Hanoi University of Science and Technology

IT3103 - Object-Oriented Programming
Mini project report
Group 5
Demonstration of types of cell division

Team members

STT	Full name	Student ID
1	Nguyễn Văn Đăng	20215033
2	Lê Anh Đạt	20215025
3	Nguyễn Tiến Đạt	20194503
4	Nguyễn Thành Đạt	20204947

1. Members' contribution

Task	Task details	Contributed member
Designs	- General class diagram	Nguyễn Văn Đăng
	- Use case diagram	Nguyễn Văn Đăng
	- Detailed class diagram	Nguyễn Văn Đăng
Resource gathering	- Eukaryotic components and description with image	Lê Anh Đạt
	- Prokaryotic components and description with image	Nguyễn Tiến Đạt
	- Prokaryotic amitosis demonstration video/edit	Nguyễn Tiến Đạt
	- Eukaryotic mitosis demonstration video/edit	Lê Anh Đạt
	- Eukaryotic meiosis demonstration video/edit	Nguyễn Thành Đạt
Coding project	- Basic classes: Cells, prokaryotic cells, eukaryotic cells, components, process,...	Nguyễn Thành Đạt
	- Main controller / GUI design	Nguyễn Văn Đăng
	- Help screen controller / GUI design	Nguyễn Tiến Đạt
	- Choice screen controller / GUI design	Nguyễn Văn Đăng
	- Process screen controller / GUI design	Nguyễn Văn Đăng
	- Components screen controller / GUI design	Lê Anh Đạt (60%) Nguyễn Văn Đăng (40%)
Work presentation	- Demo video	Lê Anh Đạt
	- Report	Nguyễn Văn Đăng
	- Presentation slides	Nguyễn Thành Đạt

2. Project description

2.1. Project requirements

Topic: Demonstration of types of cell division

Basic knowledge: Cell cycle, Direct cell division or amitosis, Mitosis and Meiosis

Specifications:

- GUI: Freely design at will
- Design: Includes the following function:
    + Main screen: Title of the application, options to choose type of cell (prokaryotic or eukaryotic), help menu and quit
        - For each type of cell, user can choose a cell process (Amitosis, Mitosis and Meiosis) for demonstration
        - The help menu shows basic usage and aim of the application
        - Quit button exits the application after a confirmation reply from the user
    + In the demonstration:
        - Display cell components: Each components has different functions, it is required to display and explain them
        - Button to start demonstration
        - In the process demonstration screen, the bottom scroll indicate the video progress, which can be dragged or click to navigate quickly
        - The user can also choose to pause the video, skip to the next phase of the process or go back to the previous. They can chooses to replay the whole video at will

The project's requirement is to make an application to demonstrate cell's division and display cell's component of two types: Prokaryotic and Eukaryotic, using the language Java and applying four major OOP principles into the project, namely encapsulation, abstraction, polymorphism and inheritance.

2.2. Use case diagram

With the requirement mentioned above, we have come up with several use cases for the program, demonstrated in this figure below:

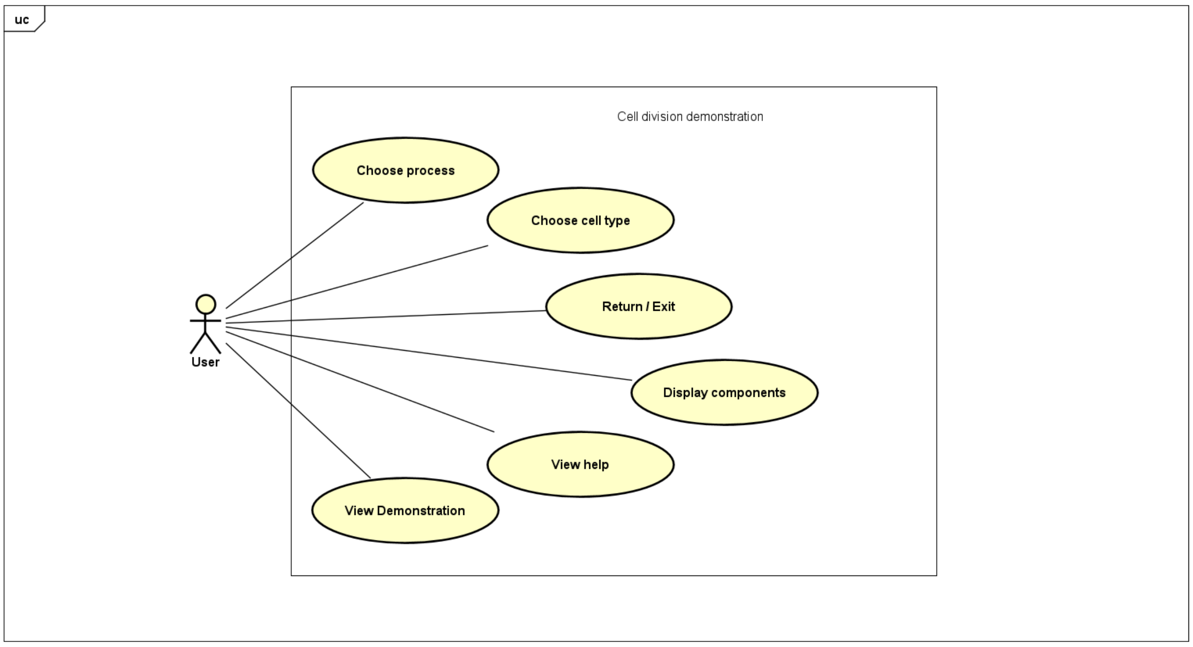


Figure 1: *Use case diagram*

The user, when in the program, can choose to view cell's components, choose cell types between prokaryotic and eukaryotic to watch other function with that type, choose to view components of each type of cell, view the help screen, view the demonstration of cell division and return/exit from the application

3. Design

3.1. General class diagram

Our initial idea is that we have a class for viewing cell process, viewing cell component, viewing help screen and a class for main screen display. After a thorough consideration, we decided that the general class diagram will be as follow:

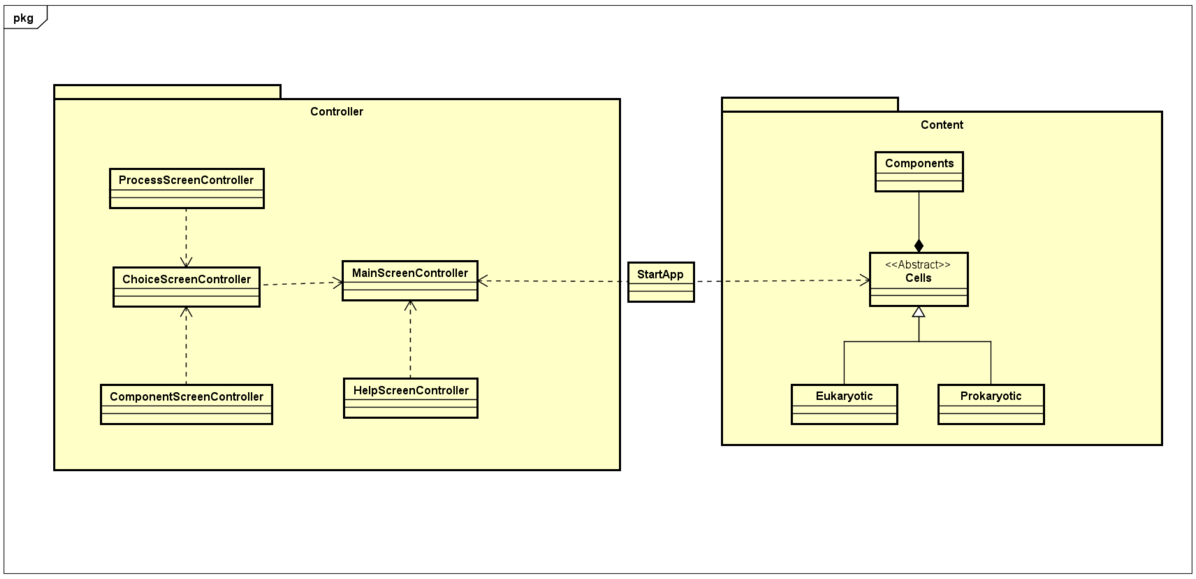


***Figure 2**: General class diagram*

3.2 Detailed class diagram

The program will consist of 2 main packages: "content" and "controller", wrapped in a bigger package named "cell_division", and another package named "screen" to hold fxml screen files, "video" that contains video for demonstration and "image" for images.

- **Content package**: Hold basic classes to support UI screens in the controller package, contains classes such as:
    + **Cells**: An abstract class, holding basic cell information such as *Name,* a String type attribute and *components* which is a type ArrayList contains multiple components of a cell, together with abstract functions for children classes to implement, which include *getDirectory()* which returns a string that represent the directory to the video of the needed process; similarly, *getImage()* for images; *getKeyFrames()* that return an array of Double elements which indicate frame (or second, to be precise) on which a new step of the process begin in the playback video demonstration; *getComponentX()* and *getComponentY()* which return arrays of Double elements that represent X and Y coordination for the button used to trigger the show function of components in the component screen, as well as three abstracts methods that represent *Amitosis(), Mitosis(), Meiosis()* which objective is to change the cells attributes respectively to pass it into the UI controllers. All of these abstract functions must be defined in its children classes, therefore the returned values are also defined in its children.

  + **Components**: A class that holds an aggregation relationship with the Cells class, which indicates that “A cell has multiple components”, or, “Components are part of a cell”. This class has basic attributes to represent its information, namely *name* and *function*, both are strings, a constructor that creates a component object with 2 strings as input (name and function).
  + **Eukaryotic**: This class is a child of the *Cells* class above, thus indicating that the relationship of *“Eukaryotic is a type of Cell.”*, it shares attributes with its parent class, together with its own attribute named *path*, a string to the process demonstration video; *keyFrames*, an array of Doubles that hold the values of the start of each phase of a process. This class must also implement all abstract functions in its parent and the interface its parent implemented. This class has a constructor that sets the name attribute, at the same time adds various hard-coded components into the ArrayList that holds the components. The objectives of the abstract functions in the parent class had been described above. Remaining are the three interface functions. This class does not have an amitosis process, thus the *Amitosis()* function is empty, the *Mitosis()* and *Meiosis()* functions set this class’s attribute *path* and *keyFrames* into the desired value respectively with the demonstration video. And the getters function to extract all the necessary values and pass it to the required controller classes.
  + **Prokaryotic:** Similar to the Eukaryotic class, the difference is that now this class must define the *Amitosis()* function and empty the other two.
- **Controller package**: This package holds the controllers to various screens designed and defined with a fxml file edited with ScreenBuilder. Almost all of these controllers have a class member named chosenCell of the type Cells which holds the value for the cell being chosen to demonstrate and view its components. The controllers have methods, also known as event handler to handle event for buttons, images, media players and sliders…:
  + **MainScreenController**: The controller for the main screen, which contains 2 imageView, one for eukaryotic, one for prokaryotic, when user clicks into these two, they will be redirected to the function choice screen respectively. 2 buttons for Help and Quit, users can choose to see the tutorial screen or close the application with:
    - *prokaryoticChosen()*: This function handles the event where the user chooses Prokaryotic type cell, which redirects the user to the choice screen controlled by the controller constructed with a Prokaryotic cell “Example”.
    - *eukaryoticChosen()*: This function is similar to the function *prokaryoticChosen()* above.
    - *btnQuitPressed()*: This function is used to pop up an alert dialog box, using the alert package, where the user can choose

and re-confirm whether they really want to exit the application or not.
  - *btnHelpPressed()*: This function redirects the user to the help screen. We hard coded an ArrayList of images directory used to pass to the *HelpScreenController*

+ **HelpScreenController**: The controller for the help screen, consists of buttons and event handler:
  - *HelpScreenController(ArrayList<String>)*: This is the constructor of this class, which takes the ArrayList directory of tutorial images to be shown to the user.
  - *ImageView:* This is the image view field to display the images
  - *Index:* An integer attribute to keep track of the position of the image displaying
  - *btnNextPressed()*: Event handler when the next button is pressed, it increases the index by one and reset the image displayed in *ImageView*. And check if the index is further able to increase or not. If not, it hides the next button away.
  - *btnPreviousPressed()*: Event handler when the previous button is pressed, it decreases the index by one and reset the image displayed. And check if the index is further lower or not. If not, hide the previous button away.
  - *btnReturnPressed()*: This button redirects the user back to the main screen.

+ **ChoiceScreenController**: The controller for the choice screen where the user was redirected to after choosing cell type. Depends on the cell type the user chose, layout of this screen change slightly:
  - *ChoiceScreenController()*: The constructor class for this controller, which sets the *chosenCell* attribute.
  - *initialize()*: Depends on the cell passed is Prokaryotic or Eukaryotic, the layout and the button text is set differently
  - *btnReturnPressed()*
  - *btnViewComponentPressed()*: Switch screen and pass *chosenCell* into that screen controller
  - *viewProcess1(), viewProcess2()*: Depending on whether the chosen cell is Prokaryotic or Eukaryotic, these functions pass the directory of the demonstration video to the process controller.

+ **ComponentScreenController**: The controller to display information, images for component viewing. This controller has a chosenCell screen and puts buttons on screen to match with the components on the image. When a user puts their cursor on the button, the displayed information will change accordingly.
  - *initialize()*: Initializer of this class, depending on the *chosenCell* passed from the previous classes, this function sets various

buttons on the screen to work as interaction points for the user, changing their opacity to 0 so it turns invisible to the user. At the same time set their handler in the case of: user hover their cursor above the buttons; user press on one of the buttons.
    - *lock*: This value indicates if the user has pressed a component button or not.
    - *lockingID*: This is an integer value that shows the currently locked component for display. If no button is currently locked, its ID is set to 0.
    - *btnReturnPressed()*

+ **ProcessController**: The controller to control the video player for the videos demonstrating the cell division process, with a slider that can help navigate the video, a play/pause button, a next and previous button that play the next/previous phase in a process.
    - *ProcessController():* This constructor set the local class member as the video directory and double array timestamp indicating process phases.
    - *initialize()*: This is the initializer of this controller, it will get the values passed from the constructor class, set necessary values for the slider. Together with various methods to acquire necessary event handlers for the slider.
    - *isPlaying*: This value indicates if the media is playing or not, serving the purpose of toggling between playing and pausing state of media player
    - *keyPoint*: This is an array of Doubles pass from the outer controllers, used to tell where should the media player go if the user press the next or previous button
    - *cell*: This is the cell passed from the previous controllers, serve the purpose of returning to the correct screen when the return button is pressed.
    - *findCheckpoint()*: This function decides where the media player goes when the user presses the next or previous phase button.
    - *btnRestartPressed(); btnNextPressed(); btnPreviousPressed()*: This three event handlers navigate the mediaPlayer to seek to appropriate time position as desired
    - *btnReturnPressed()*

With all the description above, we design the detail class diagram in the figure below:

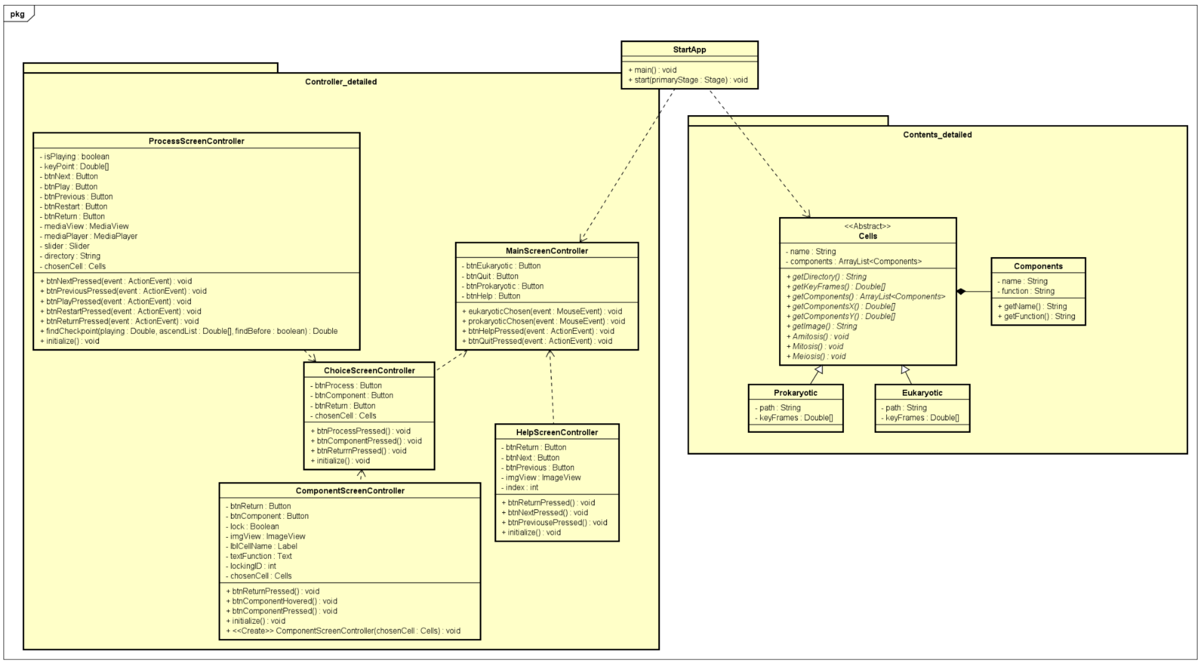


Figure 3*: Detailed class diagram*

3.3. Object-oriented programming techniques

a) Encapsulation

Encapsulation is a concept in OOP which refers to the bundling of data, along with the methods that operate on that data. It is often used to hide the internal representation, or state, of an object from the outside, a mechanism known as information hiding.

In our project, encapsulation is implemented by setting necessary class members into private, only accessible when needed with getters and setters.

b) Abstraction

Abstraction is the concept of hiding the implementation details and exposing only the functionality to the user. The user will only need to know what the buttons do, not how they are done.

In our project, the use cases are generally very simple and straightforward. The user click a button, a new screen appear, the user drag the slider, the video fast-forward or roll back, the user will not need to care about how the slider know where the video player is going, or how the media player know where to look if the user press "Next" button, or the existence of plenty attributes such as *componentX* or *componentY*.

c) Inheritance

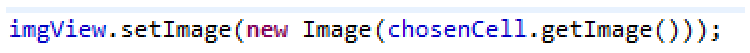
Inheritance allows a new class to inherit the properties and methods of an existing class. The new class is known as a subclass or child class, and the existing class from where the child class is derived is known as a superclass or parent class. This allows code reusability.

In our project, the "Cells" class is a parent class, whose children are "Prokaryotic" and "Eukaryotic", the children share the "name" and "components" attributes with their parents, ás well as inherit and implement all of their abstract methods.

d) Polymorphism

Polymorphism describes situations where something occurs in several different forms. In OOP, polymorphism enables users to access objects of different types through the same interface, same methods name (overwriting, overloading). Polymorphism's purpose is to enable the reuse of code, allowing the same code to work with objects of different types.

In our project, we used polymorphism in the form of method overwriting, in which the two children classes of Eukaryotic and Prokaryotic overwrite the abstract methods in their parents class - Cells, including *getDirectory(), getKeyFrames(), getComponents(), getComponentsX(), getComponentsY(), getImage(), Amitosis(), Mitosis() and Meiosis().* When these functions are called by a UI controller, for example, both Eukaryotic and Prokaryotic need to pass a directory link pointing towards its respective demonstrative image, which is done in the *getImage()* method, so when the ComponentScreenController call the methods:



```
imgView.setImage(new Image(chosenCell.getImage()));
```

In which *chosenCell* is a passed in attribute of type Eukaryotic or Prokaryotic, depending on that type, the returned value of this method is different. This is run-time (dynamic) polymorphism,

4. Overview

There are some mistakes we would like to clarify in this project:

- In the process of working on this project, there are some miscommunication happen that leads to stagnant in progressing
- There is many different types of Prokaryotic and Eukaryotic which consists of different cell structures (for example, Eukaryotic cells have animal cells, plant cells and fungi), but for the sake of completion and ease of coding, we decided to skip it and make one type as representative only
- In the process of applying release flow on Github, we did not properly apply the naming convention with the branches.
- There should be some part in the coding project that we do not include in the class diagram.
- We do not handle exceptions in classes, some unexpected bug might happen in the application's running process.

5. References

In the process of working on this project, we have made some references to some of these following sources:

1. Video used in demonstration of cell processes:
   Meiosis:
   https://www.youtube.com/watch?v=kQu6Yfrr6j0
   Amitosis (Binary fission):

https://www.youtube.com/watch?v=s4p-H5bk9xI
Mitosis:
https://www.youtube.com/watch?v=xlvLQtNF-Lk

2. Prospects of object oriented programming: PhD. Nguyen Thi Thu Trang (2021) Object-oriented Language and Theory Lecture slides - IT3103
3. General understanding of cells: https://medlineplus.gov/genetics/understanding/basics/cell/#:~:text=Cells%20are%20the%20basic%20building,and%20carry%20out%20specialized%20functions.
4. Prokaryotic and its components: Prokaryotic cells (article) | Khan Academy Prokaryotic Cell: Components, Examples with Questions and Videos (toppr.com)
5. Eukaryotic and its components: Eukaryotic Cells- Definition, Characteristics, Structure, & Examples (byjus.com)
6. JavaFX slider class: JavaFX | Slider Class - GeeksforGeeks
7. Other java language related solutions: Java Documentation - Get Started (oracle.com)
8. Display components images: Animal Cells - Cells for Infomaion (weebly.com)
9. Prokaryotic Cells- Definition, Structure, Characteristics, and Examples (byjus.com)]
10. Some other technical coding issues: Stack Overflow - Where Developers Learn, Share, & Build Careers