**PROJECT REPORT**

Introductors: Prof. Nguyen Thi Thu Trang

**GROUP 3**

*Phạm Thanh Trường 20194460*

*Phan Mạnh Tuấn 20194461*

[Company name] [Company address]

**Demonstration of types of viruses and its mechanism**

*Object – oriented Programming*

# Asignment of members

Pham Thanh Truong 20194460:

* UseCase diagram
* Build screen
* Build controller
* Fix some parts in model parts

Phan Manh Tuan 20194461:

* Class Diagram
* Build package Virus
* Build package Cell
* Fix some parts in GUI

*In the others part, we work together through discord and teamviewer so we do not mention here.*

# Mini – project description

## Describe in detail about mini-project requirement

Specifications:

**- GUI:** Design the GUI with your favor

**- Design:** the application must have these functions:

*+ On the main screen:* Title of the application, options to choose between virus with lipid envelop and virus without lipid envelop, help menu and quit.

• User can choose to investigate one of the two types of viruses in the main menu to start the application.

• After choosing the desired type, the application will show a variety of viruses in order for user to select.

• The help menu shows basic usage and aim of the application.

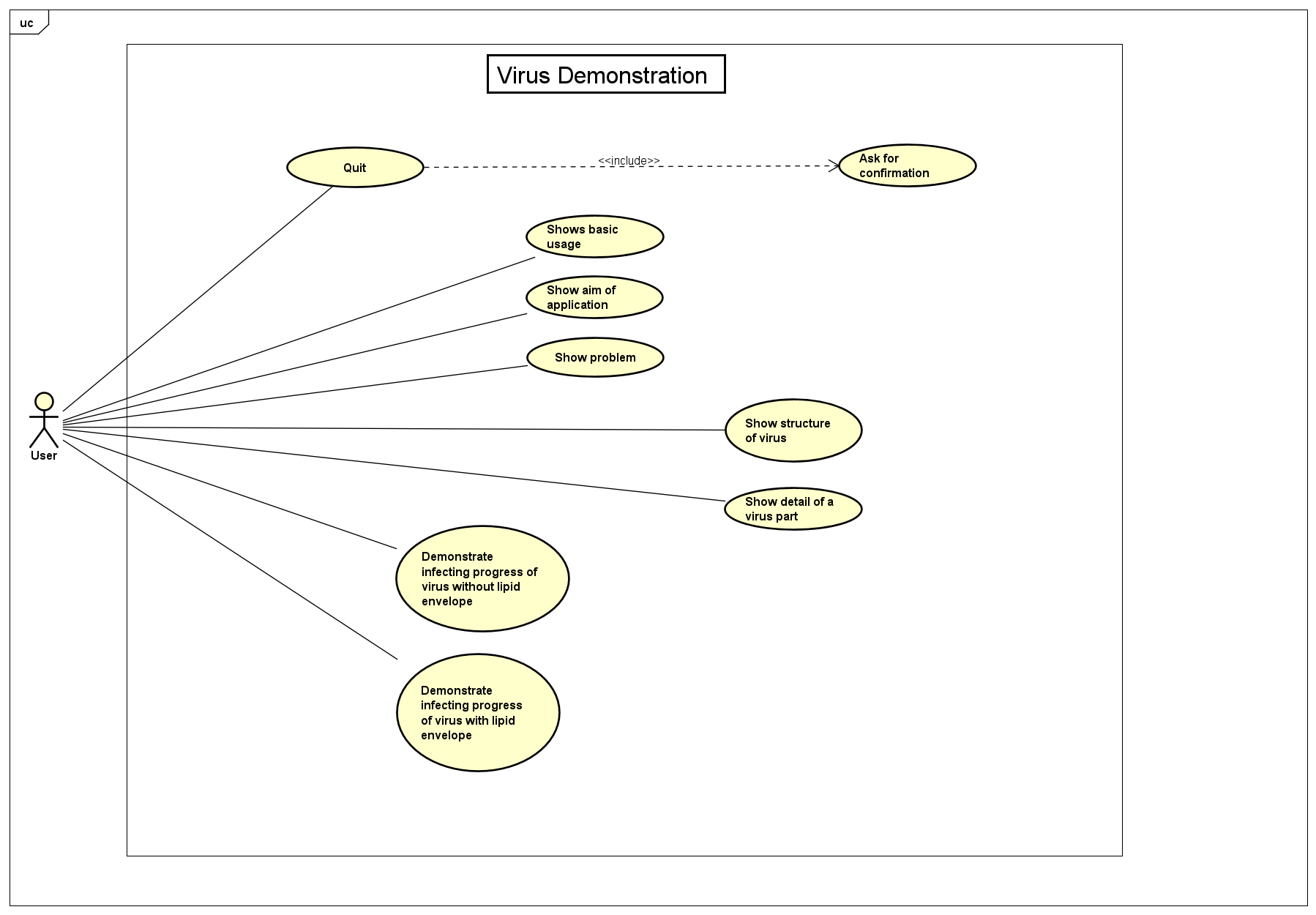
• The quit button exits the application. Be sure to ask for confirmation.

*+ In the demonstration:*

• Display the structure of the virus. Note that each virus has different structure, you should clearly display and explain them.

• One button to start demonstrating the progress of virus infecting the host cell. Different viruses have the same basic mechanism of spreading with minor difference - remember to show that There is always return button for user to get back to the main menu at any time.

## Usecase and explanation



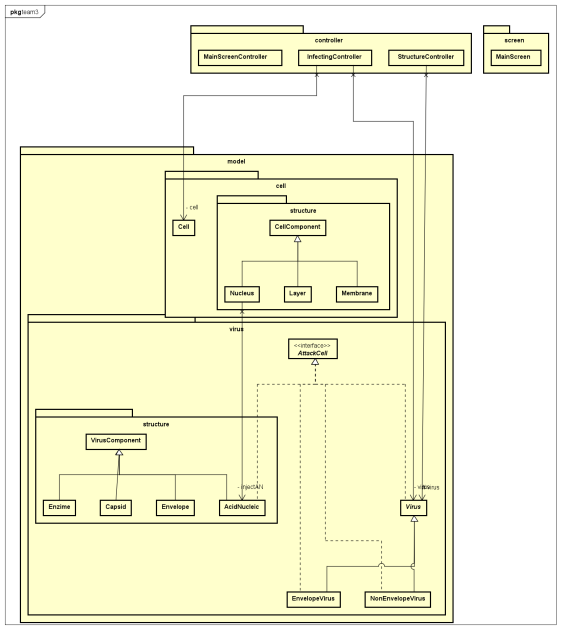
## UseCase Explaination

*Explaination:*

* First of all, our window always have a quit button with ask for confirmation.
* In the main screen, we has “Help” menu which contains “Usage”, “Aim of application” and “Problem” items on the top of scene.
* In the center, application has a split pane contain 2 list of virus (with and without envelope). Choose a virus to “show structure of virus”.
* In the StructureScreen, if you choose a part, screen will show the detail information of that part.
* In the StructureScreen, you also can choose to demonstrate infecting process to move to InfectingScreen. There’re 2 types of virus, so we have 2 infecting process type in UseCase diagram.
* Every Screen have “Back” button or “Quit” programme.

# Explanation of design ideas

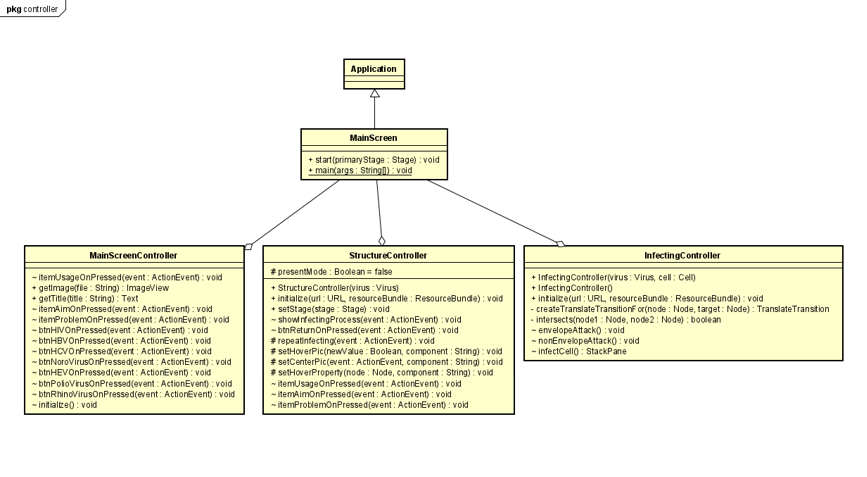
## *General class diagram*



* Package “screen” contains MainScreen, initialize the application.
* Package “controller” contains MainScreenController, StructureController, InfectingController to control the GUI and action of each scene.
* Package “model” contrain subpackage “cell” and “virus”. Each has many part inheritate from Component class. Virus has method “AttackCell”, each type of virus (Virus with lipid envelope – Virus without envelope) has a specific way to attack.

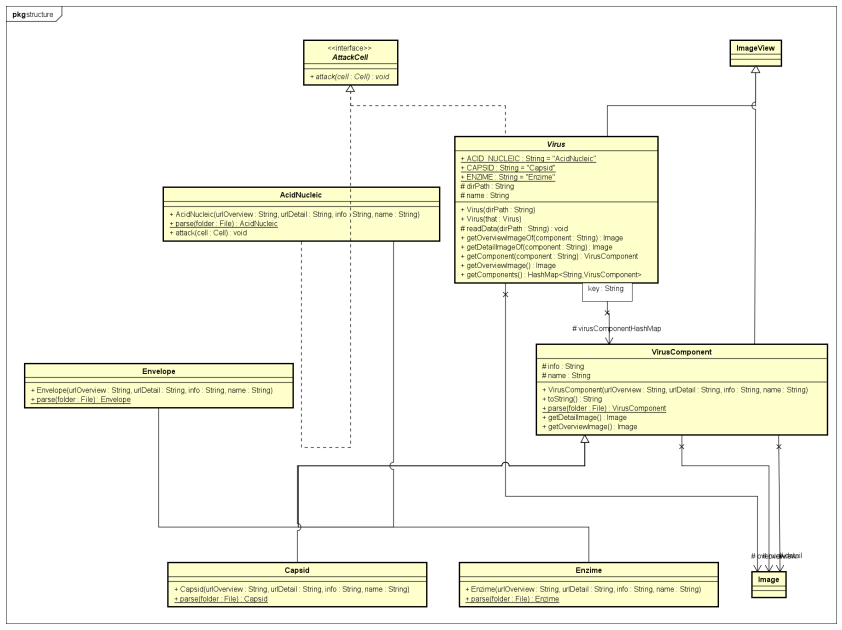
## *Class diagram for each package*

* Controller



*MainScreen* is defined inheritates from *Application* and own MainScreenController, StructureController, InfectingController.

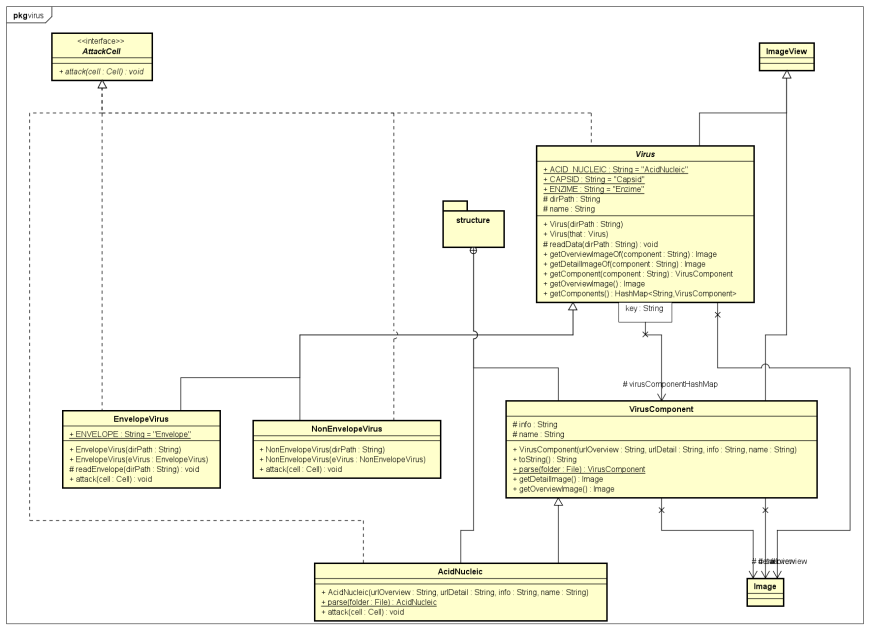
* VirusStructure



“Viruscomponent” stands for a part of real Virus.

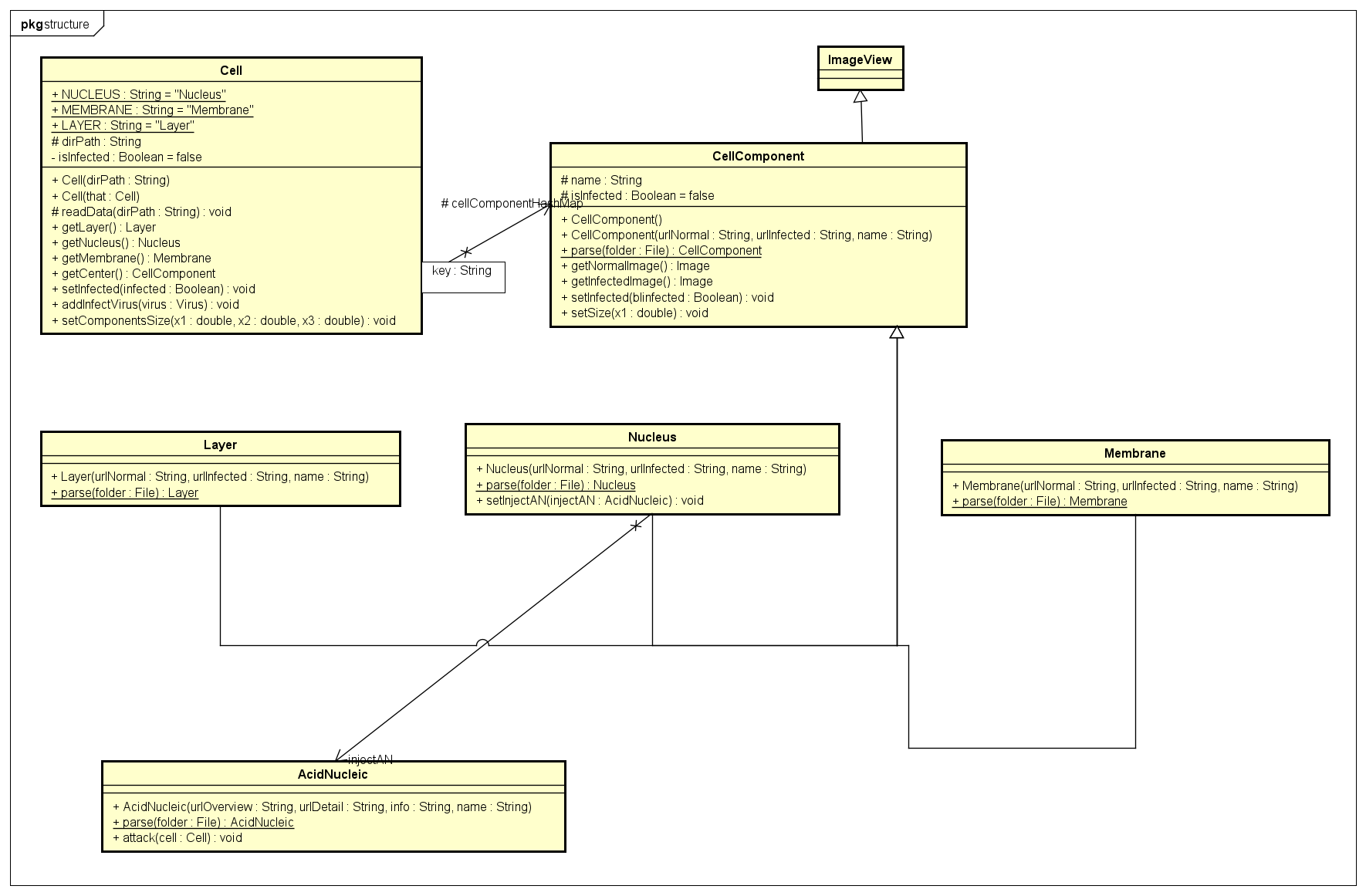
Parts of Virus: AcidNucleic, Enzime, Capsid, Envelope, ... Inheritate from VirusComponent and has some specific methods.

* Virus



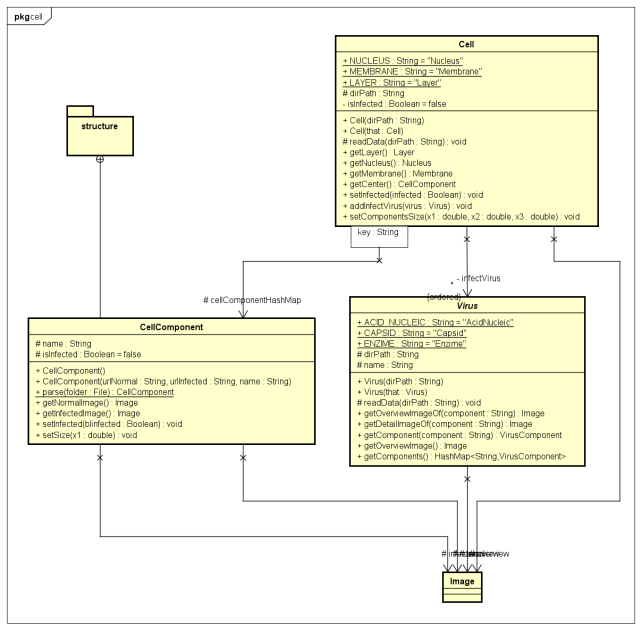
“Virus” will have a virusComponentHashMap to store all the components of a virus.

* **CellStructure**



Similar to Virus, we also create class name CellComponent to abstract CellComponent in real life. Parts of Cell: Layer, Membrance, Nucleus inheritate from CellComponent with some specific attributes and methods.

* **Cell**



“Cell” will have a cellComponentHashMap to store all components of cell.

## 3.3 Class Explanation

### 3.3.1 MainScreenController

|  |
| --- |
| **MainScreenController** |
|  |
| + getImage(file : String) : ImageView  + getTitle(title : String) : Text  ~ itemUsageOnPressed(event : ActionEvent) : void  ~ itemAimOnPressed(event : ActionEvent) : void  ~ itemProblemOnPressed(event : ActionEvent) : void  ~ btnHIVOnPressed(event : ActionEvent) : void  ~ btnHBVOnPressed(event : ActionEvent) : void  ~ btnHCVOnPressed(event : ActionEvent) : void  ~ btnNoroVirusOnPressed(event : ActionEvent) : void  ~ btnHEVOnPressed(event : ActionEvent) : void  ~ btnPolioVirusOnPressed(event : ActionEvent) : void  ~ btnRhinoVirusOnPressed(event : ActionEvent) : void  ~ initialize() : void |

*MainScreenController* has *method itemUsageOnPressed, itemAimOnPressed, itemProblemOnPressed* to handle 3 menuitems action in “Help menu”. *getImage()* is used to load image into a ScrollPane in Usage scene, *getTitle()* to set title for each Image.

Example:

@FXML  
void itemProblemOnPressed(ActionEvent event) {  
 JFrame frame = new JFrame();  
 JOptionPane.*showMessageDialog*(frame,  
 "COVID-19 has been spreading all over the world and there is the need of understanding \n" +  
 "different type of viruses, as well as the way they infect in order to have the basic knowledge to \n" +  
 "prevent them",  
 "Problem",  
 JOptionPane.*INFORMATION\_MESSAGE*);  
}

*btnHIVOnPressed, btnHBVOnPressed, btnHCVOnPressed, btnNoroVirusOnPressed, btnHEVOnPressed, btnPolioOnPressed* are 6 method to handle action “move to structure scene” of that virus.

Example:

@FXML  
void btnHIVOnPressed(ActionEvent event) throws IOException {  
 Stage primaryStage = (Stage) btnHIV.getScene().getWindow();  
 Virus hiv = new EnvelopeVirus(getClass().getClassLoader().getResource("virus/HIV/").getFile());  
  
 FXMLLoader loader = new FXMLLoader(getClass().getClassLoader().getResource("fxml/virus\_structure.fxml"));  
 StructureController hover = new StructureController(hiv);  
  
 loader.setController(hover);  
 Parent root = loader.load();  
 Scene scene = new Scene(root);  
  
 primaryStage.setTitle("Virus Structure");  
 primaryStage.setFullScreen(true);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
}

*Initialize()* method use to access @FXML components after constructer method being invoked.

### 3.3.2 StructureScreenController

|  |
| --- |
| **StructureController** |
| # presentMode : Boolean = false |
| + StructureController(virus : Virus)  + initialize(url : URL, resourceBundle : ResourceBundle) : void  ~ showInfectingProcess(event : ActionEvent) : void  ~ btnReturnOnPressed(event : ActionEvent) : void  # repeatInfecting(event : ActionEvent) : void  # setHoverPic(newValue : Boolean, component : String) : void  # setCenterPic(event : ActionEvent, component : String) : void  # setHoverProperty(node : Node, component : String) : void  ~ itemUsageOnPressed(event : ActionEvent) : void  ~ itemAimOnPressed(event : ActionEvent) : void  ~ itemProblemOnPressed(event : ActionEvent) : void |

Attribute *#presentMode* is to decide Structure Scene is being appeared or not.

*btnReturnOnPressed*: Return to MainScreen. *showInfectingProcess void* to start demonstrate the infecting process of that virus to Cell. Purpose of *setHoverProperty(Node node, String component)* is when you move the mouse into a component button field, that component will be highlighted to explore. setHoverPic(newValue : Boolean, component : String) and setCenterPic(event : ActionEvent, component : String) help user to click a component to see detail information or return to structure again.

The others method are similar to *MainStructureController.*

### 3.3.3 InfectingScreenController

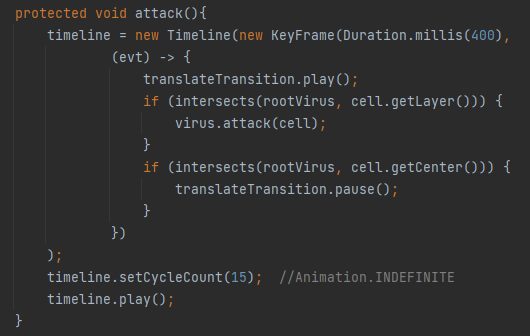
|  |
| --- |
| **InfectingController** |
|  |
| + InfectingController(virus : Virus, cell : Cell)  + InfectingController()  + initialize(url : URL, resourceBundle : ResourceBundle) : void  # createTranslateTransitionFor(node : Node, target : Node) : TranslateTransition  # intersects(node1 : Node, node2 : Node) : boolean  # attack() : void |

The last scene is InfectingScene, *InfectingController* has *createTranslateTransitionFor(*node : Node, target : Node) to create moving into cell for viruses.

protected TranslateTransition createTranslateTransitionFor(Node node, Node target) {  
 TranslateTransition translateTransition = new TranslateTransition();  
 translateTransition.setNode(node);  
 translateTransition.setDuration(Duration.*millis*(2000));  
 Bounds nodeBound = node.localToScene(node.getBoundsInLocal());  
 Bounds targetBound = target.localToScene(target.getBoundsInLocal());  
 translateTransition.setByX(targetBound.getCenterX() - nodeBound.getCenterX());  
 translateTransition.setByY(targetBound.getCenterY() - nodeBound.getCenterY());  
 return translateTransition;  
}

*Intersects(node1:Node, node2:Node*) to check whether node1 intersects with node2 or not. When the Intersects() value is True, we invoke attack() method.

*Attack() :* When we invoke this method, it’s depend on which virus we’re investigating to use different way to attack (Polimophism)



### 3.3.4 Virus Class

*Virus* defined as the superclass for *EnvelopeVirus* and *NonEnvelopeVirus* *.It* also extends from *ImageView* class which can display an *Image* in it. *Virus* models common structures and behaviors of virus. Both *EnvelopeVirus* and *NonEnvelopeVirus* contains *attack(Cell cell)* methods to attack the given cell. *Virus* is an *abstract* class which the attack method can be implemented in both *EnvelopeVirus* and *NonEnvelopeVirus* class; *attack*() method has been defined in *Virus* class because its implementation depends on the type of Virus, each Virus has different attack type. Such methods are referred to as *abstract* methods and are denoted in the interface AttackCell.

**public interface AttackCell {**

**public void attack(Cell cell);**

**}**

This class declared *dirPath* and *name* as variable which is *dirPath* is from *String* data type while *name* is also String data type. *dirPath* is a variable to store the address of the data of entire Virus. This class also declare *overview* as variable and used a *Package* from *Javafx* call *Image* which it is a class to represent an overview Image of Virus. Finally, this class also create a HashMap named *virusComponentHashMap* which hold a number of VirusComponents and there is a one to many relationship (*Association*) between two classes : Virus and VirusComponent

|  |
| --- |
| VirusComponent |
| # info : String |
| # name : String  # overview : Image  # detail : Image |
| + VirusComponent(urlOverview : String, urlDetail : String, info : String, name : String)  + toString() : String  + parse(folder : File) : VirusComponent  + getDetailImage() : Image  + getOverviewImage() : Image |
|  |

|  |
| --- |
| Virus |
| - name : String |
| - overview : javafx.scene.image.Image |
| - dirPath : String |
| - virusComponentHashMap : HashMap<String,VirusComponent> |
| + Virus(dirPath : String)  + Virus(that : Virus)  # readData(dirPath : String) : void |
| + getOverviewImageOf(component : String) : Image  + getDetailImageOf(component : String) : Image  + getComponent(component : String) : VirusComponent  + getOverviewImage() : Image |
| + getComponents() : HashMap<String,VirusComponent> |
|  |
|  |

*Virus* class have a constructor with a parameter *dirPath* . This constructor call method readData to readData from dirPath to the object. A constructor with a parameter Virus to use when cloning a Virus. It also has mutator methods *getOverviewImageOf (String component),* *getDetailImageOf(component : String)* with Image data type that will return the corresponding image of given component of Virus. getComponents() is used to get all component of Virus. This class is also declare getOverviewImage() : Image method which return the OverviewImage of Virus itself.

### 3.3.5 VirusComponent Class

*VirusComponent* has three variables which is i*nfo, name, detail and overview* .This is the superclass of some virus component like *Capsid, AcidNucleic, Envelope, Enzime.*

4 important components above have its own method, but because of time constraint we just implement detail for AcidNucleic (the most important component). *VirusComponent* is a concrete class because Virus can have many components other than 4 above which it is not very important in this project scope (but these must be showed) .

The constructor contains *urlDetail, urlOverview, info* and *name*. *urlDetail and urlOverview* contain the corresponding address to the Image, *info* contains the information about the component and name contains name of it.

*VirusComponent* class has many to one class relationship with Virus called *Association*

|  |
| --- |
| VirusComponent |
| # info : String |
| # name : String  # overview : Image  # detail : Image |
| + VirusComponent(urlOverview : String, urlDetail : String, info : String, name : String)  + toString() : String  + parse(folder : File) : VirusComponent  + getDetailImage() : Image  + getOverviewImage() : Image |
|  |

|  |
| --- |
| Virus |
| - name : String |
| - overview : javafx.scene.image.Image |
| - dirPath : String |
| - virusComponentHashMap : HashMap<String,VirusComponent> |
| + Virus(dirPath : String)  + Virus(that : Virus)  # readData(dirPath : String) : void |
| + getOverviewImageOf(component : String) : Image  + getDetailImageOf(component : String) : Image  + getComponent(component : String) : VirusComponent  + getOverviewImage() : Image |
| + getComponents() : HashMap<String,VirusComponent> |
|  |
|  |

### 3.3.6 EnvelopeVirus Class

*EnvelopeVirus* was defined as the subclass for *Virus*. It inherite all the method and attribute from *Virus* class. Moreover, it implement *attack(Cell cell)* method :

**@Override**

**public void attack(Cell cell) {**

**cell.setInfected(true);**

**cell.addInfectVirus(this);**

**}**

When this method is called, the cell set infected state to true and the EnvelopeVirus gets into the sell itself. The relationship between Cell and EnvelopeVirus is Association. The implement of Cell will be explain clearly later.

### 3.3.7 NonEnvelopeVirus Class

*NonEnvelopeVirus* was defined as the subclass for *Virus*. It inherite all the method and attribute from *Virus* class. Moreover, it implement *attack(Cell cell)* method :

**@Override**

**public void attack(Cell cell) {**

**cell.setInfected(true);**

**setImage(getDetailImageOf(ACID\_NUCLEIC));**

**this.setFitWidth(34);**

**this.setPreserveRatio(true);**

**((AcidNucleic) virusComponentHashMap.get(ACID\_NUCLEIC)).attack(cell);}**

When this method is called, the cell set infected state to true and the NonEnvelopeVirus now has only AcidNucleic remain in the ImageView. After that, it gets the AcidNucleic component and call attack method . The process above mimic the real life when non-envelope virus will inject genetic material to the host cell and all remain components will not participate in the inject process anymore. The implement of Cell and will be explain clearly later.

### 3.3.8 AcidNucleic Class

Like we mention above, for the time constraint so in 4 subclasses of *VirusComponent* we only implement the own behaviors for *AcidNucleic* 3 remaining components implements will let for future work. For the *AcidNucleic* we implement the AttackCell interface so it has to implement method *attack(Cell cell)*:

**@Override**

**public void attack(Cell cell) {**

**cell.getNucleus().setInjectAN(this);**

**}**

When method is called, the AcidNucleic will be a part of Nucleus component of Cell (will be explain later). The relation between AcidNucleic and Nucleus is *Association.*

### 3.3.9 Cell Class

*Cell*  extends from *StackPane* class which can contain many *ImageView* in it. *Cell* models common structures and behaviors of a cell.*Cell* has many common thing like *Virus,* we regconize it late so if we have time, there will be a super class of *Virus* and *Cell. Cell* class have a constructor with a parameter *dirPath* . This constructor call method readData to readData from dirPath to the object and then add all *CellComponent (extends ImageView)* to the Cell object. A constructor with a parameter *Cell* to use when cloning a Cell. It also has some mutator methods which will return corresponding Component.For the simplicity, it only have 4 Components now (detail implement later) . This class is also declare has an attribute named isInfected, the value of it will be set to true when

the Cell is attack. We need this attribute to get the right Image of CellComponent to set to the Cell Object. It has *setInfected(Boolean infected)* :

**public void setInfected(Boolean infected) {**

**isInfected = infected;**

**for (CellComponent component : cellComponentHashMap.values()) {**

**component.setInfected(infected);**

**}**

**}**

Cell class have a list to store all the Viruses attack it : infectedVirus ArrayList<Virus>;

We can add Virus to this list through a method called addInfectedVirus:

**public void addInfectVirus(Virus virus) {**

**Virus newVirus;**

**if (virus instanceof EnvelopeVirus) {**

**newVirus = new EnvelopeVirus((EnvelopeVirus) virus);**

**} else {**

**newVirus = new NonEnvelopeVirus((NonEnvelopeVirus) virus);**

**}**

**newVirus.setFitWidth(34);**

**newVirus.setPreserveRatio(true);**

**infectVirus.add(newVirus);**

**this.getChildren().add(newVirus);**

**int max = 70;**

**int min = -70;**

**newVirus.setTranslateX(((Math.random() \* (max - min)) + min));**

**newVirus.setTranslateY(((Math.random() \* (max - min)) + min));**

**}**

In *addInfectedViru*s process, The *Virus* itself it an *ImageView* has been added to the *Cell* class (which extends *StackPane*) to display to the Application. The position of the Virus with be random in the size of the Cell.

### 3.3.10 CellComponent Class

The CellComponent has 4 attributes and extend ImageView :

**protected Image normal;**

**protected Image infected;**

**protected String name;**

**protected Boolean isInfected = false;**

*normal* and *infected* respectively stand for the image when normal and when infected.

*name* is the name of CellComponent, *isInfected* is the state of the Component (default is false) , base on it will return the Corressponding Image for the CellComponent:

**public void setInfected(Boolean blinfected) {**

**isInfected = blinfected;**

**if (isInfected) {**

**setImage(infected);**

**} else {**

**setImage(normal);**

**}**

**}**

### 3.3.11 Nucleus Class

*CellComponent* has 4 children class, but for the simplicity we just implement the own behaviors for the *Nucleus* Class. It has an attribute call *injectedAN* to store the AcidNucleic of Virus injected to it.

**private AcidNucleic injectAN;**

a method setInjectAN(AcidNucleic injectAN). This method will combine the Image of Virus’s AcidNucleic and Image of Nucleus itself to one Image. After that the combine Image will be set for Nucleus (because it extends *ImageView*).