**Hanoi University of Science and Technology**

**\_SOICT-HUST\_**

PROJECT REPORT

**Semester 20212**

**Mini-Project**

**Demonstration of type of viruses** **and its mechanism**

**Group 06:**

Hoàng Đình Dũng – 20214882

Nguyễn Tiến Doanh – 20214881

Nguyễn Việt Dũng – 20214883

Đỗ Nghiêm Đức – 20214892

**Object-Oriented Programming**

**Instructor: Prof. Nguyễn Thị Thu Trang**

**I. Assignment of member**

Hoàng Đình Dũng – 20214882

* Hav virus: image, text preparation, core code
* GUI: controller for Attack (each virus), Fix code for StructureController
* Diagram: Controller class diagram
* Report (write, summary and format)

Nguyễn Tiến Doanh – 20214881

* Covid virus: image, text preparation, core code
* GUI: MainScreenController, StructureController, fix code AttackController,…
* Diagram: usecase diagram, (initial) general class diagram
* Fix and clean all code, merge code.

Nguyễn Việt Dũng – 20214883

* Papilloma, Adeno virus: image, text preparation, core code
* Diagram: (final) general class diagram, detail core class diagram
* Report, slide

Đỗ Nghiêm Đức – 20214892

* Influenza, Chikungunya virus: image, text preparation, core code
* Diagram: (final) general class diagram, detail core class diagram
* Report, slide

**II. Mini-project Description**

1. Introduction:

The rapid global spread of COVID-19 has underscored the critical importance of understanding viruses and their modes of infection. As the world faces the ongoing challenges of this pandemic, it becomes increasingly vital to grasp the basic knowledge of different types of viruses and their mechanisms of infection in order to develop effective prevention strategies. This report aims to provide an overview of the fundamental aspects of viruses, focusing on their basic structure and the methods by which they infect host cells. In this project, Java programming and JavaFX with GUI are the way we applied to create the virus application.

2. Usecase diagram

A diagram of a company

Description automatically generated

The user launches the application and is presented with the main screen, which includes the title and multiple options to choose.The user can choose the “Help” menu to acquire familiarity with the basic usages and the aims of the application. Alternatively, the user can select the “Quit” option to exit the application.

In addition to the "Quit" and "Help" buttons, the main screen also presents options to explore the two categories of viruses: those with a lipid envelope and those without. These buttons allow users to choose their desired virus category for further investigation.

Upon selecting a specific virus category, such as viruses with a lipid envelope, the application proceeds to display a list of viruses falling under that category. For example, users may be presented with options like Covid, Influenza if users selected “Enveloped Virus” option, or HAV, Adeno if users selected “Non Envelope Virus”. By selecting one of these viruses, users can delve into detailed information about the selected virus, including its structure and infection mechanism.

Furthermore, once a specific virus is chosen, the application provides a button to start the demonstration of the virus infecting a host cell. This button triggers a visual representation of the virus's infection process, illustrating how it attaches to the host cell and injects its genetic material into the cell.

Throughout the application, a "Return" button is consistently available to allow users to navigate back to the main menu from any stage of exploration or demonstration.

3. General class diagram

A screenshot of a computer

Description automatically generated

**Relationship between classes:**

* Virus package
  + Envelope Virus and Non envelope Virus inherits from Virus. All types of viruses inherit from their respective groups, and all aggregate VirusComponent.
  + HostCell aggregates CellComponent.
  + Viruses implement Vesicle, Attack and GetField interfaces.
* Controller package
  + All types of attacks associated with each virus inherit from AttackController, which aggregates HostCell.
  + Dependencies exist between controller classes and exception classes and virus classes as well.

4. Detailed diagram

*4.1. Core diagram*

A computer screen shot of a computer flowchart

Description automatically generated

The amount of viruses can be identified are alot, each displaying distinctive attributes but the similarities still exist. That's where the idea of using polymorphism comes from.

We define a superclass called "Virus" that represents the common properties and behaviors shared by all viruses. The "Virus" class have methods such as "acidNucleic" and "capsid" that are common to all viruses.

Next, we create two subclasses: "EnvelopeVirus" and "NonEnvelopeVirus." The "EnvelopeVirus" class would inherit from the "Virus" class and add the specific properties and methods related to envelope viruses, such as "lipidBilayer" and "glycoProtein." Similarly, the "NonEnvelopeVirus" class would inherit from the "Virus" class.

Now, we can write code that can work with both envelope and non-envelope viruses, treating them as instances of the “Virus” class. For example, we can create multiple "Virus”  objects and populate it with instances of both "EnvelopeVirus" and “NonEnvelopeVirus” objects. Then, we can iterate through them and call the common methods defined in the "Virus” class such as constructor or accessor, and the code will automatically execute the appropriate behavior based on the actual type of the virus.

We also have the "VirusComponent" class as the aggregate class represents all the attributes of all "Virus" instances. "VirusComponent" contains information of Virus attributes such as "name", "structure", as well as many methods such as "ReadData" to extract data. By utilizing a "VirusComponent" class as an attribute within the "Virus" class, we can encapsulate specific components, such as the capsid, enabling more comprehensive representations of viruses within the program.

After constructing "Virus", we define the "cell" package as the victim of viruses' attack. The "HostCell" class generates and maintains instances of the target cells within the system , and "CellComponent" serves a similar role as the "VirusComponent", acting as an aggregate or container for the different cell instances. By showing the interaction between virus and cell, the application has the capability to provide users with insights into the attack methodology employed.

In addition, we define an "Attack" interface that includes methods which show the virus's attacking behavior to the cell, "getField" interface to get a Hashmap of attributes of a virus, and "Vesicle" interface that represents the vesicle created by the cell after being attacked by the virus.

**There are 4 techniques of OOP which are used in this model as below:**

* **Encapsulation**
  + Private access modifiers protect sensitive information by restricting direct access from external sources. They encapsulate the data members and data methods of a class, ensuring they are kept together and accessible only within the class itself. By designating attributes as private, their visibility is limited to the class, promoting data encapsulation and preventing unauthorized access or modification. This enhances code organization, maintains data integrity, and enhances security.
  + For example: virus components are set to be private, so they are only accessed in the class Influenza.A screen shot of a computer

    Description automatically generated
* **Inheritance**
  + Within my Java project, I make use of inheritance to create a hierarchical structure among classes, enabling me to inherit properties and behaviors from a superclass and customize them in specialized subclasses.
  + For example, viruses inherited from *EnvelopeVirus* and *NonEnvelopeVirus* classes. These two classes also inherited from *Virus* class. In addition, *AttackController* is inherited by each virus attack controller class to get some general point in attack process.
* **Abstraction**
  + Abstract class *Virus* is used to prevent creating any Object instance and optimize method structure and reduce the object to its essence so that only the necessary characteristics are exposed to the users. For example, *Virus* class generalize the general structure: image, name, capsid and acid nucleic. A screen shot of a computer

    Description automatically generated
  + In addition, *NonEnvelopeVirus* and *EnvelopeVirus* class are also abstract classes for more or specialized implementations.
* **Polymorphism**
  + It is expressed when each virus classes implements the interface *GetField*, *Vesicle* and *Attack*, then overide each method in the interfaces.
  + The way of applying polymorohism is that it enhances code modularity, reusability, and facilitates dependency injection and inversion of control principles.

*4.2. Controller diagram*

A diagram of a computer

Description automatically generated

We will explain some crucial attribute and method of each controller and the process when using:

* **MainScreenController:**
  + btnCovidPressed: choose covid virus
  + btnHavPressed: choose hav virus
  + btnInfluenzaPressed: choose influenza virus
  + btnAdenoPressed: choose adeno virus
  + btnChikungunyaPressed: choose chikungunya virus
  + btnPapillomaPressed: choose papilloma virus
* **StructureController**
  + inittialize: show virus image and set button for each component to show the detail of each one as shown below**:**

A computer screen with colorful text

Description automatically generated

* + showVirusComponent: to demonstrate detail of each virus
  + mainBtnPressed: to back to main screen
  + infectBtnPressed: to show attack process of the virus
* **AttackController:**
  + In the controller, each object is initiated and three Btnpressed are set to check if this button is pressed or not.
  + checkBtn methods are created to check if button is pressed in child classes of AttackController.
  + stepBtnPressed methods are used to show the step of attack method respectively.
    - * Step 1: the virus approach to the cell membrane.
      * Step 2: the virus attacks cell membrane and tends to go into the cell.
      * Step 3: the virus release its genetic material.
  + Note: there are only 2 stepBtnPressed method (for button 1 and 2) which is completely implemented, the 3nd one is created and leave empty, then implemented in each child class because there are differences in attack process.
* **Controller for each virus:** 
  + The virus controller overides the last step to show the differenct attack process of its own.
  + There is just controller of Papilloma virus which is set to overide even step 2 despite of the difference after atack cell membrane
* **The general mechanism of operation:** In the application, user choose the type and name of virus in MainScreenController, then the StructrueController will set control for this type of virus. After that, user can choose component to show details of virus by each component button or choose infect button to view the infect process. It will be set control by each attack controller corresponding to the virus.

5. References