Hanoi University of Science and Technology

Icon

Description automatically generated with medium confidence

IT3100E-131678 Object-oriented Programming

Semester 20212

Mini-Project

**Demonstration of types of cell division**

*Lecturer: PhD. Nguyen Thi Thu Trang*

Group 13

|  |  |
| --- | --- |
| Student Name | Student ID |
| Trần Cát Khánh | 20204916 |
| Hồ Minh Khôi | 20204917 |
| Trần Ngọc Khánh | 20200326 |
| Trần Quốc Khánh | 20204881 |

**TABLE OF CONTENTS**

1.Assignment of Members.............................................................................1

2.Mini-project description.............................................................................3

2.1 THEORETICAL OVERVIEW

2.2 PROJECT OVERVIEW

2.3 DESIGN DETAILS

3.Use case diagram and explanation........................................................5

4.Class diagram and explanation.............................................................7

4.1 GENERAL DESIGN

4.2 DESIGN DETAILS

5.Limitations and further extensions...................................................11

5.1 LIMITATIONS

5.2 FURTHER EXTENSIONS

6.References...............................................................................................11

# Assignment of members

| Task | Subtask | Member |
| --- | --- | --- |
| Use case diagram  Class diagram | Early version | Hồ Minh Khôi |
| Final version - Use case diagram | Trần Cát Khánh |
| Final version - Class diagram | Hồ Minh Khôi |
| Initial logic | General design | Hồ Minh Khôi |
| Components & functionality - coding | Trần Cát Khánh (50%)  Hồ Minh Khôi (50%) |
| GUI design - menu | Main menu | Hồ Minh Khôi |
| Main action events | Hồ Minh Khôi |
| Linking action event | Trần Quốc Khánh |
| GUI design - album view | FXML of album view | Trần Cát Khánh |
| Album view logic | Trần Cát Khánh |
| Help menu | Hồ Minh Khôi |
| GUI design - media player | Media player - first build | Trần Ngọc Khánh |
| Media player - set keyframes - if/else | Trần Quốc Khánh |
| Media player - set keyframes - loop | Hồ Minh Khôi |
| GUI design - component view | Component view + controller for Prokaryotes | Trần Cát Khánh |
| Component view + controller for Animal + Fungi | Trần Ngọc Khánh |
| Component view + controller for Plant + Protist | Trần Quốc Khánh |
| Documents preparation | Cell component images & Photoshop-ing to extract layers | Hồ Minh Khôi |
| Images for album view | Trần Cát Khánh |
| Videos required for media player | Trần Quốc Khánh (50%)  Trần Ngọc Khánh (50%) |
| Editing videos (Video for media player + Demo video) | Hồ Minh Khôi |
| Components’ name and functions | Trần Quốc Khánh (50%)  Trần Ngọc Khánh (50%) |
| Report and Slides | Report outline | Trần Ngọc Khánh (50%)  Trần Cát Khánh (50%) |
| Finishing report | Hồ Minh Khôi (80%)  Trần Cát Khánh (20%) |
| Slides preparation | Trần Cát Khánh (60%)  Trần Ngọc Khánh (20%)  Trần Quốc Khánh (20%) |

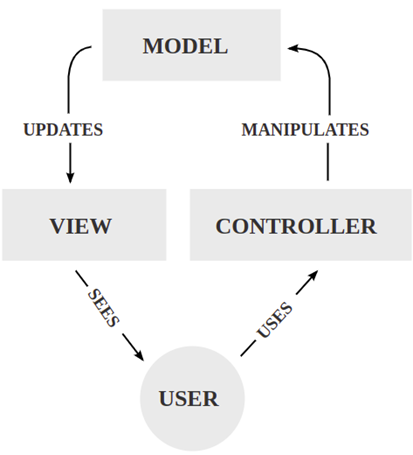
# Mini project description

**2.1 THEORETICAL OVERVIEW**

**A, Model - View - Controller (MVC) Pattern**

A MVC pattern is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements:

* **Model**: The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
* **View**: Rendering presentation of the data (from model) in a particular format to user.
* **Controller**: Catch interactions between the user and the application, also modify the View to adapt with the actions of the user.



***Figure 1***. *MVC Pattern (Source: Wikipedia)*

**B, OOP Techniques**

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance and polymorphism.

**Encapsulation**

Encapsulation works by forming a protective barrier around the information contained within a class from the rest of the code.

In OOP, we encapsulate by binding the data and functions that operate on that data into a single unit known as the class. This hides private details of a class from the outside world and only exposes functionality important for interfacing with it.

**Abstraction**

We are abstracting away the implementation details of a class and only presenting a clean, easy-to-use interface via the class’s member functions.

**Inheritance**

Classes can be organized into hierarchies where a class might have one or more parent or child classes. If a class has a parent class, we say it is derived or inherited from the parent class and it represents an “IS-A” type relationship. That is to say, the child class “IS-A” type of the parent class.

Therefore, if a class inherits from another class, it automatically obtains much of the same functionality and properties from that class and can be extended to contain separate code and data. A nice feature of inheritance is that it often leads to good code reuse since a parent class’s functions don’t need to be re-defined in any of its child classes.

**Polymorphism**

In OOP, polymorphism allows for the uniform treatment of classes in a hierarchy. Because derived objects share the same interface as their parents, the calling code can call any function in that class’ interface. At run-time, the appropriate function will be called depending on the type of object passed leading to possibly different behaviors.

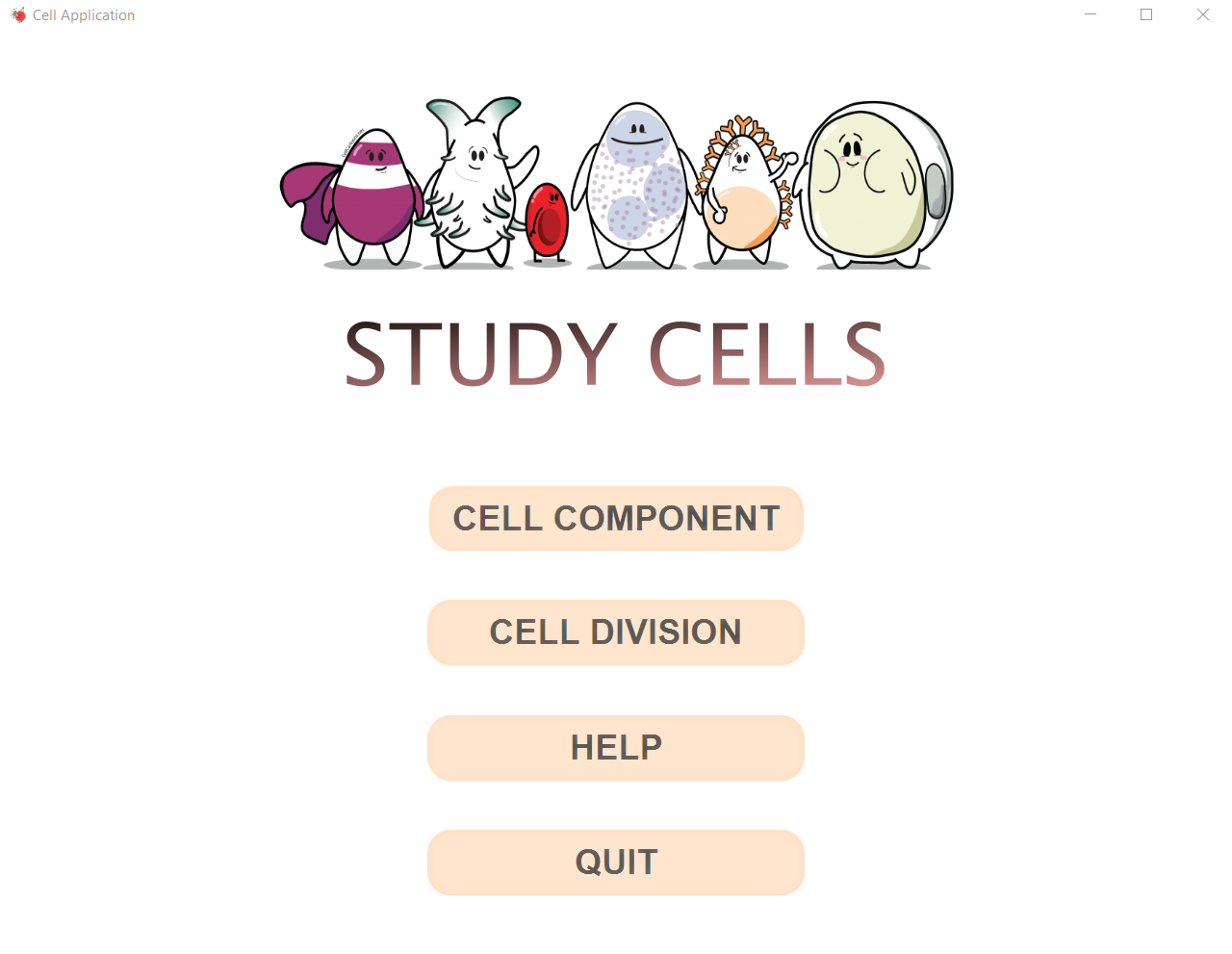
**2.2 PROJECT OVERVIEW**

- The mission of our project is to build an application which visualizes the cell division process of 2 types of cells, Prokaryotic cells and Eukaryotic cells, as well as visualizing components of the cells and their functions.

- You can view our project [here](https://github.com/hmkhoi2701/OOP.DSAI.20212.13) or watch our demo [on Youtube](https://www.youtube.com/watch?v=XCg1tfyJsj4).

**2.3 DESIGN DETAILS**

- On the main screen:



***Figure 2.*** *Launch screen of Cell App*

Title of the application, options for user to choose the function they want (view cell component or view division process), help menu,and quit.

• The help menu shows basic usage of the application.

• The quit button exits the application. The application should ask for confirmation before closing

• For view cell component function:

Ask the user to select the type of cell (Eukaryotes or Prokaryotes), then, the specific cell.

Then, a album related to that type of cell shows up as a introduction to the user.

The user can swap between the album and viewing structures of the cell.

When viewing structures, the user can click on the name of the component to view the name of the component and its function.

Throughout the process, there’s always a Back button for the user to return.

• For view cell division function:

Ask the user to select the type of cell, then, the division method related to that cell.

Then, a media player will appear corresponding to the division method.

Within the media player, there is a bottom bar with the progress bar of the executing phase, the user can also choose to move to the previous or next phase; or pause, continue, stop, or replay.

Throughout the process, there’s always a Back button for the user to return.

• Additionally, a button to Github and a button to Youtube demo of the app is present on the main screen.

# Use case diagram and explanation

Based on all the requirements we decided to develop five use cases (as shown in the figure for our application. To be more specific:

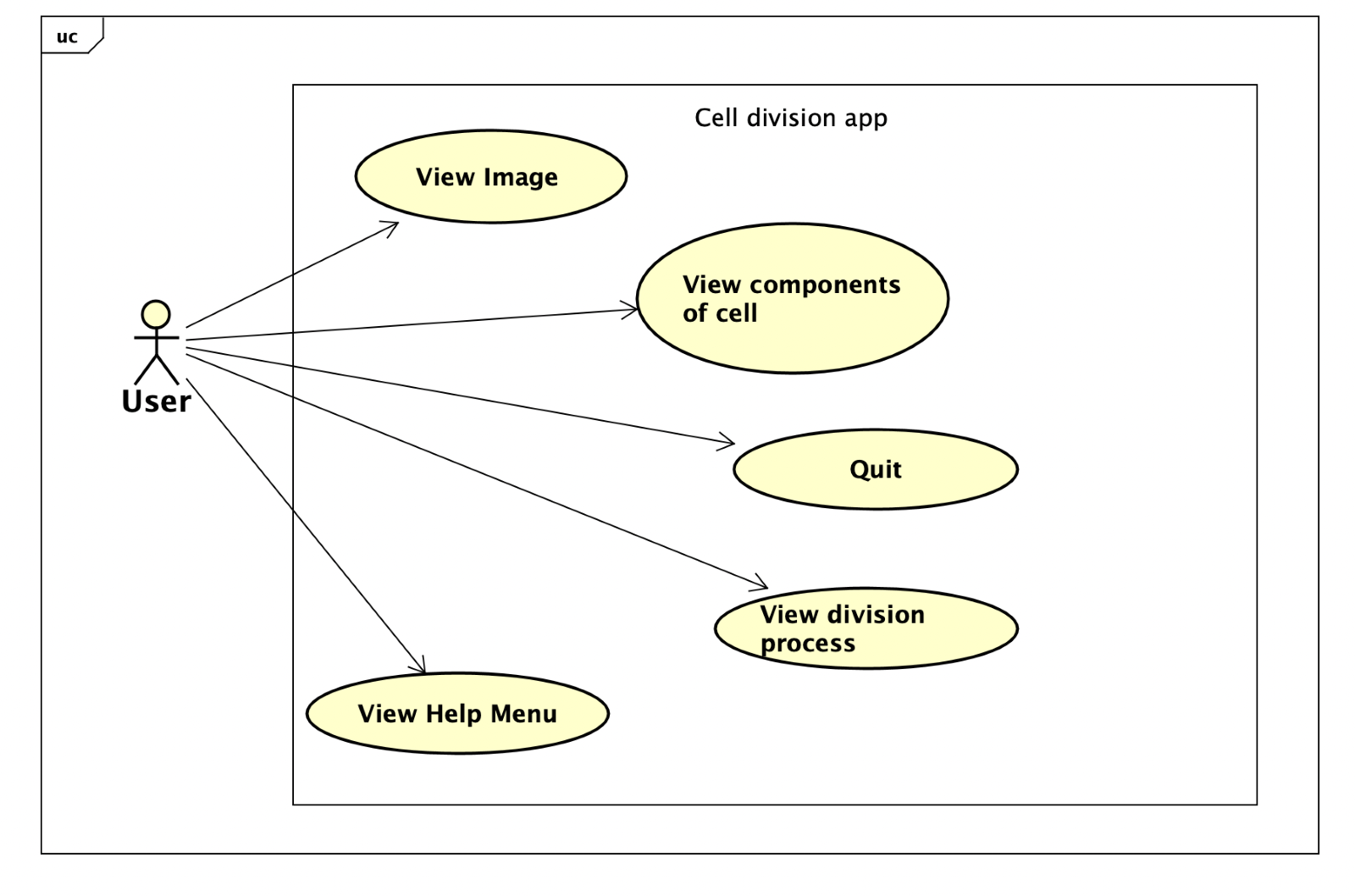
- For the 3 view use cases, the application will:

First, the software will show two types of cells for the user: prokaryotic and eukaryotic. The user interacts with the software by choosing one of them.

1. View Image, The software will continue showing different examples of each type, and the user can choose one of these examples to investigate. The software will also allow the user to a new screen, which has View Album Image and has View Components Option.

2. View components of cell, The software will show the components of the type of cell and can click on the components to find out its name, its image and its function.

3. View division process of cell , The software will show the cell division process of the type of cell the user chooses .



***Figure 3.*** *Use Case Diagram*

- View help menu use case: Show the user manual to the user.

- Quit use case: Ask the user for confirmation whether the user wants to leave/ continue staying in the app. Then close/remain the current window.

# Class diagram and explanation

**4.1 GENERAL DESIGN**

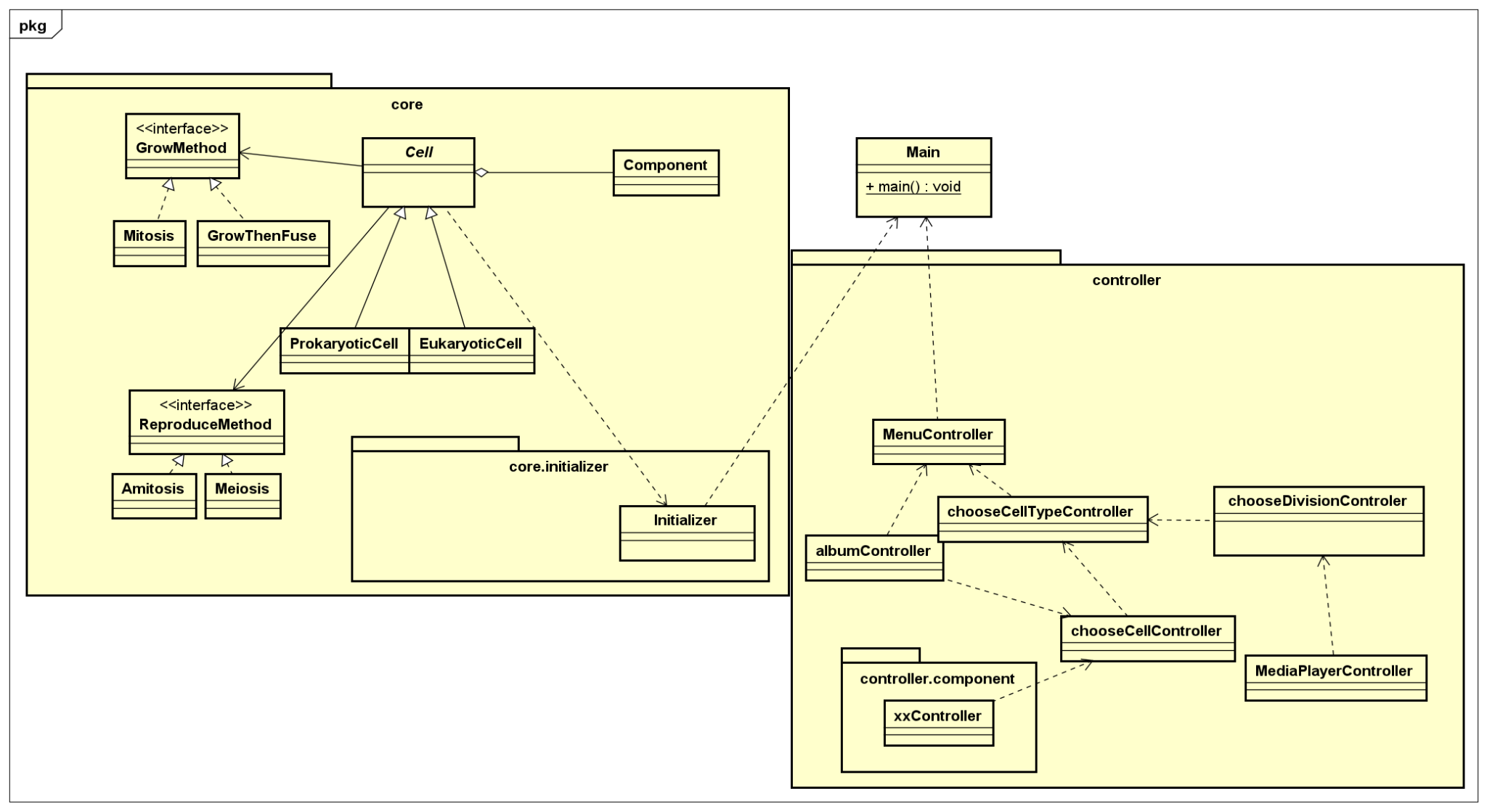
Here, ***Cell*** is our center of study. A cell can be classified as Prokaryotic or Eukaryotic, and consists of many components. Since a cell exists with its components, they make an *aggregation*. In fact, cell and cell components make a *composition* relation, however, the concept of component here can be more generalized, so we chose an aggregation relation.

Also, a cell can divide to grow, therefore Cell class can associate with GrowMethod or ReproduceMethod. The name format x + Method represents the meaning of the method to the body.

Next, an Initializer helps initialize all data required for the app to launch properly.

For the ***controller*** package, the classes are to represent the stages of the app, corresponding to the use cases.

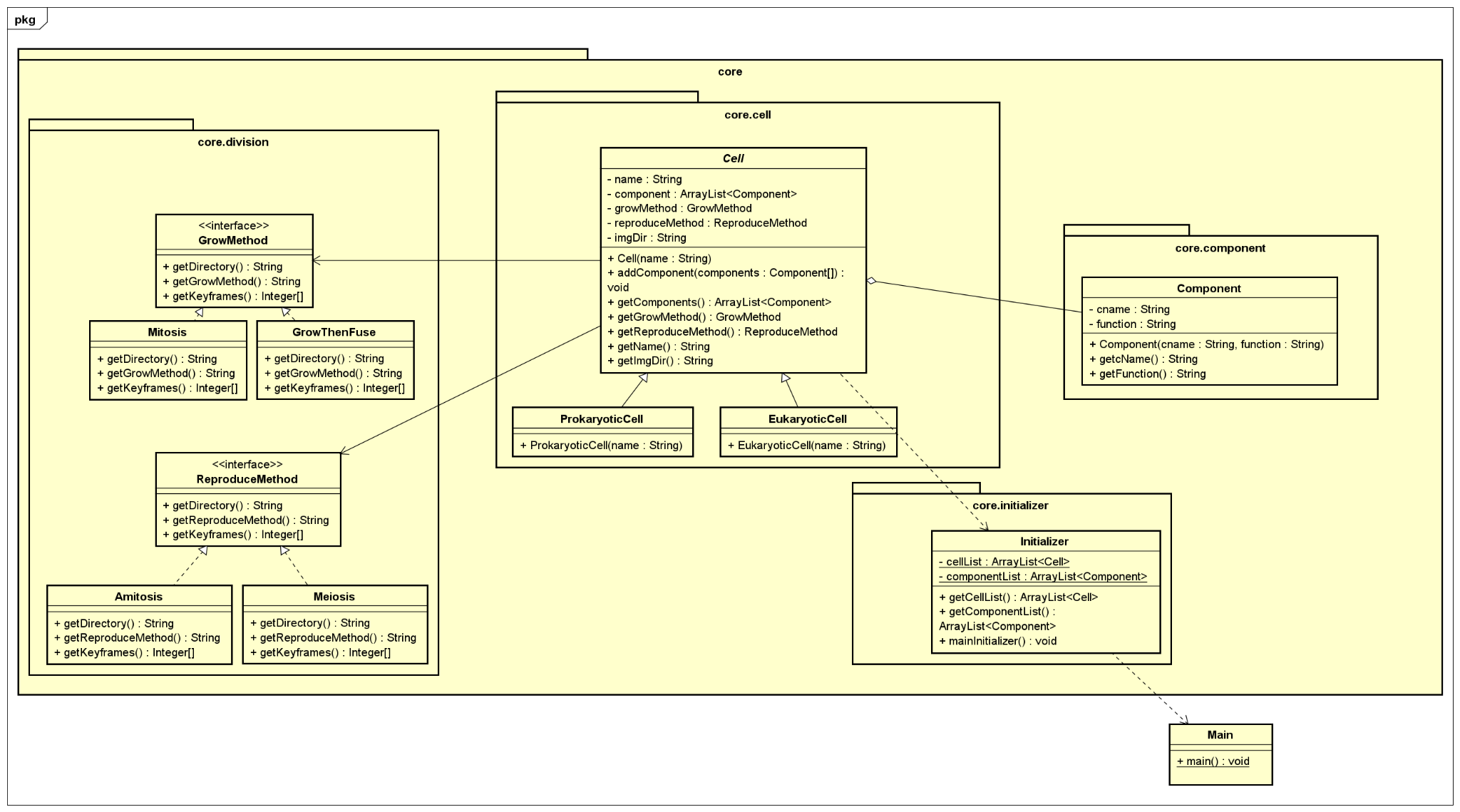
More details of our design will be discussed in section 4.2



***Figure 4.*** *General Class Diagram*

**4.2 PACKAGE DETAILS**

**A, Model**



***Figure 5.*** *Model (Core) class diagram*

To be more precise, for the Cells, we apply a **Strategy pattern**. A cell, when constructed using a specific constructor (ProkaryoticCell(String name) / EukaryoticCell(String name)), will automatically get its grow method, reproduce method and directory. While this still has some limitations (refer to section 5), it makes the coding process more convenient. When we scale up the project, which means the limitations have been solved, the only thing we need to modify is to add a grow method and a reproduce method to the constructor, instead of modifying a huge load of code.

Some necessary methods used in our model are present in the following table:

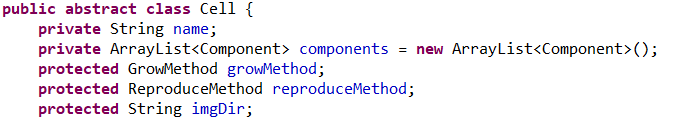
|  |  |  |
| --- | --- | --- |
| Class/Interfaces | Method/Attribute related | Operation, explanation |
| Cell | name, getName() | Name of cell (e.g. Animal cell, Bacteria,...) |
| addComponent() | From an array of components, add each component to the cell |
| ProkaryoticCell | constructor | Also add to an ArrayList<Cell> in class Initializer which acts as a storage |
| EukaryoticCell |
| Component | cName, function | Name, function of component |
| constructor | Also add to an ArrayList<Component> in class Initializer which acts as a storage |
| All class that implements the (x+Method) interfaces | getDirectory() | Returns video directory of the division method |
| get(x)Method() | Returns name of the method |
| getKeyframes() | Returns an Integer array containing the keyframe of the phases in the video |

The 4 OOP techniques are used in the model as follows:

**Encapsulation**

Major attributes are set private for protecting information from our accessor and ​​store the data members and data methods of a class together.

For example , private String name , private String imgDir, restrict the access to Cell class, but imgDir is protected, thus can be passed to children classes.



**Abstraction**

abstract class ***Cell*** 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.

**Inheritance**

Class ProkaryoticCell and Class EukaryoticCell inherit from class ***Cell***.

Interfaces are also used: GrowMethod ,ReproduceMethod

The main purpose is code reusability. The code that is present in the class Cell can be directly used by the child classes (Class ProkaryoticCell and Class EukaryoticCell)

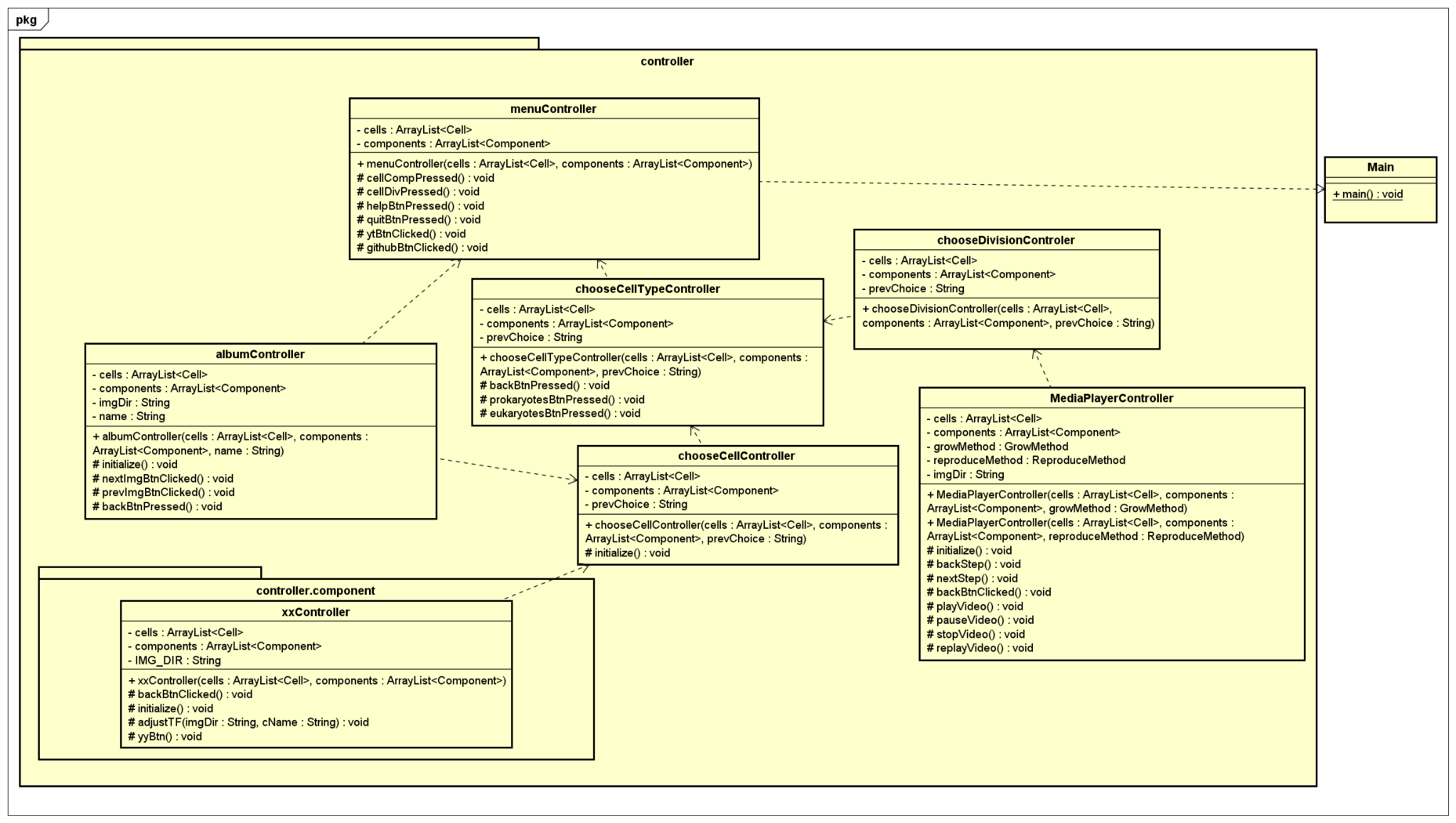
**Polymorphism**

When Amitosis and Meiosis implements reproduceMethod, override the methods getDirectory(),getReproduceMethod(), getKeyframes()

Similar things happen when GrowThenFuse and Mitosis implements GrowMethod

The main purpose is that polymorphism allows to define one interface and have multiple implementations, apply upcasting and downcasting flexibly in various usage all over the application.

**B, Controller**



***Figure 6.*** *Controller class diagram*

The following table explains all controller classes in our program:

|  |  |  |
| --- | --- | --- |
| Class/Interfaces | Method/Attribute related | Operation, explanation |
| All controller classes | cells, components and constructor | Two ArrayList created by Initializer passed through Main, and are passed through each other using constructor |
| backBtnPressed()/backBtnClicked() | Returns to previous window |
| MenuController | cellCompPressed() | Move to chooseCellTypeScreen with a selection of viewing cell components |
| cellDivPressed() | Move to chooseCellTypeScreen with a selection of viewing cell division methods |
| helpBtnPressed(), quitBtnPressed() | Performs help and quit use cases |
| ytbBtnClicked(), githubBtnClicked() | Redirect to online resources of our app |
| chooseCellTypeController |  | Choose between Prokaryotic Cell and Eukaryotic Cell and proceed using previous choice (view component or division) |
| chooseCellController  or  chooseDivisionController |  | After selecting type of cell, user need to select the specific object (to view its album and component); or select the type of cell division |
| albumController | nextBtnClicked()/prevBtnClicked() | Move to next/previous image in the album |
| xxController | Consist a lot of invisible button with their ActionEvent | Controller of the Cell component view, when the user clicks on the name of the component, they are actually clicking on one of the buttons. |
|  | adjustTF() | Make the the name and function of the component appears when clicked |
| MediaPlayerController |  | Methods and buttons used to serve the requirements: next phase, previous phase, play video, stop video, pause video or replay video |

# Limitations and further extensions

5.1 LIMITATIONS

* Some artificial errors on executing control flow leads to waste of time to fix bugs
* Lacking domain knowledge: for example, the kingdom of protista consists of many different kinds of cells; each kind of cell can reproduce in multiple ways; more division methods apart from Amitosis, Mitosis, Meiosis.
* Some excessive coding makes the program more complicated.
* Video size affects the size of the program.

5.2 FURTHER EXTENSIONS

* Broaden the knowledge space of the application: More cell types, more division methods, more images and videos.
* Make the application more interactive for students: Fun facts related to cells, Quizzes and Tests, Taking notes, etc.
* More settings for the application: Turn on music, Change languages, etc.

# References

1. Mokobi, F. (2022).Difference Between Plant Cell and Animal Cell. Retrieved 16 July 2022, from <https://microbenotes.com/plant-cell-vs-animal-cell/#animal-cell-vs-plant-cell-table-form>

### Anon(2022). Retrieved 16 July 2022, from <https://www.youtube.com/watch?v=VGcT1-XaWgk>

1. Protist summary. (2022). Retrieved 16 July 2022, from <https://www.britannica.com/summary/protist>
2. Molecular Expressions Cell Biology: Bacteria Cell Structure. (2022). Retrieved 16 July 2022, from <https://micro.magnet.fsu.edu/cells/bacteriacell.html>

### ANON(2022). Retrieved 16 July 2022, from <https://www.microscopemaster.com/paramecium.html>

1. Amyloplast - Wikipedia. (2022). Retrieved 16 July 2022, from <https://en.wikipedia.org/wiki/Amyloplast>
2. Lysosome - Wikipedia. (2022). Retrieved 16 July 2022, from <https://en.wikipedia.org/wiki/Lysosome>
3. Mitochondrion - Wikipedia. (2022). Retrieved 16 July 2022, from <https://en.wikipedia.org/wiki/Mitochondrion>

#### Cell nucleus - Wikipedia. (2022). Retrieved 16 July 2022, <https://en.wikipedia.org/wiki/Cell_nucleus>

#### Nucleolus - Wikipedia. (2012). Retrieved 16 July 2022, from <https://en.wikipedia.org/wiki/Nucleolus>

#### Model–view–controller - Wikipedia. (2022). Retrieved 16 July 2022, from <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

#### What Is Object-Oriented Programming? 4 Basic Concepts of OOP | Indeed.com. (2022). Retrieved 16 July 2022, from <https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming>

1. [Nguyen Le Anh Quan](https://viblo.asia/u/NguyenLeAnhQuan) (2018) Retrieved 16 July 2018, from <https://viblo.asia/p/design-pattern-strategy-pattern-924lJYwXZPM>