HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT
OBJECT-ORIENTED PROGRAMMING

Topic: Cell Division Demonstration

Class code: 131678

Lecturer: Nguyễn Thị Thu Trang

Group: 9

20200194	Nguyễn Huy Hải	hai.nh200194@sis.hust.edu.vn
20204909	Nguyễn Trung Hiếu	hieu.nt204909@sis.hust.edu.vn
20204877	Nguyễn Trung Hiếu	hieu.nt204877@sis.hust.edu.vn
20200212	Cù Duy Hiệp	hiep.cd200212@sis.hust.du.vn

Hanoi, 2022

CONTENTS

ABOUT OUR GROUP

Group 9 has 04 members, and the work was thus divided,

20200194 Nguyễn Huy Hải
Display cell component
Build screen
Build amitosis
Write report
Fix some parts of GUI
Build package Controller
Build Controller class diagram

20204909 Nguyễn Trung Hiếu
Prefix display cell component and design
Rebuild and design screen
Write report
Build slide
Fix some parts of GUI
Make demo video
Fix general class diagram
Fix use case diagram
Combine code

20204877 Nguyễn Trung Hiếu
Build Mitosis, Meiosis
Write report
Build package Controller
Build package Model
Build use case diagram
Build class diagram
Build model class diagram
Fix general class diagram
Combine code

20200212 Cù Duy Hiệp
Provide Content of cell component and division
Customize GUI
Build slide
Build Help menu

CHAPTER 1

Project Description

1.1 PROJECT REQUIREMENT

Topic Demonstrations of types of cell division

Basic knowledge Cell (prokaryotic or eukaryotic) and cell division (binary fission, mitosis, and meiosis).

Specifications These are as follows:

GUI: Design at own will.

Design: The application must have these functions,

- Main screen: Title of the application, options to choose the type of cell (prokaryotic or eukaryotic), help menu, and quit.
  - Help menu: Shows the basic usage and aim of the application.
  - Quit button: Exits the application. Ask for confirmation.
- For each type of cell, the user can choose to investigate one of the cell processes for demonstration.

In the demonstration:

- Display the cell components. Note that each component has different functions, you should display and explain them.
- Start button: Start demonstrating the cell division process through separate phases.
- On the bottom bar, the user can see the progress bar of the executing phase and choose to pause, continue, or go backward or forward a step in the execution.
- The user can also replay the process.
- Back button: Return to the main menu at any time.

1.2 USE CASE DIAGRAM

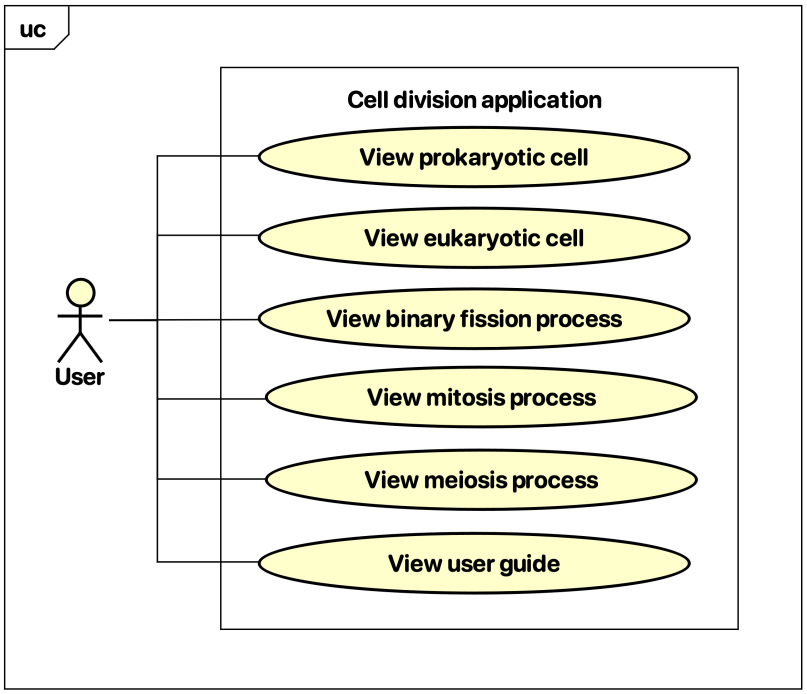

Figure 1.1 Use case diagram

Explanation The above figure shows the use case diagram for a cell division application. It contains a sole actor, namely, user, and six use cases. These use cases are: View prokaryotic cell, View eukaryotic cell, View binary fission process, View mitosis process, View meiosis process, and View user guide.

All of these six use cases are associated with the user.

User interaction with the program Upon opening the application, the user is welcomed with the Home screen. It contains the title of the program alongside with two buttons, Help and Quit on the top, and, in the center, two buttons titled “Prokaryotic cell” and “Eukaryotic cell”.

To view a particular type of cells, the user can choose either one of the two aforementioned center buttons. The user is then moved to a screen displaying a kind of cell, its components and buttons to display its related division process(es). If we choose a particular component, the screen will show the detail information of that part.

If the user choose to view the cell division processes, he is moved to a screen showing the process. The user can start, pause, continue, go backward or forward a step and watch the progress bar to know which stage of process is showing on screen.

CHAPTER 2

Design

2.1 GENERAL CLASS DIAGRAM

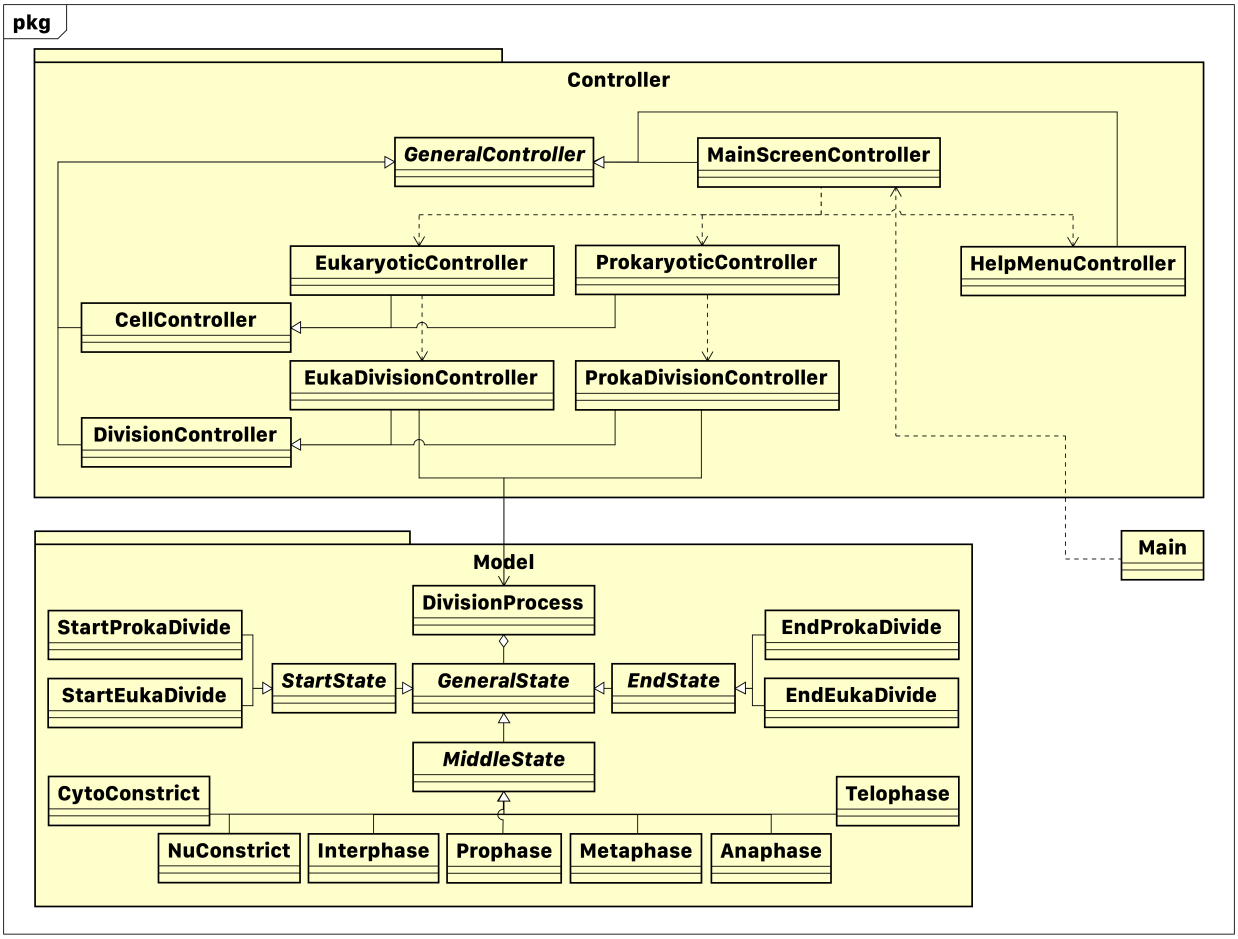

Figure 2.1 General class diagram

The `Main` class, depends on `MainScreenController`, is used to initialise the application.

Package `controller` contains `MainScreenController`, `CellController` (furtherly inherited by `ProkaryoticController` and `EukaryoticController`), `DivisionController` (inherited by `ProkaDivisionController` and `EukaDivisionController`) to control the GUI and action of each scene.

Package `model` contrains the abstract class `GeneralState`, which holds basic information of cell, its components, and its states throughout its life. In this package, we also find the class `DivisionProcess`, which aggregates on `GeneralState`. This class chronologically connects a state with its next/former state.

2.2 PACKAGE MODEL

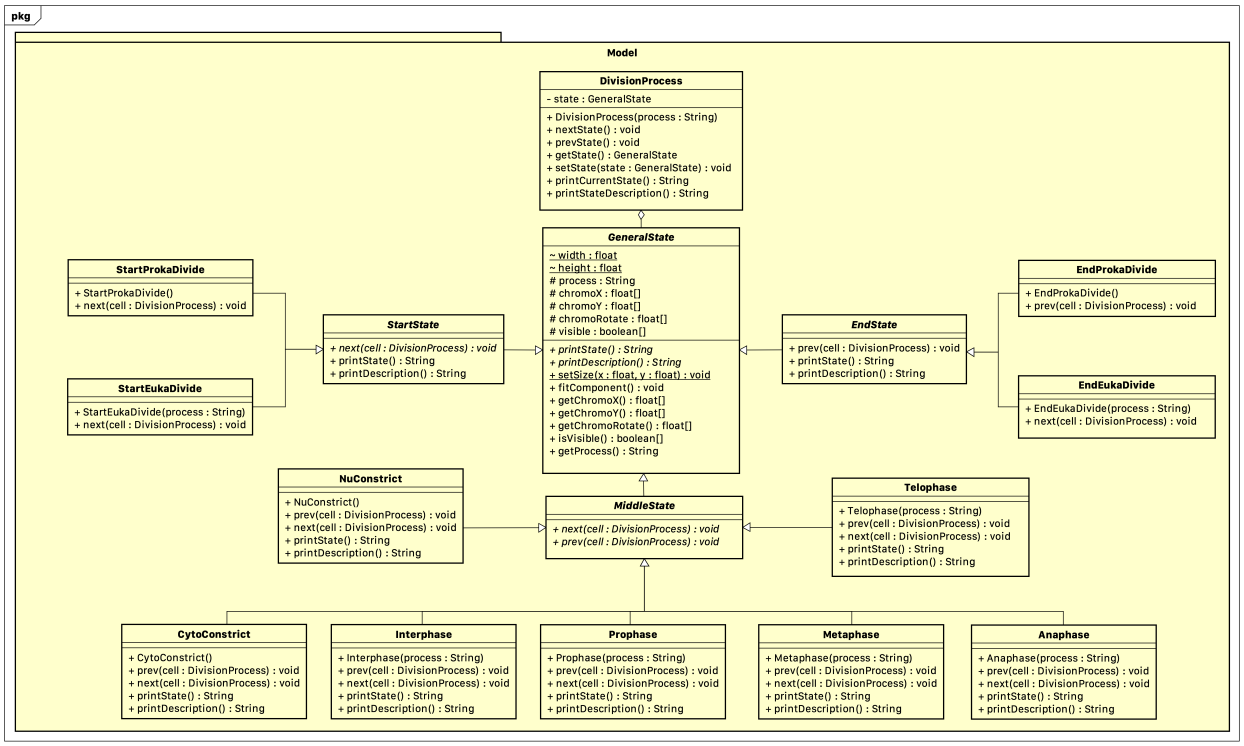

Figure 2.2 Class diagram for package model

2.2.1 Explanation

The core of the class `Model` is the `DivisionProcess`, which serves as a container to our cell division processes. It contains an instance of a `GeneralState` object, which represents the states of the division process.

2.2.2 GeneralState class

`GeneralState` is an abstract class, the parent of all states/phases in the package. `GeneralState` implement the following attributes and methods,

- `width`, `height` are the size of the `anchorPane` that holds the component, used for resizing the component and coordinate when displayed. Their modifiers are `default static` for `package-private` purpose.

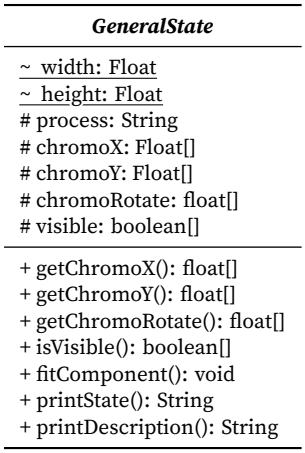

Figure 2.3 GeneralState class

- `process` is the name of the cell division process of the state. In this process we consider 3 cell division processes: binary fission, mitosis, meiosis. Note that we only use this for mitosis and meiosis because the same states in mitosis and meiosis behave differently.

- `chromoX`, `chromoY`, `chromoRotate` is the coordinate information of chromosomes (X coordinate, Y coordinate, rotation angle).

- `visible` is the list of booleans that represents the visibility of some component of the cells.

- `getChromoX()`, `getChromoY()`, `getChromoRotate()`, `isVisible()` is the getter of the attributes.

- `printState()` and `printDescription()` are abstract methods which print the current state and the status of that state in the process.

- `fitComponent()`: A method to fit the chromosomes coordinate with the size of the pane.

2.2.3 DivisionState class

Consider the class `DivisionState`,

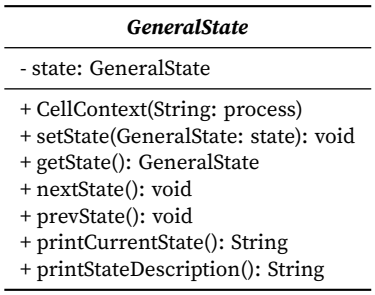

Figure 2.4 DivisionState class

`DivisionProcess` is the container of the state, where it stores an instance of `GeneralState`, and act based on that state. `setState()` and `getState()` are the getter and setter of the state, `nextState()` and `prevState()` call the next and previous state of the current state. `printCurrentState()`, `printStateDescription()` return the information of the current state.

The remaining classes are implemented as shown in Figure 2.2. Note that all data about chromosome coordinates and visibility of objects are defined in each class constructor, so each state data is independent of each other.

2.3 PACKAGE CONTROLLER

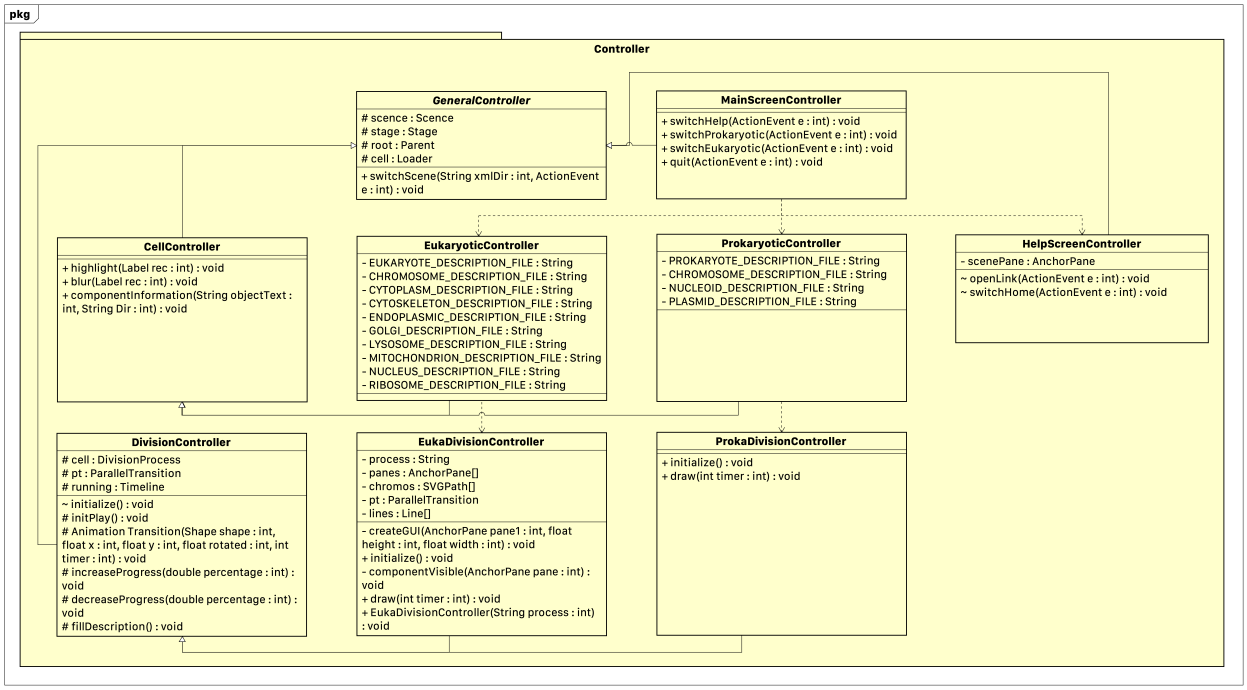

Figure 2.5 Class diagram for package controller

2.3.1 Explanation

Our packages contain controllers for multiple fxml: `Main-Screen`, `Prokaryotic`, `Eukaryotic`, `ProkaryoticDivision` and `Euka-ryoticDivision`. They all inherit from a base controller named `General-Controller`.

2.3.2 GeneralController class

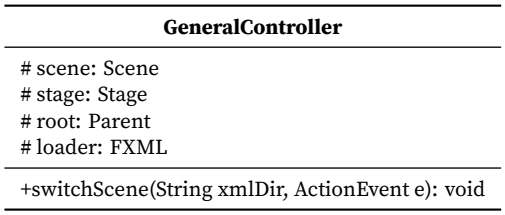

Figure 2.6 GeneralController class

`GenaralController` helps us to transfer different screens with their xml by a method, that is `switchScene()`.

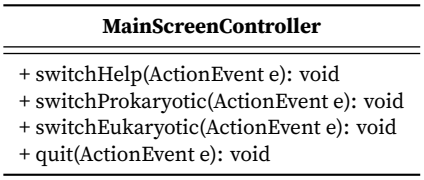

Figure 2.7 GeneralController class

2.3.3 MainScreenController class

`MainScreenController` is a child of `GeneralController`, it has 3 methods: `switchHelp()`, `switchProkaryotic()`, `switchEukaryotic()` which inherit the method `switchScene()` to switch to screen Help, Prokaryotic or Eukaryotic. `quit()` is another method in `MainScreenController` that helps to raise alerts when users try to click button quit or close the application.

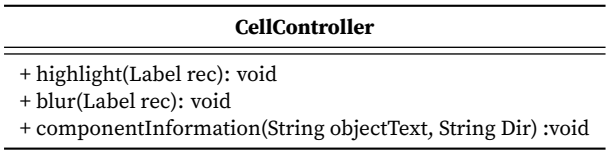

Figure 2.8 CellController class

2.3.4 CellController class

`CellController` help us to display detailed components of the cell, it is another child of `GeneralController` with 3 methods which are:

- `highlight()`: highlights the components when the mouse enters the area.
- `blur()`: blurs the components' name when the mouse exits the area.
- `componentInformation()`: show the detailed information of the organelles of the cell with different text files path leading to.

2.3.5 ProkaryoticController class

ProkaryoticController
- PROKARYOTE_DESCRIPTION_FILE: String - CHROMOSOME_DESCRIPTION_FILE: String - NUCLEOID_DESCRIPTION_FILE: String - PLASMID_DESCRIPTION_FILE: String

Figure 2.9 ProkaryoticController class

`ProkaryoticController`, extending from the class `CellController`, helps us to control the screen Prokaryotic. It inherits all the method `highlight()`, `blur()` and `componentInformation()` to display detailed information of `Prokaryotic()` cells. Also it has the method `switch()` to transfer different screens.

- `myImage`: the Image of the Prokaryotic cell.
- `blur()`: blurs the components' name when the mouse exits the area.
- `recChro`, `recNu`, `recPlas`: the label of specific components of the cell.
- All parameter with `FILE`: The path of the text file that contain the detailed information of each components

2.3.6 EukaryoticController class

Similar to `ProkaryoticController`, `EukaryoticController` helps us to control the screen Eukaryotic using inherited methods from `CellController`.

2.3.7 DivisionController class

`DivisionController` is the base class for controlling animation of the division process. It is an abstract class with some methods that both prokaryotic division and eukaryotic division can use.

`DivisionController` implements some important attributes: the `objectField` contains name of the state, `descriptionField` contains description of the state. There are also play, pause, next, prev, replay buttons, and progress bar.

In this abstract class, we implement 2 abstract methods: `next()` and `initialize()`, which are must have for any child controller. Others methods of this class:

- `initPlay()`: initialize the running sequence of states by defining a TimeLine that triggers the next method after each 5 seconds.
- `Transition()`: create translation and rotation animation for objects.
- `playPressed()`, `stopPressed()`, `restartClicked()`: control the stop, play, restart buttons' behavior.

- `fillDescription()`: display name and details of a state in cell division process

2.3.8 ProkaDivisonController class

Two class inherit from the class `DivisionController`: `ProkaDivisonController` and `EukaDivisionController`.

`ProkaDivsionController`: control the amitosis process of prokaryotic cells:

Attributes: implement the cell and chromosomes graphic with shape in `javafx`. The membrane we use ellipse to visualize and the chromosomes we use `SVGPath`.

Methods: `next()` and `prev()` moves to the next or previous state. `draw()` is used for creating animation to move the component of the cells.

2.3.9 EukaDivisonController class

`EukaDivisionController` is more complex because it controls two process, mitosis and meiosis:

Attributes: List of panes and a `GridPane` to hold the cell components, `SVGPath` to represent the chromosomes, and lines to represent the spinning fibers.

Methods:

- `next()` and `prev()` moves to the next or previous state. Note that these two are more complex than the one in prokaryotic because it needs to handle the cell splitting when moving next the telophase. Even in telophase, we need to consider splitting strategy based on whether we are on phase 1 or phase 2 of meiosis.
- `createGUI` takes a pane and size as input and draws components on that pane with respect to its size. It draws the outer wall, the membranes, the chromosomes and the centroids.
- `initialize()` create the initial state of the meiosis or mitosis division process.
- `componentVisible`: update the visibility of the components based on the visible information of the DivisionProcess state.
- `draw()`: runs animation based on the coordinate information of `DivisionProcess` state. It also runs the animation of spinning fibers. Note that spinning fibers only visible in two state (telophase and anaphase), it is different than other animations (require used of a Timeline instead of Transition), and it's different in 3 scenarios: mitosis, first phase of meiosis and second phase of meiosis. Because of that, in order to handle it effectively, we use logic `if-else` directly in the controller instead of model for easy implementation.

LIST OF FIGURES