Hanoi University of Science and Technology
School of Information and Communication Technology

Mini Project Report

Demonstration of types of viruses and its mechanism

Class	149324 — Object-oriented Programming	
Lecturer	Nguyen Thi Thu Trang	
Team members	Group 21	
	Chu Trung Anh	20225564
	Phan Tran Viet Bach	20225435
	Nguyen Huu Cong	20225476

Hanoi, 2024

Contents

1. Assignment

Each team member has been assigned specific tasks to ensure the efficient and effective development of our virus visualization application.The tasks are divided into several key areas, each critical to the project’s success.

- Chu Trung Anh - 20225564 (leader)
  - Diagram 30%
  - Package gui 40%
  - Package virus 20%
  - Report 30%
  - README 50%
  - Slide 20%
- Phan Tran Viet Bach - 20225435
  - Diagram 40%
  - Package gui 20%
  - Package virus 60%
  - Report 50%
  - README 20%
  - Slide 30%
- Nguyen Huu Cong - 20225476
  - Diagram 30%
  - Package gui 40%
  - Package virus 20%
  - Report 20%
  - README 30%
  - Slide 50%

2. Mini-project description

2.1 Overview

This mini-project leverages Object-Oriented Programming (OOP) techniques using Java to create an interactive application that demonstrates the structure and mechanisms of viruses. The app provides a detailed visualization of various viruses, illustrating their genetic material, protein shells, and envelope structures. Users can explore how viruses infect host cells, replicate their genomes, and assemble new virus particles through dynamic simulations.
The team will collaborate and share information via GitHub throughout the project.
To effectively develop this application, they will design use-case diagrams and class diagrams during the development phase.
All project ideas, along with the reasons for choosing them, must be detailed in the report and the team's presentation.

2.2 Requirement

2.2.1 Basic knowledge

- **Basic structure of virus:**
  - Every virus consists of two fundamental components: nucleic acid and a capsid.
  - Based on their structure, viruses are classified into two categories: those with a lipid envelope and those without.
    - Viruses without an envelope will dissolve their capsid upon reaching the target cell.
    - Viruses with an envelope typically have anchors known as glycoproteins. The infection mechanism in this case operates like a lock-and-key system: upon encountering a host cell with a compatible outer structure, the virus uses its glycoproteins to attach and then injects its nucleic acid into the cell.

2.2.2 Specifications

- **GUI:** Free of choice; however, the main focus of this project is on structuring the application using OOP principles, so excessive focus on the interface is not necessary.
- **Design:** the application must have these functions:
  - It should display the title of the application, options to choose between a virus with a lipid envelope and one without, a help menu, and a quit button.

  - Users can select either type of virus from the main menu to start the application.
  - After selecting the desired type, the application will display various viruses for the user to choose from (for example, if the user selects a virus with a lipid envelope, the application might display HIV, COVID, and Rotaviruses for the user to choose from. The selection of viruses for demonstration is up to you).
  - The help menu provides basic usage instructions and explains the purpose of the application.
  - The quit button exits the application, with a confirmation prompt.
- In the demonstration:
  - Display the structure of the virus. Each virus has a different structure, so clearly display and explain these differences.
  - Include a button to start demonstrating how the virus infects the host cell. While different viruses share a basic infection mechanism, there are minor differences that should be highlighted.
- Ensure there is a return button for the user to go back to the main menu at any time.

2.3 Use case diagram

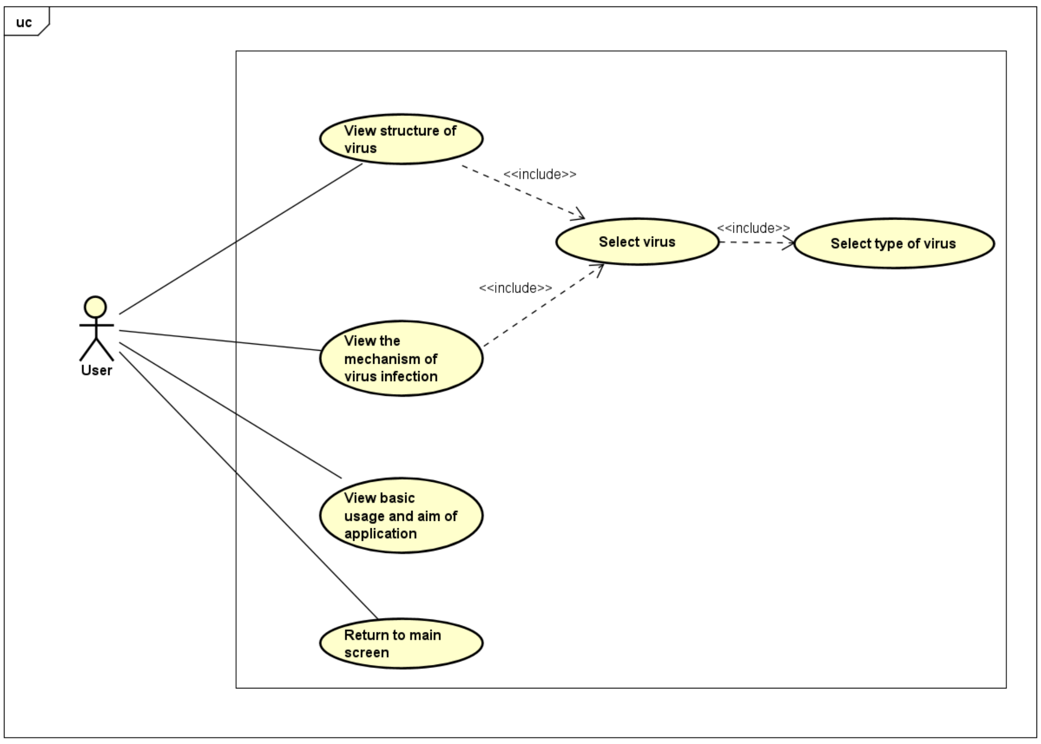

Figure 2.1: Use case diagram

Explanation

- **Actor:**
  - **User:** The person who uses the system and performs various actions within it.
- **Use Cases:**
  - **View structure of virus:** The user can view information about the structure of a virus.
  - **View the mechanism of virus infection:** The user can view information about how a virus infects.
  - **View basic usage and aim of application:** The user can view the basic usage instructions and the purpose of the application.
  - **Return to main screen:** The user can return to the main screen of the application.
- **Includes:**
  - **Select virus:** This use case is included in both ”View structure of virus” and ”View the mechanism of virus infection.” This means that before viewing the structure or mechanism of infection of a virus, the user needs to select a specific virus.
  - **Select type of virus:** This use case is included in ”Select virus,” meaning that in order to select a specific virus, the user first needs to choose the type of virus.

How it works

- **Select type of virus:** The user begins by choosing between viruses with a lipid envelope and those without, in the ”Select type of virus” step.
- **Select virus:** After selecting the type, the user chooses a specific virus they wish to view..
- **View structure of virus:** To see the structure of the virus, the user selects the virus first and choose ”Structure” tab.
- **View the mechanism of virus infection:** Similarly, to view the infection mechanism of the virus, the user must first select the virus and open ”Infect mechanism” tab.

3. Design

3.1 General class diagram

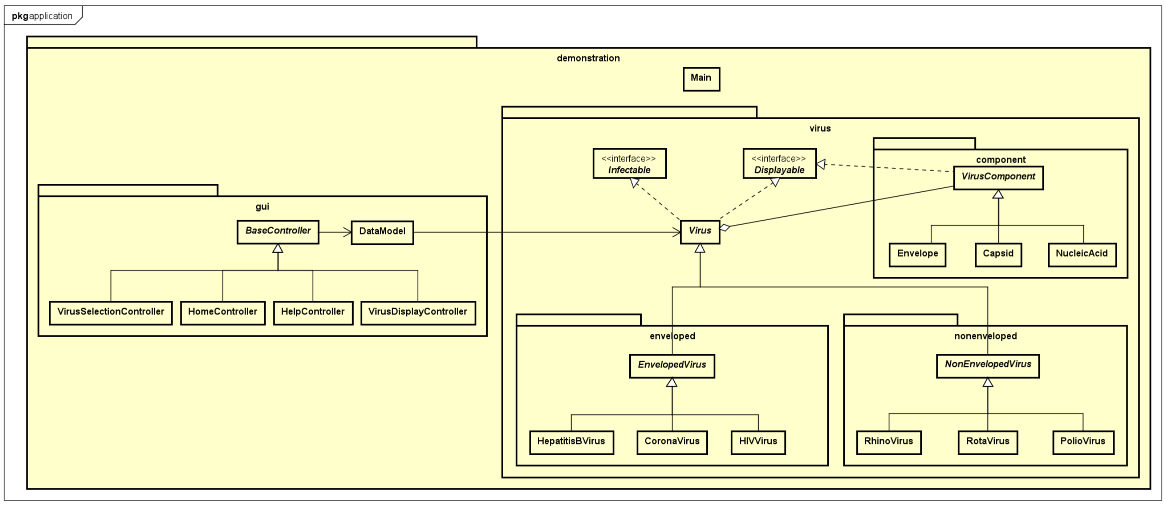

Figure 3.1: General class diagram

Main class: This is the entry point of the application.

GUI package: This package contains the classes related to the graphical user interface (GUI) of the application. It includes the BaseController class, which is an abstract class that all other controller classes extend. The DataModel class in this package is used to store the state of the GUI, specifically the currently selected virus and its type. Other classes like HomeController and HelpController are specific controllers for different scenes in the application.

Virus package: At the core, the Virus class serves as the base for all viruses, encapsulating common properties and behaviors. It implements two interfaces: Infectable, which defines methods related to infection mechanisms, and Displayable, which outlines methods for displaying virus details.

Derived from Virus, there are two main subclasses: EnvelopedVirus and NonEnvelopedVirus. EnvelopedVirus further extends to specific viruses such as HIV, Coronavirus, and Hepatitis B virus, each with unique characteristics related to their envelope structures and infection mechanisms. Similarly, NonEnvelopedVirus extends to specific viruses like Poliovirus, Rotavirus, and Rhinovirus, focusing on viruses without lipid envelopes.

The VirusComponent class represents general components of a virus, such as Envelope, Capsid, and NucleicAcid, each providing detailed information about their specific structures and functions. These components help in constructing detailed representations of different viruses.

Relationships in the diagram show that Virus inherits from Infectable and Displayable, and specific virus classes inherit from either EnvelopedVirus or NonEnvelopedVirus, ensuring a hierarchical structure that supports extensibility and clarity. The design allows for comprehensive modeling of various viruses and their components, facilitating detailed simulations and demonstrations of their behaviors and structures.

3.2 Detailed class diagrams

3.2.1 gui package

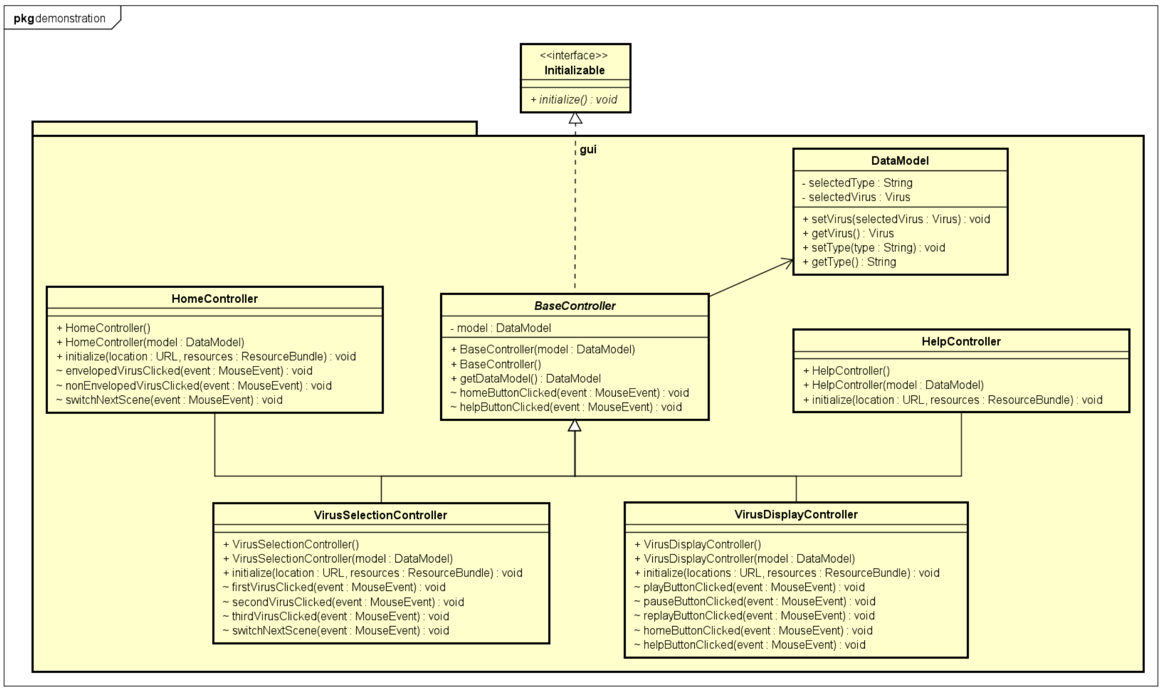
pkg demonstration
<<interface>>
Initializable
+ initialize() : void
gui
DataModel
- selectedType : String
- selectedVirus : Virus
+ setVirus(selectedVirus : Virus) : void
+ getVirus() : Virus
+ setType(type : String) : void
+ getType() : String
HomeController
+ HomeController()
+ HomeController(model : DataModel)
+ initialize(location : URL, resources : ResourceBundle) : void
~ envelopedVirusClicked(event : MouseEvent) : void
~ nonEnvelopedVirusClicked(event : MouseEvent) : void
~ switchNextScene(event : MouseEvent) : void
BaseController
- model : DataModel
+ BaseController(model : DataModel)
+ BaseController()
+ getDataModel() : DataModel
~ homeButtonClicked(event : MouseEvent) : void
~ helpButtonClicked(event : MouseEvent) : void
HelpController
+ HelpController()
+ HelpController(model : DataModel)
+ initialize(location : URL, resources : ResourceBundle) : void
VirusSelectionController
+ VirusSelectionController()
+ VirusSelectionController(model : DataModel)
+ initialize(location : URL, resources : ResourceBundle) : void
~ firstVirusClicked(event : MouseEvent) : void
~ secondVirusClicked(event : MouseEvent) : void
~ thirdVirusClicked(event : MouseEvent) : void
~ switchNextScene(event : MouseEvent) : void
VirusDisplayController
+ VirusDisplayController()
+ VirusDisplayController(model : DataModel)
+ initialize(locations : URL, resources : ResourceBundle) : void
~ playButtonClicked(event : MouseEvent) : void
~ pauseButtonClicked(event : MouseEvent) : void
~ replayButtonClicked(event : MouseEvent) : void
~ homeButtonClicked(event : MouseEvent) : void
~ helpButtonClicked(event : MouseEvent) : void

Figure 3.2: gui package

- DataModel class is used to store data in the application, specifically the type of virus chosen (enveloped or non-enveloped virus) and the virus selected for demonstration. These informations are kept as fields in a DataModel object and are accessed using setters (to pass the user's input to the model) and getters (to obtain the user's input for presentation on the UI).
- All JavaFX controllers used in this project will inherit from a BaseController abstract class, which implements the Initializable interface from the JavaFX library for scene setups. BaseController also provides common event handling methods used in FXML scenes including when the user clicks the Home or Help buttons. Additionally, to pass application data between different scenes, we modify the FXML controllers' constructor to take a DataModel as parameter and keep it within them as a field. That way, the controllers can gain access to the user's selections and provide desired information.
- For switching between scenes, we use a common procedure. We first create an FXML Loader to load FXML files, providing it with the location of the file and the corre-

sponding controller. Notice that the controller will also be provided with the DataModel for the application, which should be initially created in the Main class. Then, from the ActionEvent provided by JavaFX, we can obtain the current Stage of the application (i.e., the Stage on which the ActionEvent occurred). Finally, we set the Scene of the Stage to be the new Scene loaded by the FXML Loader as mentioned above.

- Controllers for each scene inherit from the BaseController, with modifications to suit with the scenes. HomeController handles the selection of enveloped or non-enveloped virus and passes the information to the data model before switching to virus selection scene. VirusSelectionController will then display the viruses of the selected type for the user to choose (here, we use 3 viruses for each type) and the selected virus will be recorded by the data model. Then, VirusDisplayController will load the required information from the selected virus, including virus description, image, infection mechanism, spreading methods, diseases that it causes, and visualization of the infection process by means of a video.

3.2.2 virus package

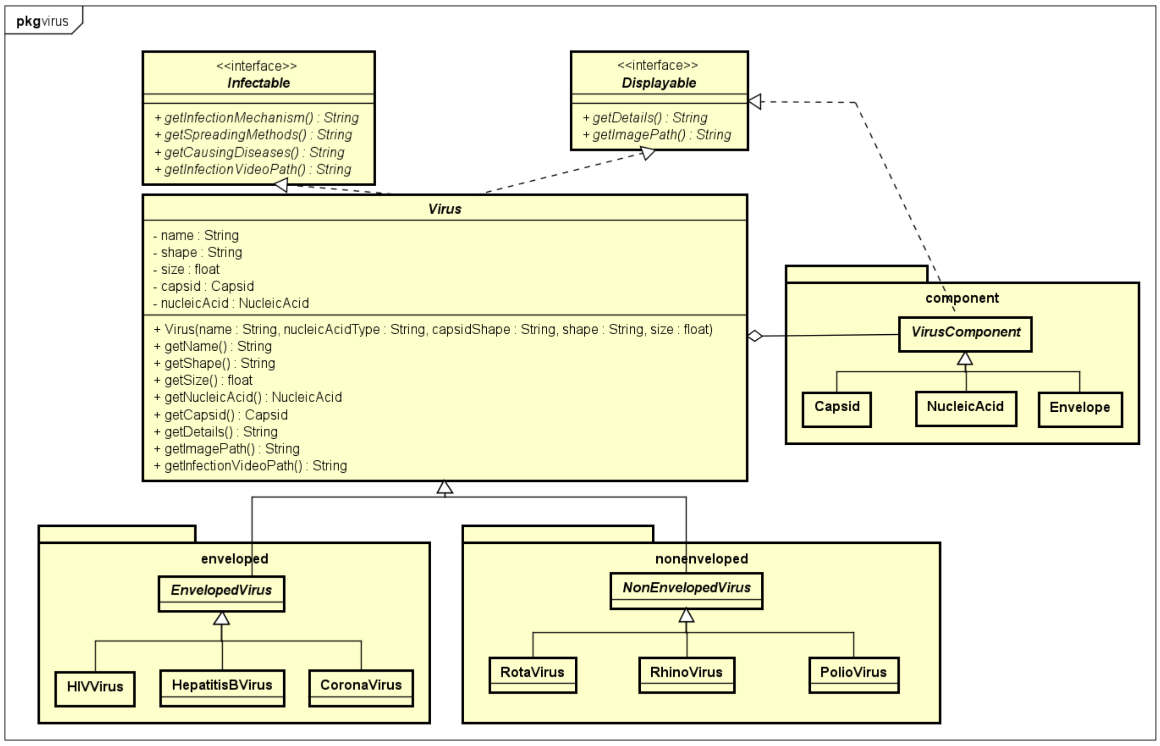

Figure 3.3: virus package

- In this project, we generalize the viruses using an abstract superclass named Virus, keeping information such as name, size (diameter, in micrometers), and shape of the virus. Along with that, components of a virus are also implemented in a subpackage named component, and these components are aggregated by Virus, including NucleicAcid and Capsid. In the case of enveloped viruses, they also have an Envelope object to represent the glycoprotein envelope. All of these fields have their corresponding getters to allow information extraction.

- We create two interfaces to represent two functionalities we expect from the viruses: an Infectable interface with methods used for demonstrating infecting-related information, and a Displayable interface for methods used in displaying details and images of viruses in the UI. The Virus class implements these two interfaces, and the VirusComponent abstract superclass from the component package also implements the Displayable interface.
- To display infecting information of a virus, we used 1 method called getInfectionVideoPath used for displaying the infection video of a virus, and 3 methods for showing details of infection, including infection mechanism, causing diseases, and spreading methods. getInfectionMechanism method is common for each of the 2 types of viruses and therefore implemented in the abstract classes EnvelopedVirus and NonEnvelopedVirus, while the other methods are overriden in each virus we create to provide unique info regarding each of them.
- Displayable interface provides 2 methods: getDetails and getImagePath. Viral components, as they do not have their own images and should not be necessary in the scope of this project, will simply leave the getImagePath empty and implement the getDetails method to provide their details. The virus will provide its own details such as name, shape, and size, and when the UI requires details of a virus, it will acquire details of the virus and those of the virus' components. Each virus will also be provided with an image of itself, and the directory will be provided to the UI controllers using getImagePath method.

3.2.2.1 component package

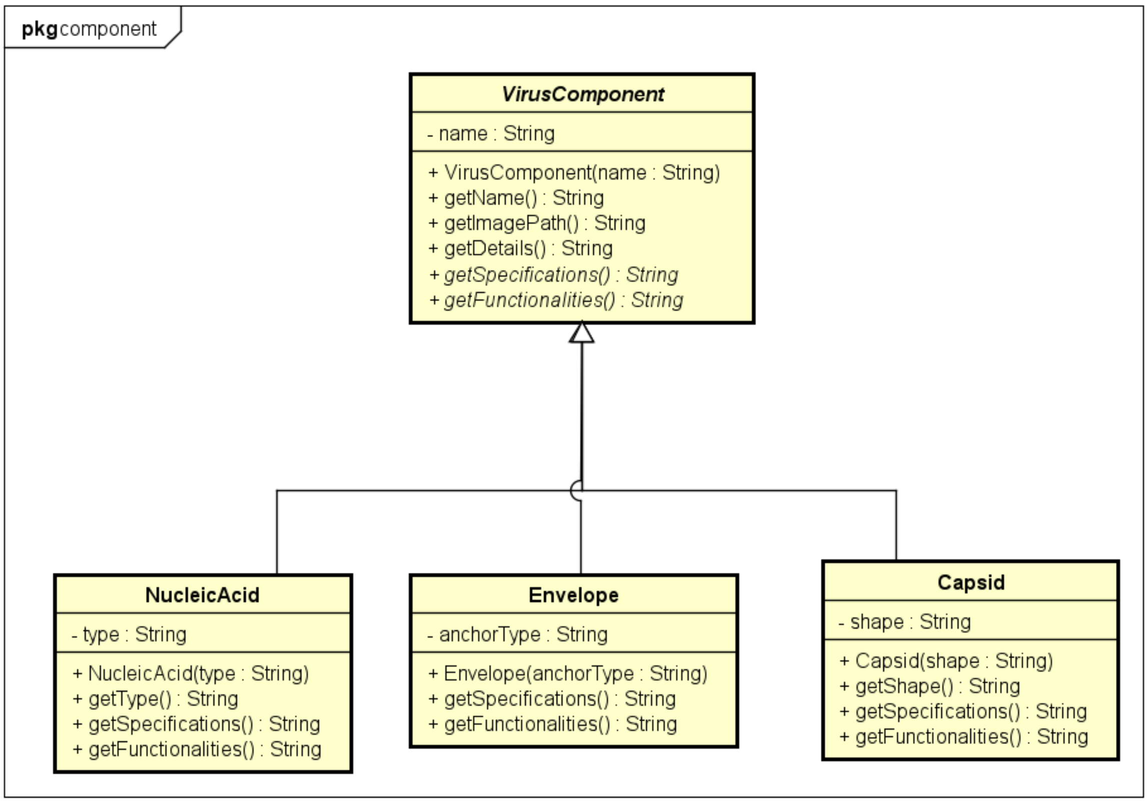
pkg component
VirusComponent
- name : String
+ VirusComponent(name : String)
+ getName() : String
+ getImagePath() : String
+ getDetails() : String
+ getSpecifications() : String
+ getFunctionalities() : String
NucleicAcid
- type : String
+ NucleicAcid(type : String)
+ getType() : String
+ getSpecifications() : String
+ getFunctionalities() : String
Envelope
- anchorType : String
+ Envelope(anchorType : String)
+ getSpecifications() : String
+ getFunctionalities() : String
Capsid
- shape : String
+ Capsid(shape : String)
+ getShape() : String
+ getSpecifications() : String
+ getFunctionalities() : String

Figure 3.4: component package

- Viral components we used in this project includes NucleicAcid, Capsid, and Envelope. They are generalized by the abstract class VirusComponent, which implements a common procedure for the getDetails method in the Displayable interface: a component will provide its specifications (e.g., type of nucleic acid, glycoprotein type of envelope) and its role to the virus. getSpecifications and getFunctionalities are 2 abstract methods that each component will have to override to provide its own information.
- Specifications of each component can be as follows: for nucleic acid, it can be either DNA or RNA; for capsid, it is the shape of the lipid capsid; for envelope, it is the type of glycoprotein used to construct the envelope.

3.2.2.2 enveloped package

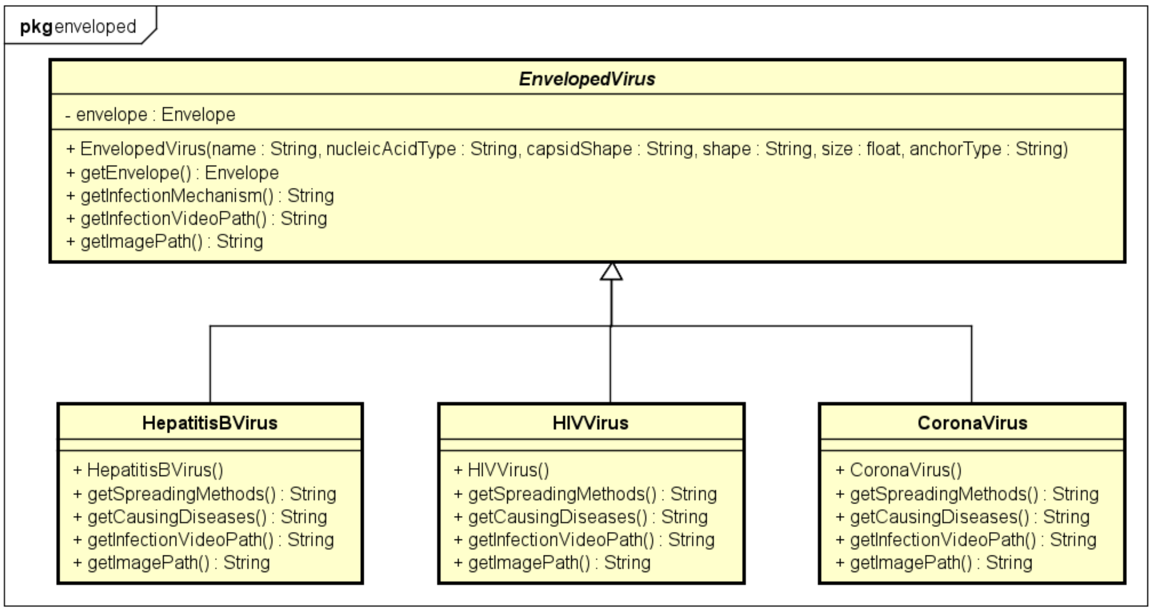
pkg enveloped
EnvelopedVirus
- envelope : Envelope
+ EnvelopedVirus(name : String, nucleicAcidType : String, capsidShape : String, shape : String, size : float, anchorType : String)
+ getEnvelope() : Envelope
+ getInfectionMechanism() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
HepatitisBVirus
+ HepatitisBVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
HIVVirus
+ HIVVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
CoronaVirus
+ CoronaVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String

Figure 3.5: enveloped package

- This subpackage contains classes for the enveloped viruses, generalized by the abstract EnvelopedVirus class that inherits from the Virus class. EnvelopedVirus has an additional envelope field, keeping an Envelope object to represent an enveloped virus' structure.
- Methods for accessing medias, getInfectionVideoPath and getImage, are overriden to return the directory storing medias of enveloped viruses. For each virus inheriting EnvelopedVirus, exact media names are then provided and added to the directory.
- getInfectionMechanism is also overriden to provide the infection mechanism of enveloped viruses, making small adjustments between different viruses by adding their names and components' names/info to better describe the infection process.
- Note that viruses also override other methods from Infectable interface to provide required specified info regarding the infecting process.

3.2.2.3 nonenveloped package

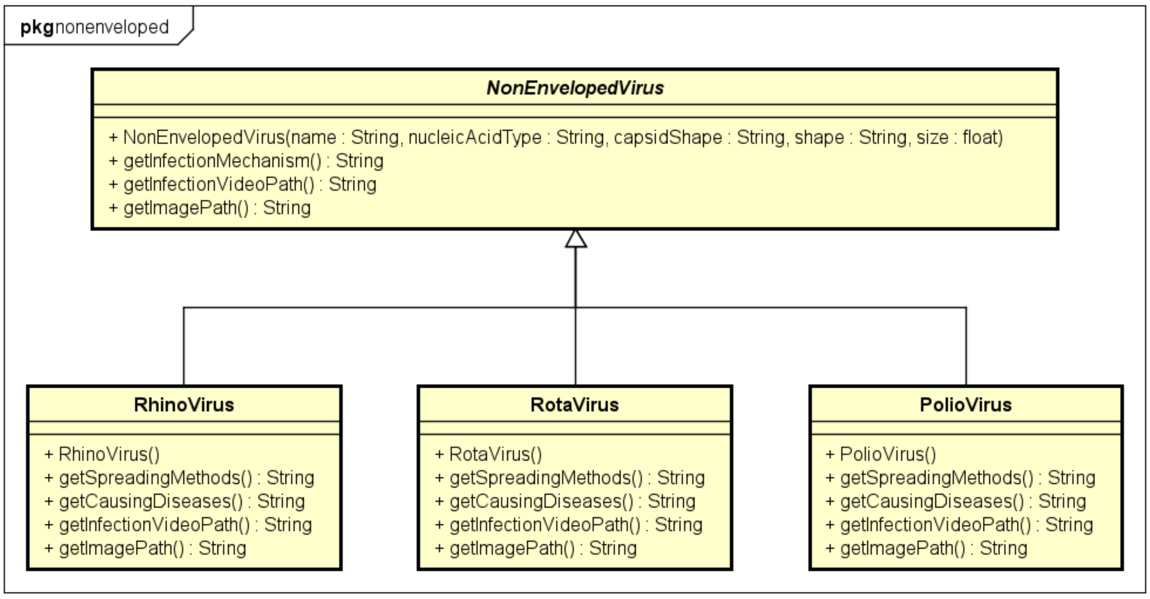
pkg nonenveloped
NonEnvelopedVirus
+ NonEnvelopedVirus(name : String, nucleicAcidType : String, capsidShape : String, shape : String, size : float)
+ getInfectionMechanism() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
RhinoVirus
+ RhinoVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
RotaVirus
+ RotaVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String
PolioVirus
+ PolioVirus()
+ getSpreadingMethods() : String
+ getCausingDiseases() : String
+ getInfectionVideoPath() : String
+ getImagePath() : String

Figure 3.6: nonenveloped package

- Similar to the enveloped subpackage, non-enveloped viruses are generalized by the abstract NonEnvelopedVirus class.
- Media-accessing methods are overriden similarly to provide directories to medias of non-enveloped viruses. Infection mechanism is also overriden similar to that of enveloped viruses, and specified information of infecting processes are also provided in each virus by overriding methods in Infectable interface.