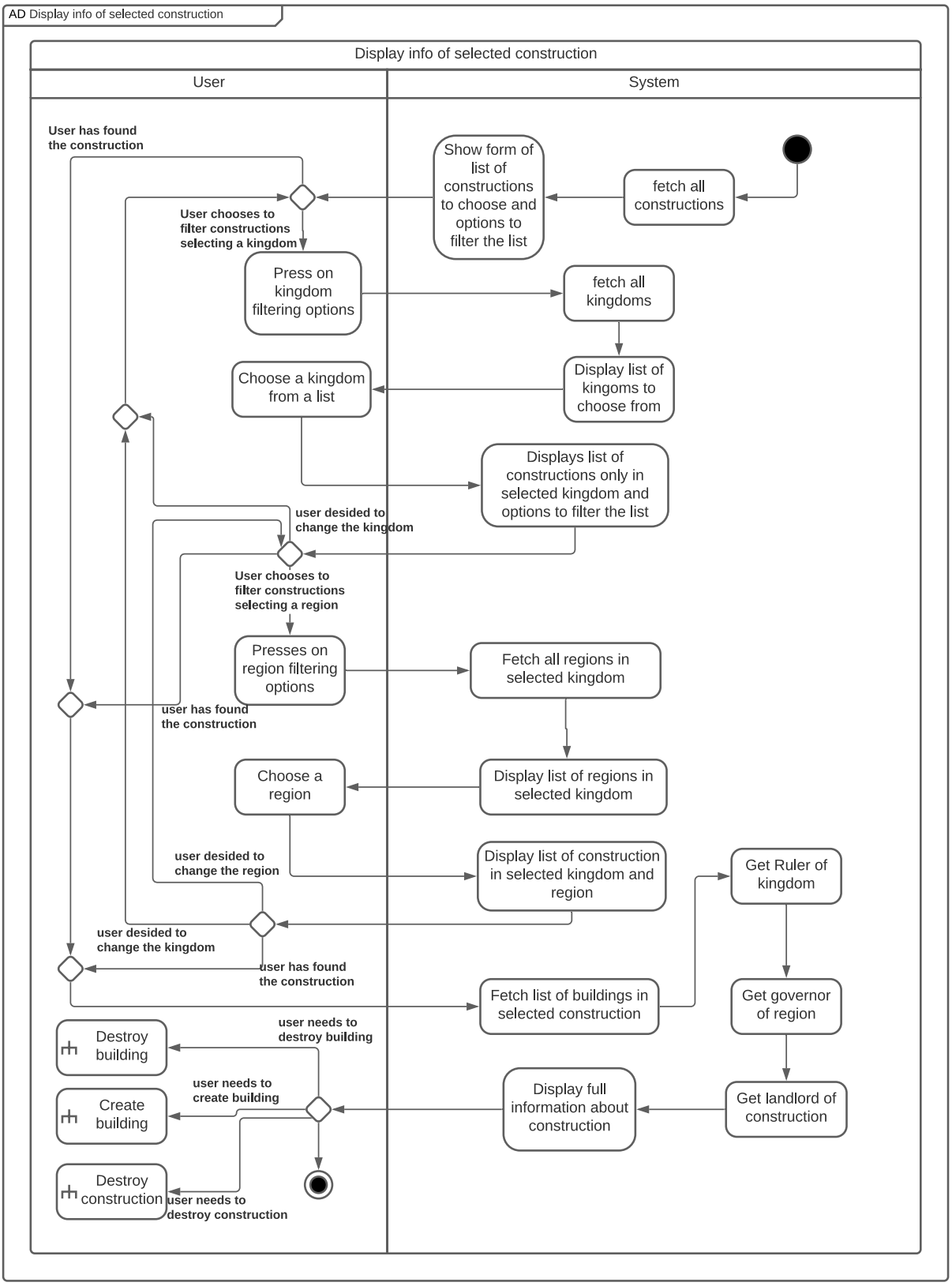


The scenario of the selected use case (as text)

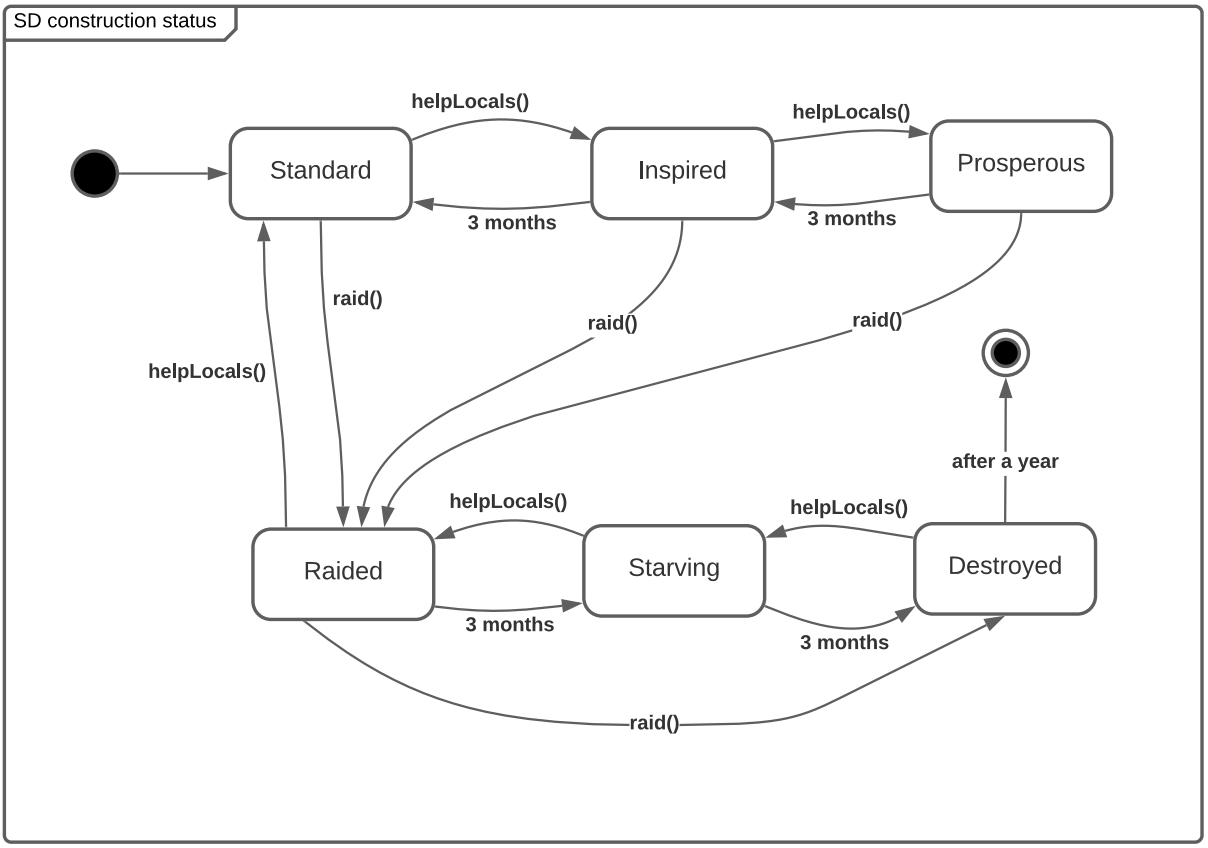
Display info of selected construction

The use case starts with a user having a need to look up all the information concerning the specific construction or making some operations on its objects. The system should show the user the list of all constructions shown in a system, sorted in alphabetical order. In order to make the search easier, the system should provide the user with the possibility to choose a kingdom, which will display a list of constructions that are found only in the selected kingdom, and a region that will narrow the list even more. After the user found the construction that he needs and clicking on it, full information about the construction is shown, along with the kingdom it belongs to, a ruler of the kingdom, a region it is in, governor of the region, the landlord of this construction and the list of buildings that are in selected construction. It should provide possibility for transition to the following use cases: destroy building, destroy construction, and create building.

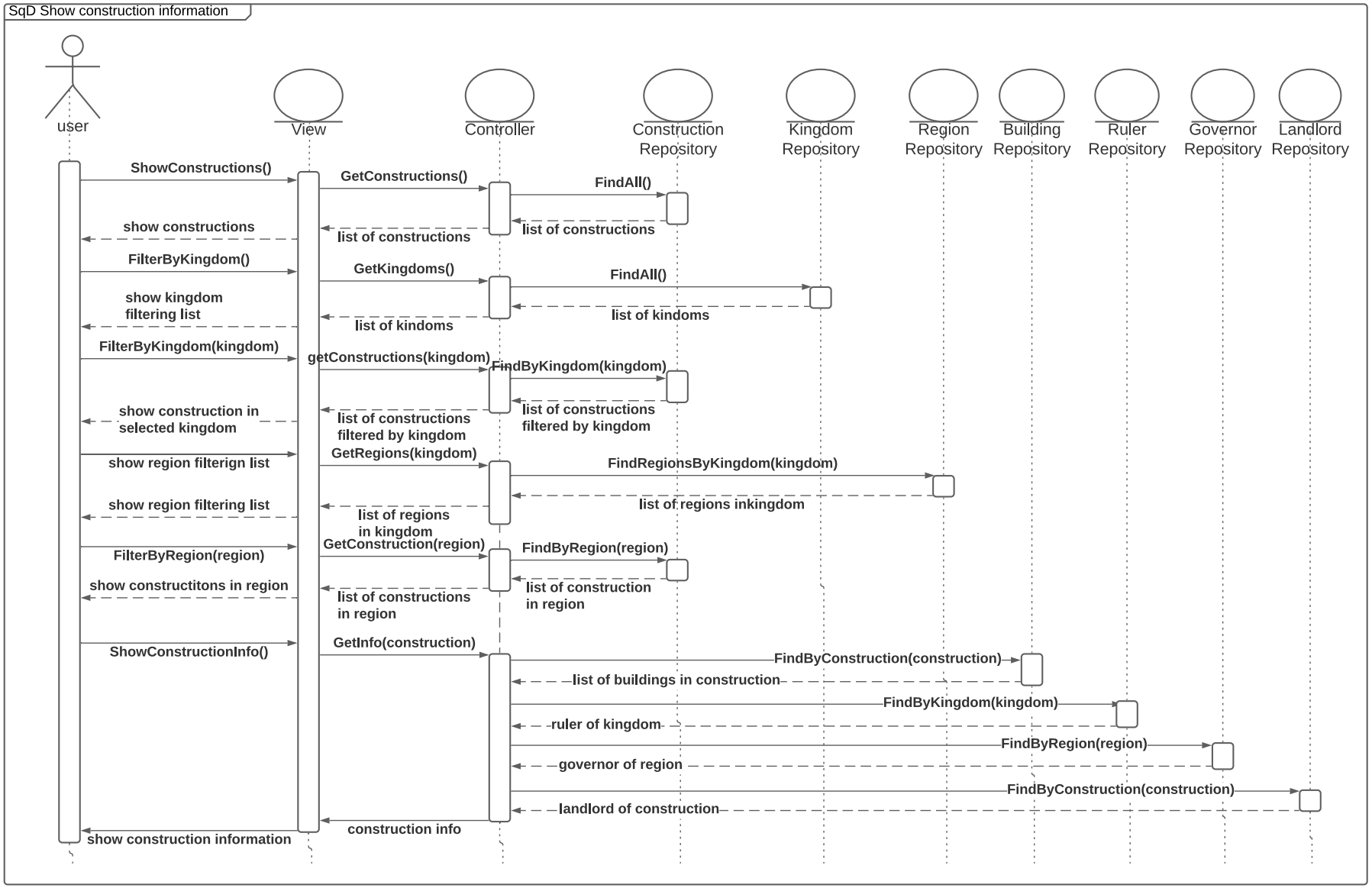
The activity diagram for the picked use case



The state diagram for the selected class

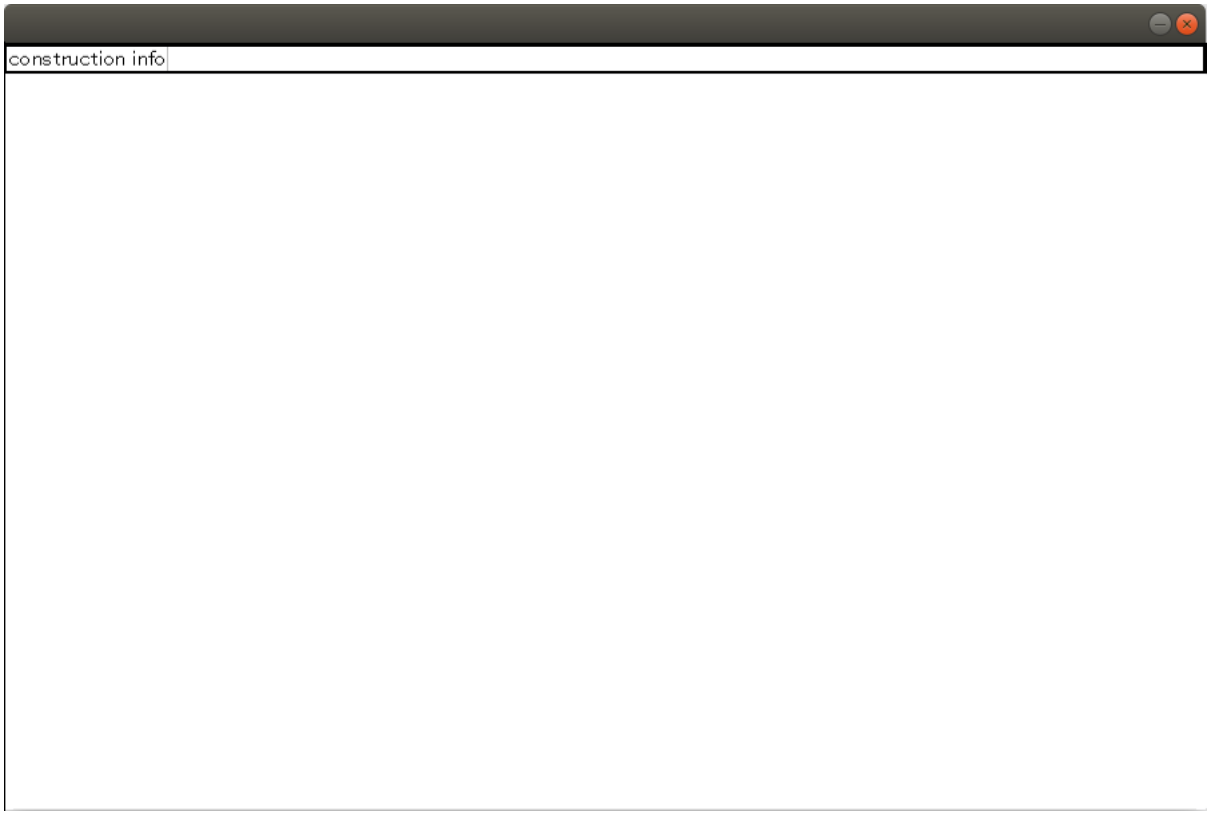


The interaction (sequence) diagram for the selected use case

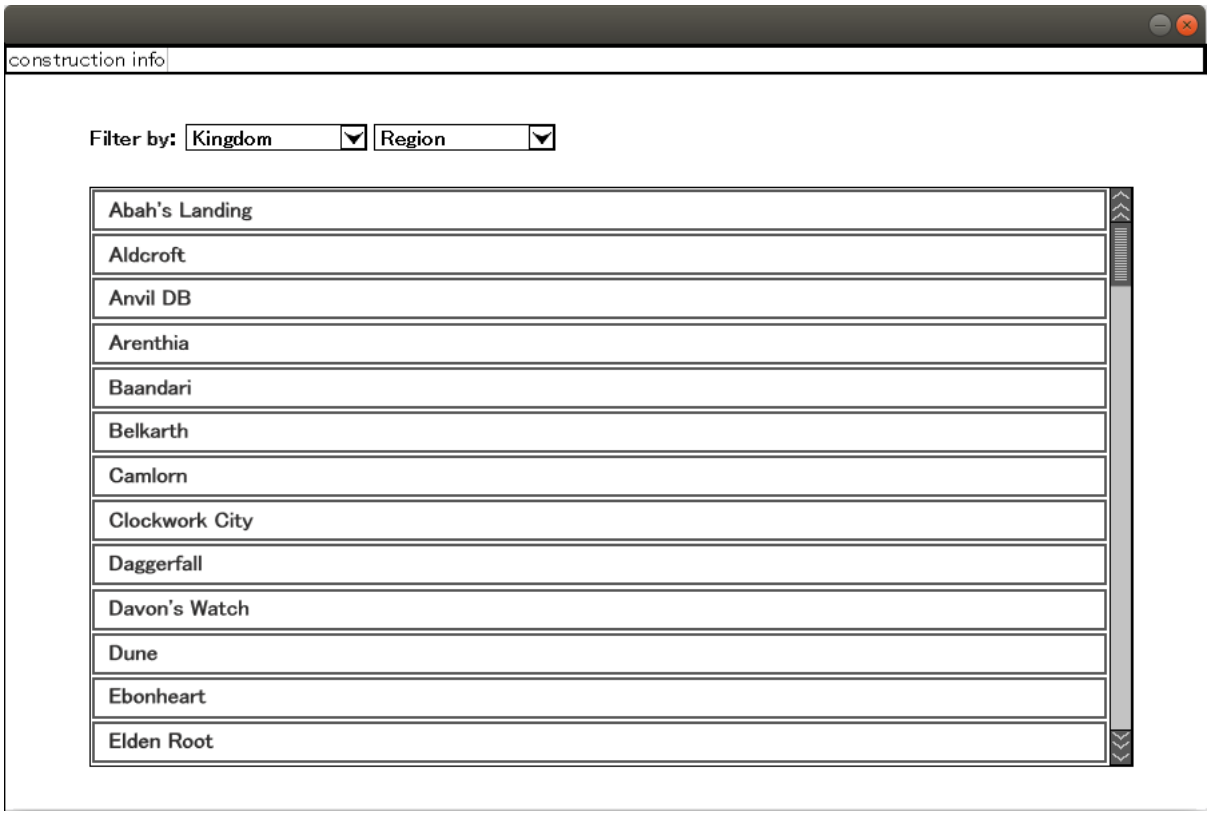


The GUI design

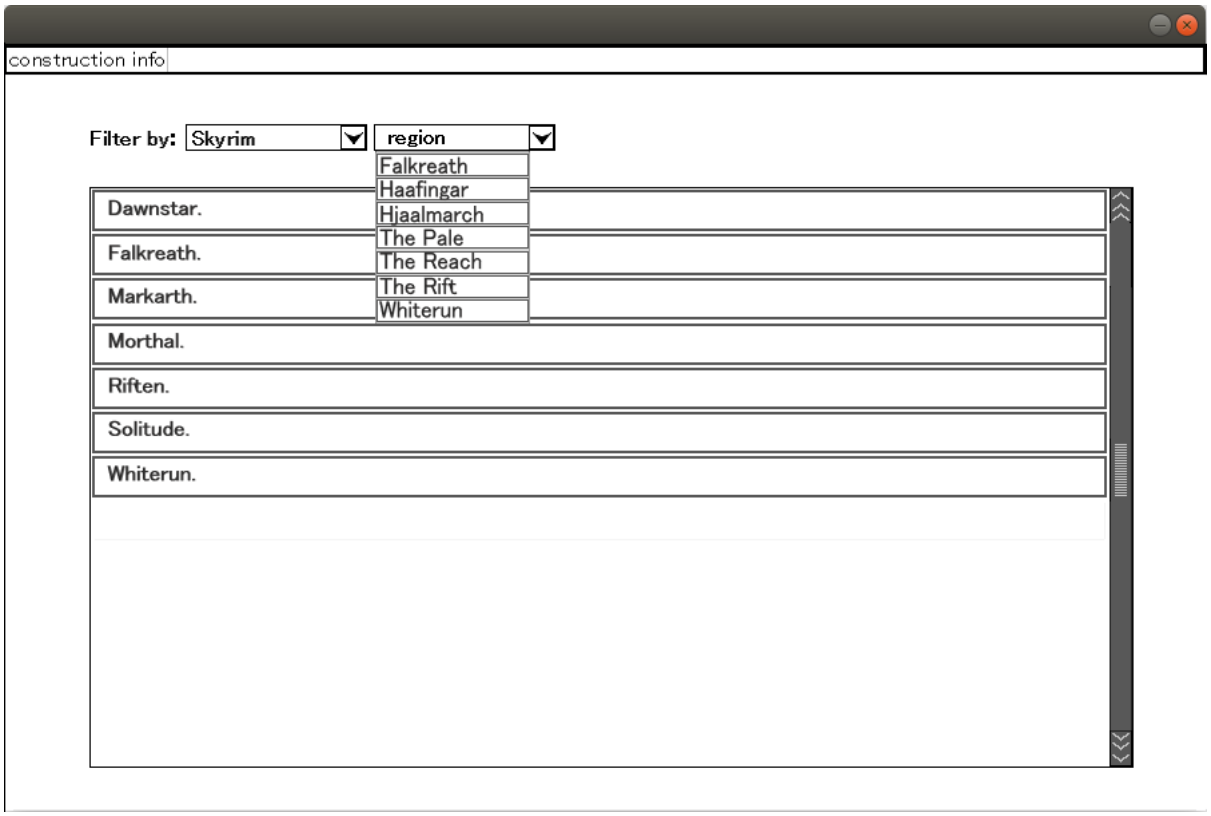
1. Main menu of the Application. Use case starts with user clicking on the top left button to display construction info.



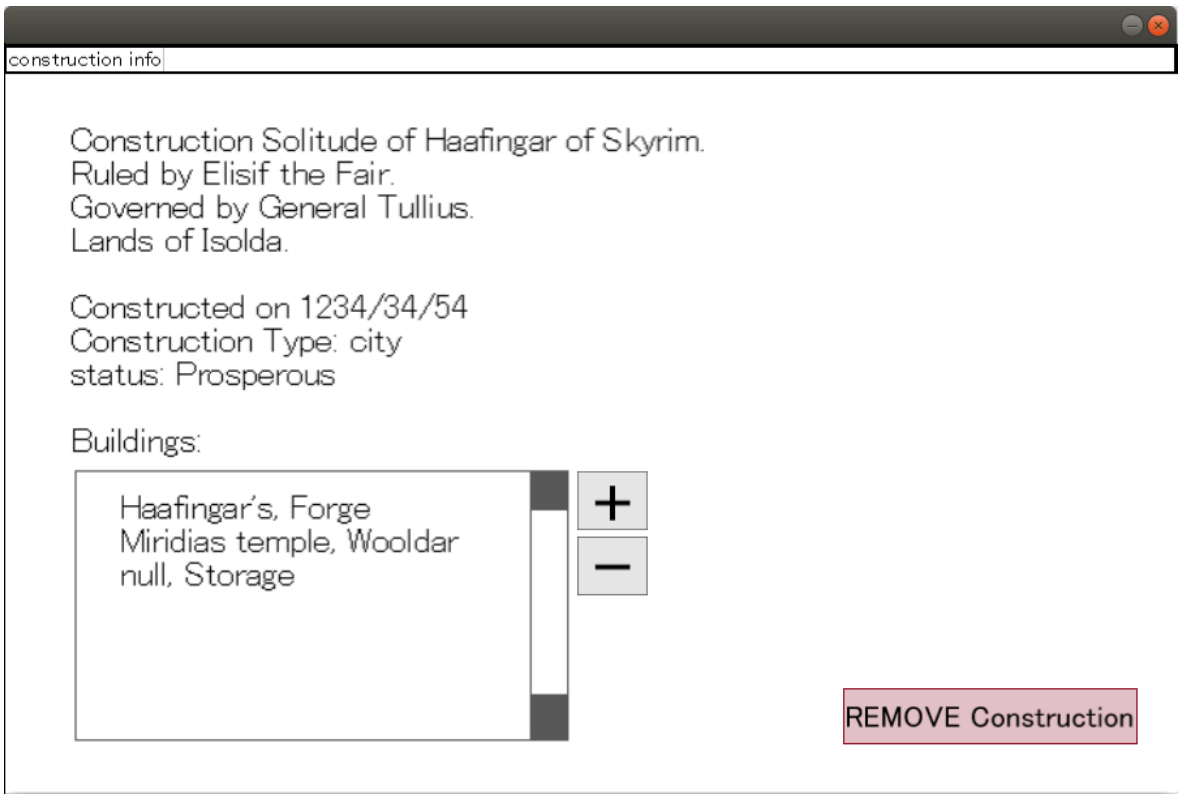
2. Construction selection menu without filtering applied.



3. Construction selection menu with chosen kingdom and example of choosing region for filtering.



4. Information about construction displayed



The discussion of design decisions and the effect of dynamic analysis

The projects implementation will be based on Java, using Spring, Hibernate and Lombok.

Firstly the analytical class diagram experienced some changes due to Java not being able to directly implement some UML constructs that were present on the diagram. Constructs than are transferred from to the design class diagram and changes made to them are described below.

Inheritances

- Multi-aspect inheritance.
  Due to the fact that the aspect information differed one from another in a few attributes, there was no need to store them as different classes, they were merged into the superclass. The classes that represented additional aspect (Elf and Human classes) are represented as an attribute race of Enum type of those 2 values. In addition, the Person class has attributes needed for both variations of aspect which are now not mandatory, as they are needed only if the race attribute equals to a specific value.
- Multi inheritance
  The class with the least number of attributes and functionalities was chosen, in this case it was a Region class, that became an interface, having abstract methods needed to operate on attributes that a Region required. The Region class became an RegionImp class that can be instantiated. The Capital class extends Construction class and implements Region interface, hence it can be treated as an object of types Region, Construction and of MapComponent separately.

Associations:

- Association for an attribute.
  This association connecting Ruler and Governor classes was replaced by a simple class in between the two aforementioned ones and one to many associations between each pair. (Ruler -> GoverningContract -> Governor)

Attributes

- Class extent and persistence
  Class extent and persistence is handled by the database, because, as mentioned before, the project solution is based on hibernate.
- Optional attributes
  The types of attributes is changed from primitive to Object in order to be able to store null.
- Multi-value attributes
  Multi-value attributes are now represented as Sets.
- Derived attributes
  Are now represented as getter methods with the corresponding name, that will calculate and return the required value based on the business logic.
- Enums
  Enums are now represented as entities of enumerated values.

In addition:

- Every class in a diagram obtained a Long value, named id and marked as unique in order to behave as a primary key in the database.

- Each class obtained a repository interface where all the search functionality, requiring the extent, is used. So all the methods from classes that performed the search functionality are transferred to the repository interface. Moving class methods to repositories made them object methods because firstly they are outside classes they are operating on, secondly they belong to repository that is always a concrete object.
- Methods with Create functionality were replaced by class methods builder() and build() because of using Lombok functionality (Builder annotation) it would handle the creation of an object.

Dynamic analysis effects

State diagram introduced several methods that would change the Construction status. They are: helpLocals() and raid().

Activity and Sequence diagram represented how should a use case of Showing the Construction information. Those diagrams required several methods for retrieving data from repositories in order to make it possible to display the required information. The repository for construction obtained functionality to: fetch all constructions, fetch constructions by kingdom, fetch constructions by region. Kingdom repository obtained method to find all kingdoms. Region repository obtained method to find all regions by kingdom. Finally, ruler, governor and landlord repositories obtained functionality to find them by kingdom, region and construction respectively.