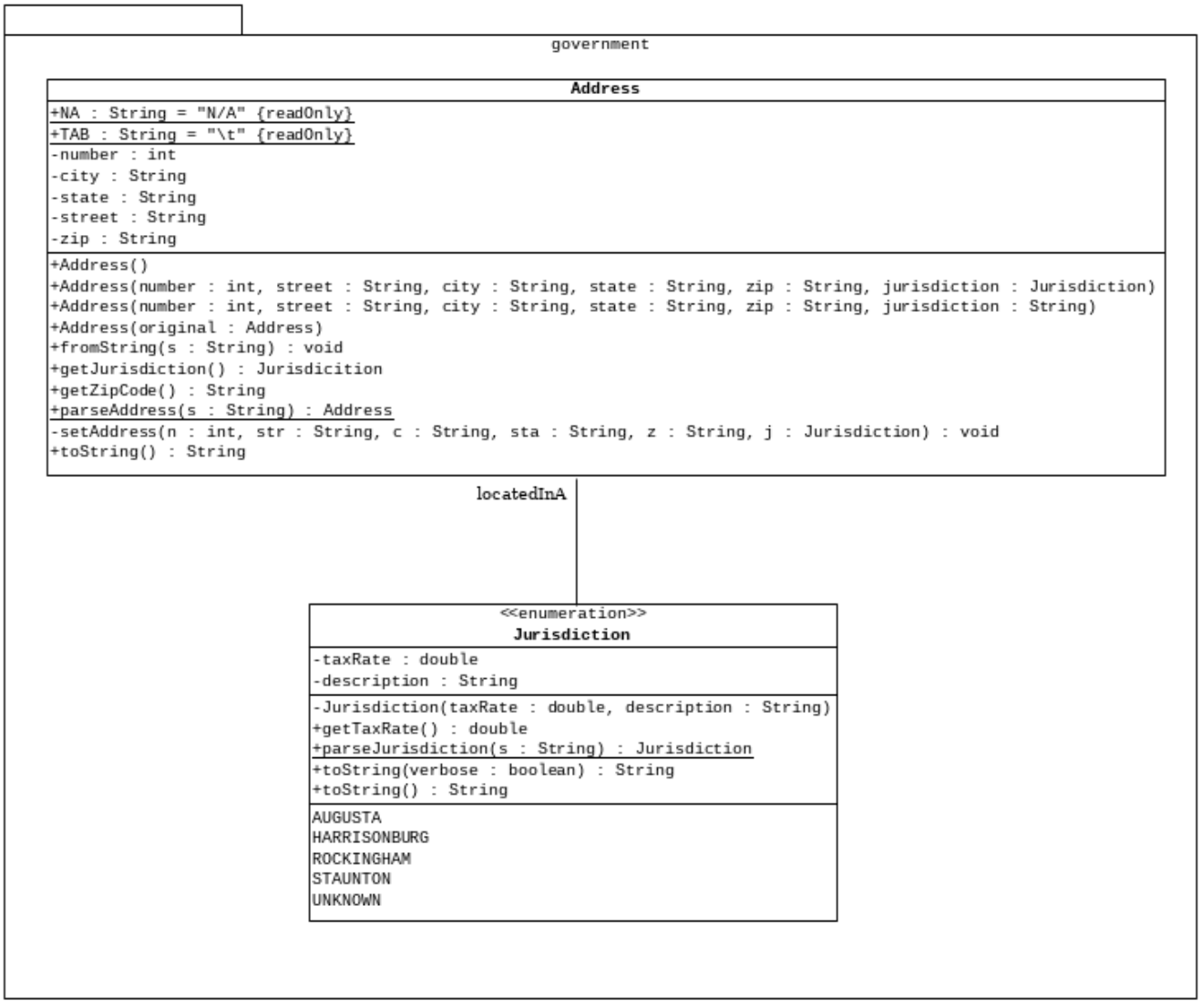
HomeBase

government Package v1.0

Class Diagram

The relationships between the various classes and interfaces of the system are illustrated in the following abstract UML class diagram.



In addition to the specifications that are contained in this class diagram, the implementation must comply with the following specifications.

The **Jurisdiction** Enum

Purpose:

`Jurisdiction` is an enumeration of the different political jurisdictions in the demonstration area.

Background:

Political jurisdictions in Virginia have the power to levy property taxes. Typically, each property has an assessed value (in hundreds of dollars) and the jurisdiction determines the tax rate (per hundred).

Virginia has an unusual system of political jurisdictions. Unlike most states, cities may not be part of the county that contains them.

The test area we are using has four political jurisdictions, Rockingham and Augusta counties, and the cities of Harrisonburg and Staunton. Augusta's tax rate is $0.85, Harrisonburg's is $0.80, Rockingham's is $0.75 and Staunton's is $1.10.

Instances:

This enumeration must define the following instance.

```
   AUGUSTA      (0.85, "Augusta"),
   HARRISONBURG (0.80, "Harrisonburg"),
   ROCKINGHAM   (0.75, "Rockingham"),
   STAUNTON     (1.10, "Staunton"),
   UNKNOWN      (0.00, "Unknown");
```

Methods:

```
    Jurisdiction parseJurisdiction(String s)
```

The purpose of this method is to return the `Jurisdiction` that corresponds to a `String` representation of its `description`. It must ignore case (e.g., `"Rockingham"`, `"ROCKINGHAM"`, and `"rocKINGham"` should all return `Jurisdiction.ROCKINGHAM`). It must return `Jurisdiction.UNKNOWN` if the `String` does not correspond to a valid `Jurisdiction`.

```
   String toString(boolean verbose)
```

This method must return a terse or verbose `String` representation of the `Jurisdiction`. The terse representation must be just the description (formatted using `%s`). The verbose representation must include the `description` and the `taxRate` (formatted using `"%s ($%4.2f per $100)"`.

```
   String toString()
```

This method must return a terse or `String` representation of the `Jurisdiction`.

The Address Class

Purpose:

`Address` is an encapsulation of an address that includes a street number, a street name, a city, a state, a Zip code, and a political jurisdiction. None of these attributes ever change.

Fields/Attributes/Instance Variables:

This class must, at a minimum, contain the following private fields:

- `number` - An `int` street number.
- `street` - A `String` containing the name of the street (which may contain spaces, periods, dashes, etc...).
- `city` - A `String` containing the name of the city/town/etc.
- `state` - A `String` containing the two-letter state abbreviation.
- `zip` - An `String` containing the Zip/postal code.
- `jurisdiction` - A `Jurisdiction` containing the political jurisdiction.

It may contain other private fields as well.

Constructors:

```
    Address()
```

This method must construct an `Address` object with `"N/A"` for all `String` attributes, `0` for all numeric attributes, and Jurisdiction.`UNKNOWN` for the `Jurisdiction`.

Methods:

```
    void fromString(String s)
```

This method must parse a tab-delimited `String` representation of an `Address` and set the owning `Address` object's attributes accordingly. If there is a problem with the `String` representation, it must leave the `Address` unchanged. Note that the `String` representation may contain "extra" tabs (to make the `String` more human readable).

```
    Address parseAddress(String s)
```

This is a factory method that must construct an `Address` from a `String` representation.

```
    String toString()
```

This method must return a tab-delimited `String` representation of this `Address`. (The `String` returned by this method can be processed by the `fromString()` method.)