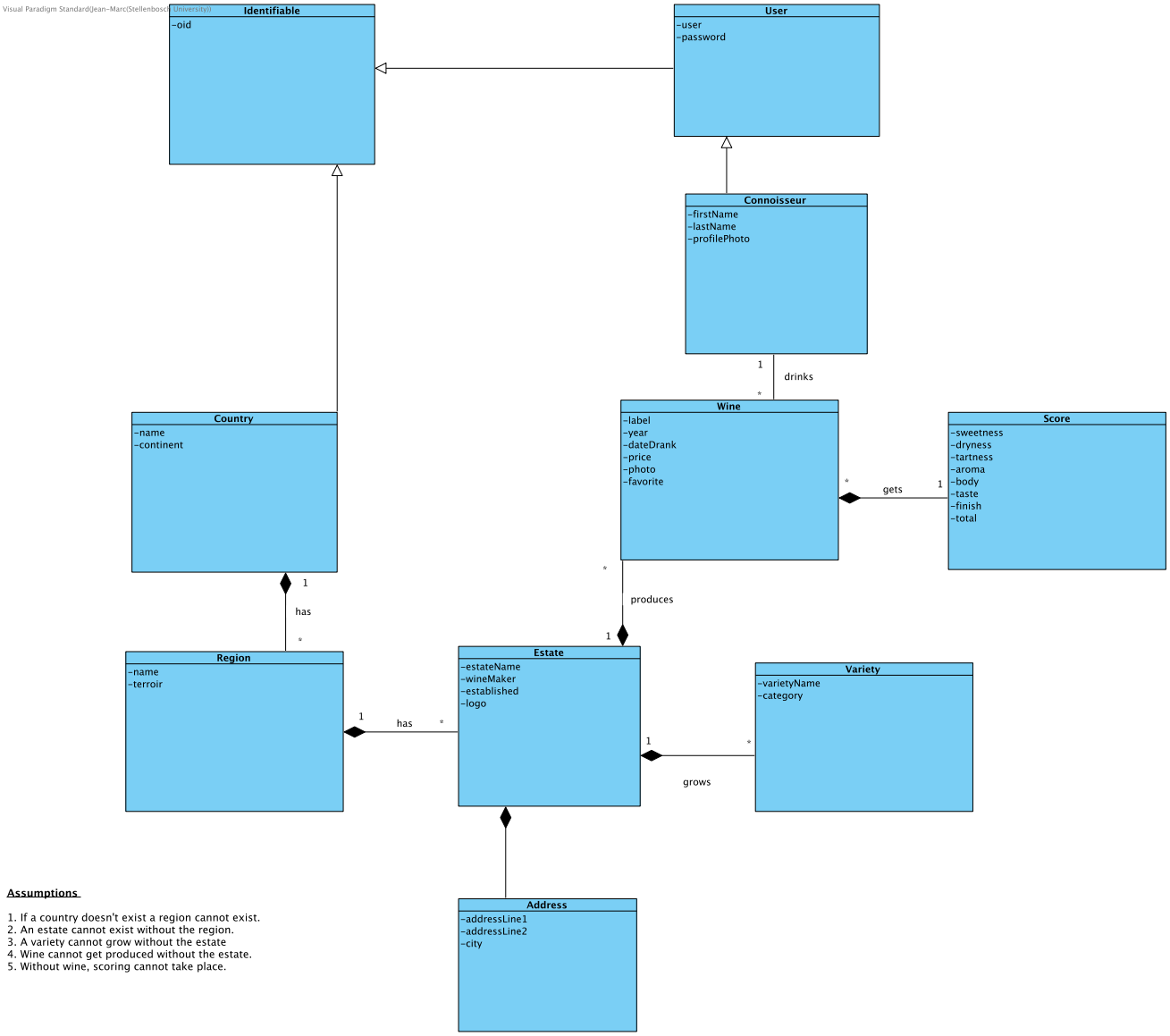
High-level Overview

For the connoisseur on the path to becoming a sommelier, one must taste and recall a vast amount of different wines ranging from numerous countries worldwide. Different regions within countries will have specific characteristics that influence their terroir and the cultivars that are grown. Technical tastings require scoring the wine against specific criteria and making notes. This system aims to help manage and store all the various wines tasted and drank by a connoisseur, allowing for ease of access to information linked to scores of different wines and their origins and estate information. This is valuable information to track for a wine enthusiast and one that would pursuit the title of a master sommelier and allow for tracking and comparing with other connoisseurs.

The user will start by inputting their details on the connoisseur page, this page will also allow them to link any wines they have drank to their name. The user will be able to input information about the wine and link it to its origins in terms of the estate it's from and region. The user will be able to populate the system with the different cultivars grown in terms of variety and link it to the estates that grow them. The user will then be able to score the wine and link it to the wine they have scored and link it to their name. The user will also be able to input the different wine regions and input notes with regards to the terroir of the region, linking the region the various varieties grown. The user will be able to input estate information and link it to the country and region they are within. The user will be able to input all the countries producing wine and link them to their various regions. The user will be able to input their connoisseur details and link

1. Class Diagram

1.1. Wine Connoisseur



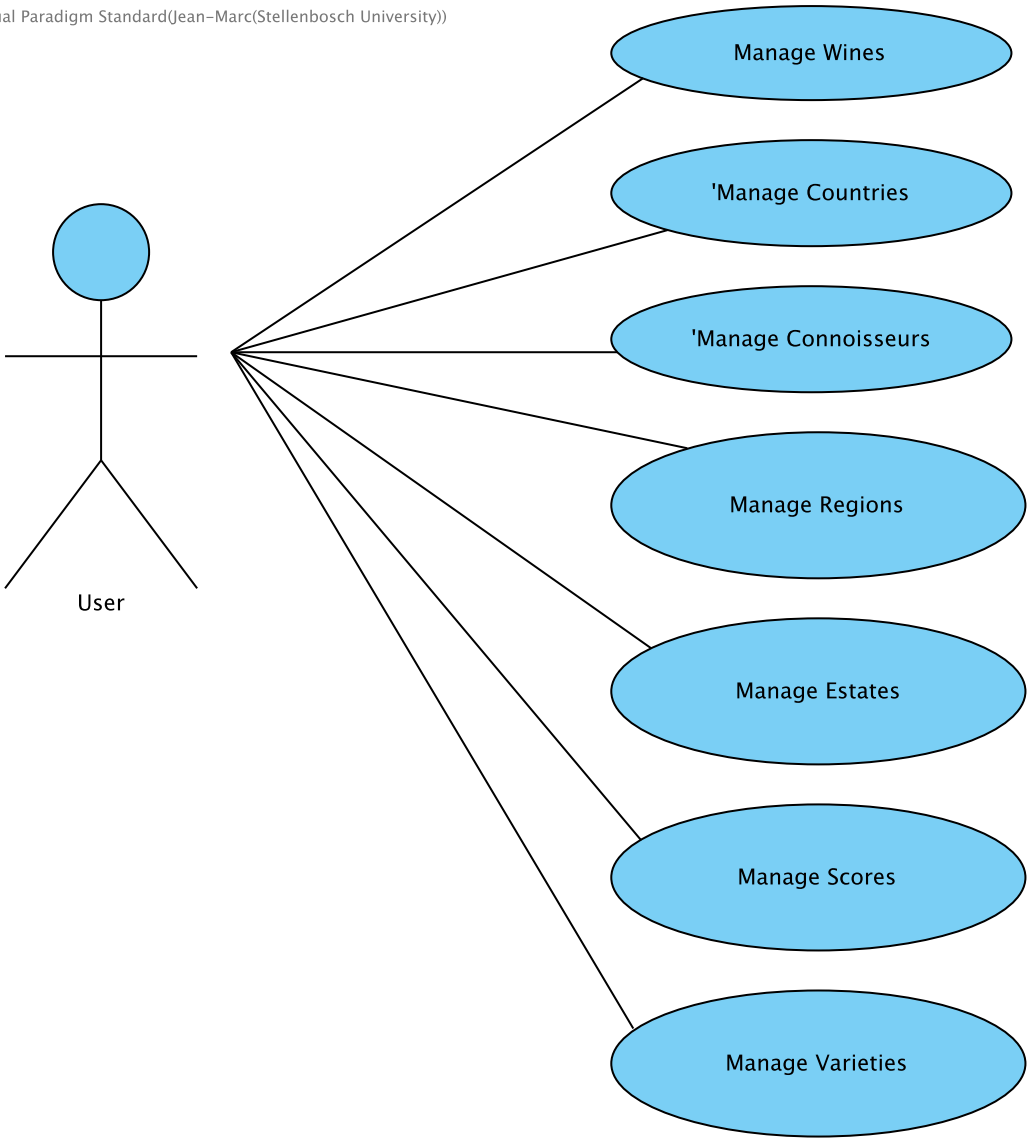
Assumptions

1. If a country doesn't exist a region cannot exist.
2. An estate cannot exist without the region.
3. A variety cannot grow without the estate
4. Wine cannot get produced without the estate.
5. Without wine, scoring cannot take place.

2. System Use Case

2.1. System Use Case

Visual Paradigm Standard(Jean-Marc(Stellenbosch University))



Code explanation

1. **Address.java**

The first two lines indicate the package being worked in and any imports needed for the class. This class is annotated with the @Embeddable so that if the entity Estate is removed so too will the address. The description properties for this class is addressLine1, addressLine2 and city and are all string data types. Getters and setters are generated for the description properties.

2. **Connoisseur.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Connoisseur Details" and "Wine Finder" and given the description properties that would be in their respective sections in the view. Next line is started by the annotation @Entity which marks that the connoisseur class is an entity. The entity is also an extension of the identifiable entity, which makes it inherit its properties. The description properties for this class is firstName, lastName; of which both are string types and are indicated by the @Column annotation as well as a @Required annotation which makes the property a requirement to be filled in. Next is followed by a @Stereotype annotation which indicates a special use of a type and for this class the type is a photo for the connoisseurs profile photo to be added. The relationship between connoisseur and wine is indicated with the @ManyToOne annotation for their relationship followed by the table in which the relationship is between (private Wine wine). Getters and setters are generated for the description properties, stereotype and ManyToOne relationship.

3. **Country.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Country Details" and "Region" and given the description properties that would be in their respective sections in the view. Next line is started by the annotation @Entity which marks that the country class is an entity. The entity is also an extension of the identifiable entity, which makes it inherit its properties. @Column indicates the description properties, the first one being a string called name and the next description property is annotated with @Enumerated with in parenthesis the data type of the enumeration, the next line is "private Continent content" references where to look for the data, "public enum Continent{….}" indicates the output for the enumeration. @Enumerated enables for the drop-down menu functions for continents. The first relationship for country is brought by the @ManyToOne which shows a many Regions can be in one country, this is followed by the annotation @DescriptionsList which instructs openxava to visualise the references as descriptions list. Getters and setters are generated for the description properties, stereotype, enumeration and ManyToOne relationship.

4. **Estate.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Estate information" and "Country finder" and given the description properties that would be in their respective sections in the view. The entity is also an extension of the identifiable entity. Next line is started by the annotation @Entity which marks that the estate class is an entity. The entity is also an extension of the identifiable entity, which makes it inherit its properties. @Column indicates the description properties will be the columns for the tables, followed by a @Required as the first property being estateName will be required to be filled in. There's another @Stereotype annotation which indicates a special use of a type and for this class the type is a photo for the estate logo to be added, followed by @Embedded referencing address (private Address address) and then @ManyToOne relationships referencing Region and a @ManyToOne referencing Country. Getters and setters are generated.

5. **Identifiable.java**

The first three lines of code indicate the package being used and the tools needed to be imported. This class is used as a super class for inheritance and is annotated with @MappedSuperClass which allows for classes to inherit properties from this class. The properties this class has and will pass on to classes that are extensions of it will be that of oid, which will be used for ID generation.

6. **Region.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Region information" and "Variety finder" and given the description properties that would be in their respective sections in the view. @Entity indicates this class is an entity class. The entity is also an extension of the identifiable entity. The description properties for this class is name, with a string data type and a @Stereotype which allows for a special type being memo which allows for notes to be taken in the interface. The relationship between Region and variety is indicated by the @ManyToOne with variety being referenced (private Variety variety). Getters and setters are generated

7. **Score.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Connoisseur", "Scoring" and "Wine finder" and given the description properties that would be in their respective sections in the view. @Entity indicates this class is an entity class. The entity is also an extension of the identifiable entity. The description properties all have integer data values, followed by annotation called @ReadOnly which makes the field not editable, followed by annotation @Calculation which calculates total column with in parenthesis referencing what fields to add for the scores, referencing total. The relationship between Score and connoisseur is indicated with a @ManyToOne relationship referencing Conoisseur. Followed by the @Stereotype for memo which allows for notes to be taken in the interface, followed by a @ManyToOne relationship with wine. Getters and setters are generated.

8. **Variety.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Variety Information" and "Estate finder" and given the description properties that would be in their respective sections in the view. @Entity indicates this class is an entity class. The entity is also an extension of the identifiable entity inheriting its properties. @Column indicates the description properties, the first one being a string type called varietyName and the next description property is annotated with @Enumerated with in parenthesis the data type of the enumeration, the next line is "private Category category" which references where to look for the data thus being category, "public enum Category{….}" indicates the output for the enumeration. @Enumerated enables for the drop-down menu functions for continents. The relationship between variety and estate is indicated by the @ManyToOne referencing estate. Getters and setters are generated

9. **Wine.java**

The first three lines of code indicate the package being used and the tools needed to be imported. The next couple lines of code are used to control the design of the interface of the entity and this is shown by the annotation @View. Each view is titled, "Bottle Details" and "Origin finder" and given the description properties that would be in their respective sections in the view. @Entity indicates this class is an entity class. The entity is also an extension of the identifiable entity inheriting its properties. @Column indicates the description properties, the first one being a string type called label, second one being an integer type called year, @Temporal allows for the storing of dates allowing for the property dateDrank, the next @Sterotype is called "MONEY" making use of the type BigDecimal referencing price which makes the interface output the inputted amount in a currency format. The next annotation is @ManyToOne which indicates the relationship between wine and estate. Getters and setters are generated.

}