**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**

Initial logic: Virus, EnvelopeVirus, NonEnvelopeVirus, VirusComponent, ,GetField, Screen class : Việt Dũng (20% idea), Đức (20% idea), Đình Dũng (20% idea), Doanh (40% : idea + implement).

**Hoàng Đình Dũng – 20214882**

* Core class : Hav class (100%), Vesicle class (100%), Attack class (100%), HostCell class (50%), CellComponent class(50%)
* GUI: 7 class Controller for Attack (each virus) (90%) , Fix code for StructureController class (10%)
* Diagram: Controller class diagram
* Report (writing, summarizing, and formatting)

**Nguyễn Tiến Doanh – 20214881**

* Core Class: Covid class(100%), 2 Exception class (100%), HostCell class(50%), CellComponent class (50%)
* GUI: MainScreenController(100%), StructureController class (90%), fix code for AttackController class (10%)
* Diagram: Use case diagram, (initial) general class diagram
* Fixing and cleaning all code, code merging.

**Nguyễn Việt Dũng – 20214883**

* Core class : Papilloma class (100%), Adeno class (100%)
* Diagram: (final) general class diagram, detailed core class diagram
* Report, slides

**Đỗ Nghiêm Đức – 20214892**

* Core class : Influenza class (100%), Chikungunya class (100%)
* Diagram: (final) general class diagram, detailed core class diagram
* Report, slides

References (we use the idea of these references)

* About static method to create an instance of a class : <https://stackoverflow.com/questions/40275951/static-method-that-returns-a-new-instance?fbclid=IwAR1yUNGpr4CZiaAai-uYqJNiaP1KpFYVCvWDYsSbrChyq7N1bDYDUv8SH5E>
* Loading image in ImageView: <https://stackoverflow.com/questions/21215299/loading-an-image-in-imageview-through-code?fbclid=IwAR0wE675ve4wa0DVyaOQGsH_A8_0OJsK6oA_BnU5SbKp4BqfjF7VSpAeJwQ>
* JavaFX Animation Tutorial: <https://www.youtube.com/watch?v=-WfyzkDodlI&list=PLhs1urmduZ2-4hFJxb2dgfWwcinNqEz0c>
* JavaFX button: <https://stackoverflow.com/questions/40097090/how-to-add-buttons-to-a-javafx-gui-via-the-controller-java-file-using-the-fxid?fbclid=IwAR2KtDysky9P_DtxJl4q1dpGuUzPXxWenSVBW0l87GunIn5hzm2ibDZ_VkU>
* At the begining of design, we copy with modify for Structure Controller, CellComponent and HostCell class from [this source](https://github.com/bluezdot/VirusLibrary), but after that, we decided to just use the idea, the basic logical, and we code again these classes. We also use the idea of MainScreen fxml. All the rest is done by us.

**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 screenshot of a computer screen

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 inheritance 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 " createVirusComponent" 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.

*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 overrides the last step to show the difference attack process of its own.
  + There is just controller of Papilloma virus which is set to override even step 2 despite of the difference after attack 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. OOP techniques.

**There are 4 techniques of OOP which are used in this project 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: In class Influenza, virus components attribute are set to be private, so they are only accessed in the class Influenza.A screen shot of a computer

    Description automatically generated A screen shot of a computer

    Description automatically generated
* **Inheritance**
  + Within our Java project, we make use of inheritance to create a hierarchical structure among classes, enabling us to inherit properties and behaviors from a superclass and customize them in specialized subclasses.
  + For example, viruses inherited from *EnvelopeVirus* and *NonEnvelopeVirus* classes, so these viruses can inherit the attribute (lipidBilayer,…) and access to pubic method. A screen shot of a computer code

    Description automatically generated
  + These two classes (NonEnvelopeVirus and EnvelopeVirus) also inherited from *Virus* class to get the basic structure of the virus (acid nucleic, capsid).
  + In addition, *AttackController* is inherited by each virus attack controller class to get some general point in attack process.
* **Abstraction**
  + The abstract class Virus is used as a blueprint for representing different types of viruses. By making Virus an abstract class, we prevent users from directly creating instances of Virus objects, as it serves as a generalization for all viruses and lacks concrete implementation details for specific virus types. Instead, it focuses on defining the essential characteristics shared by all viruses, such as image, name, capsid, and acid nucleic. This abstraction allows us to optimize the method structure and create a common interface for working with various virus types.A screen shot of a computer

    Description automatically generated
  + In addition, *NonEnvelopeVirus* and *EnvelopeVirus* class are also abstract classes for more or specialized implementations.
    - *NonEnvelopeVirus* is an abstract class that extends the Virus class, providing a more specialized implementation for viruses that do not have an envelope. EnvelopeVirus is another abstract class that extends the Virus class, specializing in viruses that possess an envelope.

A screen shot of a computer code

Description automatically generated

* + - Like *NonEnvelopeVirus, EnvelopeViru*s abstracts the common features (*lipidBiLayer* and *glycoProtein*) and functionalities of viruses with envelopes, allowing concrete subclasses to implement specific details for each type of enveloped virus.
    - A screen shot of a computer

      Description automatically generated
* **Polymorphism**
  + By having each virus class implement the interfaces GetField, Vesicle, and Attack, we enable polymorphism in our code. Polymorphism allows us to treat objects of different virus types uniformly through their common interfaces, facilitating code modularity and reusability. For example with method overriding in the interface GetField, each virus class can provide its specific implementation for retrieving fields, making the code more flexible and adaptable to different virus behaviors.

A computer screen shot of a black background

Description automatically generated A screen shot of a computer code

Description automatically generated

* + The way of applying polymorphism is that it enhances code modularity, reusability, and facilitates dependency injection and inversion of control principles.