Object - oriented Programming

Lecturer: Prof. Dr. Tran The Hung

Class ID: 147839

Interactive simulation of composition of forces

Members:
Hoang Khai Manh -20225984
Truong Linh Duyen - 20225968
Nguyen Thi Thu Huyen -202260
Hoang Quoc Hung -20226043

Contents

1 Introduction

In this project, we will explore Newton's laws of motion by examining the behavior of two objects: the cube and the cylinder. These objects have distinct properties and respond differently to external forces.

The cube is a three-dimensional object with equal side lengths. When a force is applied to a cube, it affects its center of mass. The direction and magnitude of the force determine the cube's resulting motion. Friction also plays a role in its movement, with the coefficient of friction determining the resistance to sliding.

The cylinder, on the other hand, is characterized by a circular cross-section and height. When a force is applied to a cylinder, it affects its center of mass. Similar to the cube, the direction and magnitude of the force determine the resulting motion. Friction between the cylinder and the surface affects its sliding motion.

To implement this simulation, we will employ object-oriented programming (OOP) techniques. OOP allows us to encapsulate the properties and behaviors of the cube and cylinder within separate classes, ensuring their internal workings are hidden and can only be accessed through well-defined interfaces. We will also utilize abstraction to extract common characteristics and behaviors into base classes, making code reuse and shared functionality implementation easier.

Polymorphism will enable us to handle different object types (cube and cylinder) through a common interface, facilitating uniform interaction regardless of their specific implementations. Inheritance will be used to create derived classes for the cube and cylinder, inheriting shared characteristics and behaviors from base classes, organizing code, and allowing object-specific customization.

By employing OOP techniques, we can develop a modular and adaptable simulation that accurately represents the interaction between forces and objects according to Newton's laws of motion. This project aims to deepen our understanding of physics principles and showcase the practical applications of object-oriented programming.

Get ready to dive into the fascinating world of motion and object interaction!

2 Assignment of member

Hoang Khai Manh (Leader)	Hoang Quoc Hung	Nguyen Thi Thu Huyen	Truong Linh Duyen
20225984	20226043	20220073	20225968
Slide Screen Package Screen Diagram Logic + Prototype	Slide Screen Package Utils Package Video Demo	Slide Screen Package General diagram UseCase diagram	Slide Object Package Object diagram Utils Package
MainScreen.java AppliedForce.java ControlPanel.java MainCharacter.java	ArrowPanel.java MovingImagePanel.java Characters.java ExceptionCase.java	FrictionCoefficient.java MenuofParameters.java ShowParameters.java README.md	Circle.java Square.java Objectss.java ValueInput.java

3 Mini project description

3.1 Project overview

- Create a simple interactive simulation application to demonstrate Newton's laws of motion.
- Utilize Version Control (specifically GitHub) for effective collaboration and sharing among team members.
- Create use-case and class diagrams to guide the development process.
- Apply object-oriented programming concepts such as Inheritance, Polymorphism, Abstraction and Encapsulation.
- Clearly explain the project ideas and reasons behind their selection in the report and presentation.

3.2 Project requirement

As we mentioned before, we need to create an interactive simulation application to demonstrate Newton's laws of motion with specific requirements:

- The GUI should be the same as in the reference [1] [2], with objects, sky and surfaces. However, we will design the same graphical interface but will be more artistic.
- Users can control the main object, the surface, and an actor who applies a horizontal force on the main object's center of mass to observe its motion.
- Start the application by setting up the main object (cube or cylinder) on the surface and specifying relevant parameters (side length/mass for a cube, radius/mass for a cylinder).
- During simulation, users can control the actor's force by adjusting its length and direction.
- Users can modify static and kinetic friction coefficients of the surface.
- Display statistics related to forces, mass, velocity, acceleration, and position of the main object.
- Allow users to pause, continue, and reset the simulation.

- Recalculate the main object's statistics at each time interval based on the provided formula for physical force impact.
- Users can change the main object once chosen, and modifying parameters of the current object is initially allowed.
- If time permits, implement the ability to reset and choose a different main object.

3.3 Use case diagram and explanation

3.3.1 Use case diagram

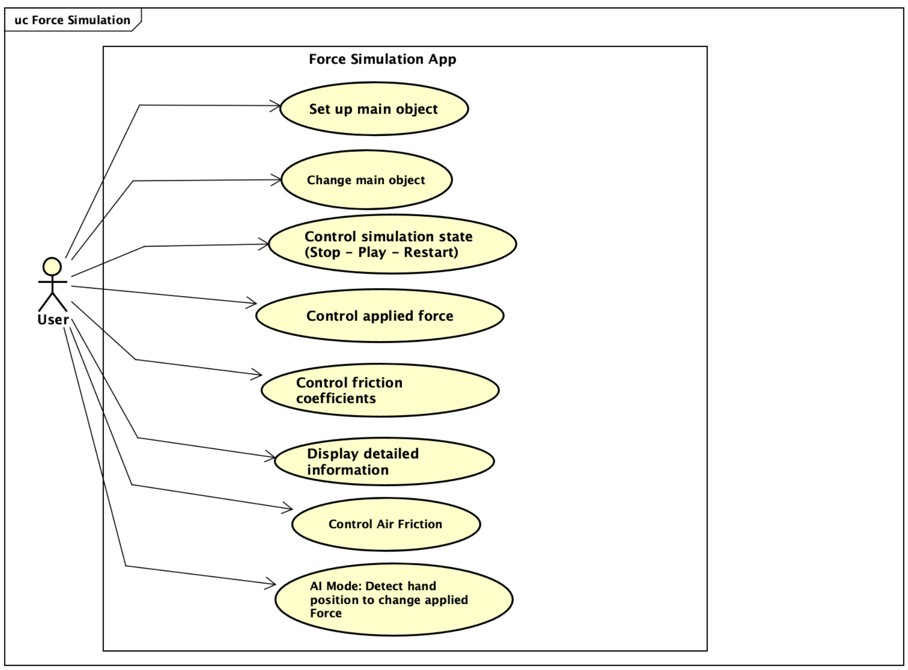

Figure 1: *Use case diagram.*

3.3.2 Explanation

- **Set up main object:** User chooses the type of the object (Square or Circle) by clicking on that object, and then passes the value for object's mass and size by filling in the corresponding blank.
- **Change main object:** User can change click another object on the bar and reset the object.
- **Control simulation state (Stop – Play – Restart):** - Users can modify the simulation state by interacting with the control buttons. The buttons available for controlling the simulation state include "Stop", "Play", and "Restart".

  - "Play" button: By pressing the "Play" button, the user can start or resume the simulation. This button initiates the simulation process or continues it if it was previously paused.

- "Stop" Button: The "Stop" button allows the user to pause the simulation. When the simulation is running, pressing this button will temporarily halt the progress and hold the current state.

- "Restart" Button: The "Restart" button is used to restart the project or simulation. When pressed, it resets the simulation to its initial state, clearing any previous progress or changes made during the simulation.

- **Control applied force:**

  - Users can modify the applied force by adjusting the slider.

  - The program will update the applied force on the object, resulting in changes to the net force, acceleration, and vector width.

  - Users will see the object move faster or slower, with the force vector width reflecting the changes. Labels for acceleration and force values will be updated.

- **Control friction coefficients:**

  - Just as with controlling the applied force, the user can adjust the coefficients of friction by adjusting the slider.

  - After the coefficients of friction have been modified, the program will update the statistics, such as changing the friction force and adjusting the width of the vector representing it.

  - The user will see that the object moves faster or slower, and the values displayed on the labels representing acceleration and force will change accordingly based on the current coefficients of friction.

- **Display detailed information:** - Users have the option to display detailed information about the forces and the object by selecting checkboxes. The checkboxes correspond to specific label attributes.

  - If a checkbox is selected, the program will display the values of the corresponding label attributes. This allows users to observe and track the detailed information related to the forces and the object.

  - Users can choose to show or hide the detailed information by selecting or deselecting the checkboxes respectively. This provides flexibility in displaying the desired level of detail in the simulation.

- **Control Air Friction:**

- **AI Mode: Detect hand position to change applied Force:**

4 Design

4.1 General class diagram

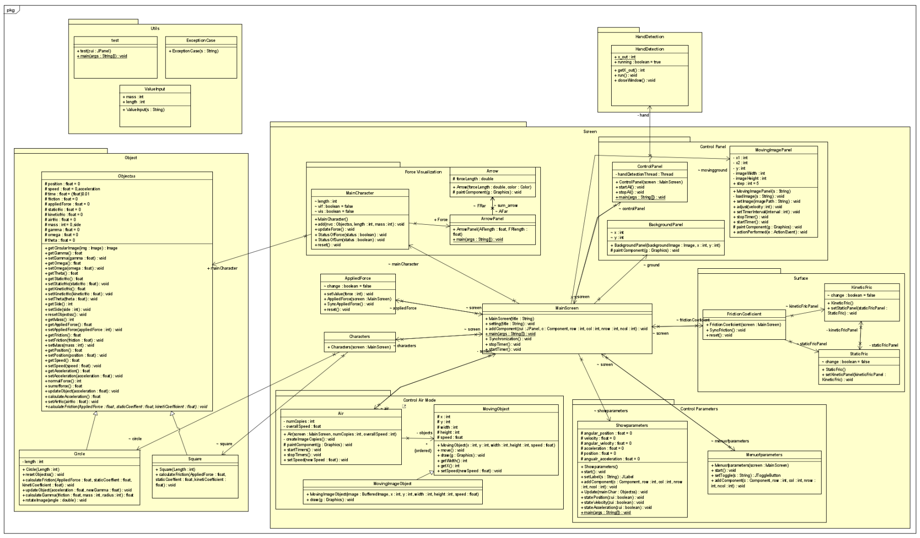

Figure 2: *General Class Diagram.*

Package Name	Class Name	Description
Utils	Utils	Contains utility methods for general functions.
	Exception Case	Handles exception cases.
	Value Input	Contains input values such as mass and length.
Object	Object	Represents a general object in the system with basic attributes and methods.
	Circle	Represents a circular object, inheriting from Object.
	Square	Represents a square object, inheriting from Object.
Force Visualization	Main Character	Represents the main character with attributes and methods for character control.
	Applied Force	Represents the force applied to objects.
	Arrow	Represents an arrow used to indicate the direction of force.

Screen	BackgroundPanel	Represents the background panel of the screen.
	ControlPanel	Represents the control panel of the system.
	MovingImagePanel	Represents the moving image panel.
	MainScreen	Represents the main screen of the system.
Surface	KineticF	Represents kinetic force.
	FrictionCoefficient	Represents the coefficient of friction.
	Surface	Represents a surface in the system.
HandDetection	HandDetection	Represents hand detection, with methods to open and close detection mode.
MovingObject	MovingObject	Represents a moving object with attributes and control methods.
	Air	Handles objects related to air.
	Showparameters	Represents display parameters.
	Menuparemeters	Represents menu parameters.

Table 1: *General class diagram information*

4.2 Detailed class diagram of each package

4.2.1 Screen Controller detail

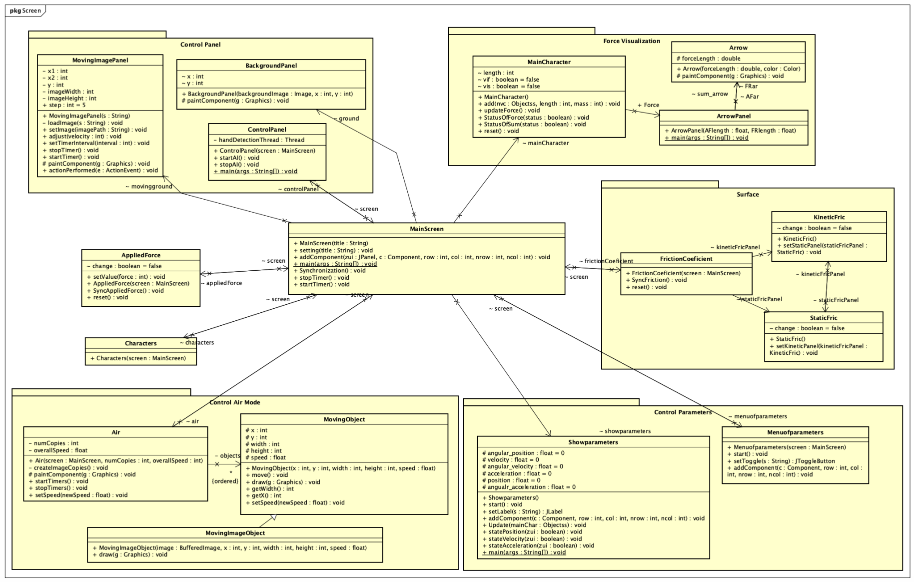

Figure 3: *Screen Controller detail*

1. MainScreen:

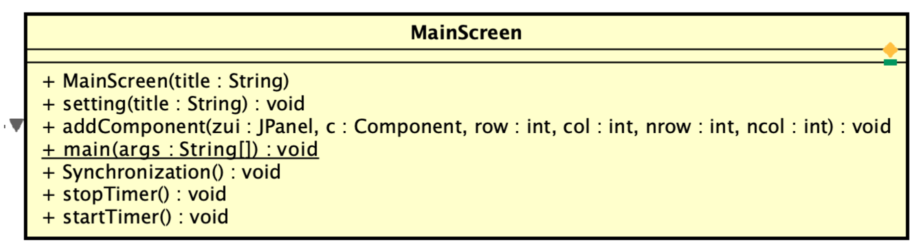

Figure 4: *MainScreen.*

- Methods:

+ MainScreen(title : String): Initialize the main screen of the application, setting up the layout and adding all necessary components.

+ setting(title : String) : Configure basic settings for the JFrame.

+ addComponent(zui : JPanel, c : Component, row : int, col : int, nrow : int, ncol : int) : Add a component to a JPanel using GridBagLayout with specified constraints.

+ main(args : String[]) : Entry point of the application.

+ Synchronization() : Synchronize the state of the main character and other components at regular intervals.

+ stopTimer() / startTimer(): Stop/Start the timer if it is running.

2. Control Panel:

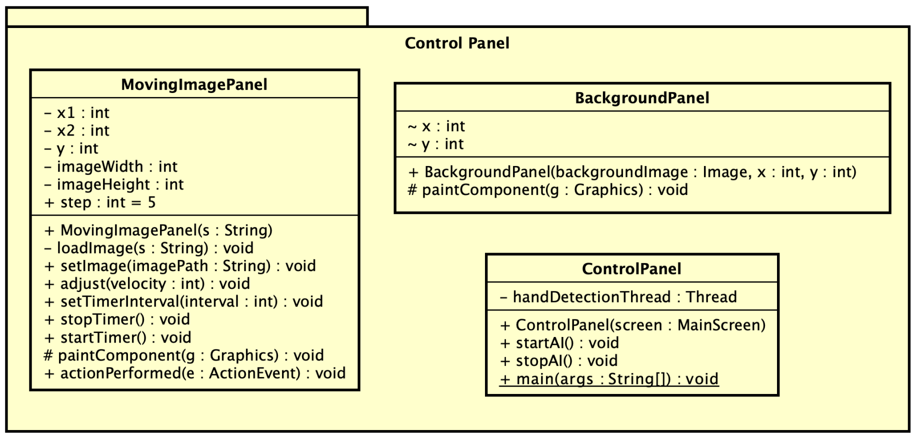

Figure 5: *Control Panel.*

Class name	Class usage	Attributes	Method
MovingImagePanel	Controls the ground	x1: int x2: int y: int imageWidth: int imageHeight: int step: int	adjustVelocity(int velocity) setTimerInterval(int interval) stopTimer() startTimer() paintComponent(Graphics g) actionPerformed(ActionEvent e)
BackgroundPanel	Creates a panel with an image on background	x: int y: int backgroundImage: Image	paintComponent(Graphics g)
ControlPanel	Controls the state of simulation and activates/deactivates some special modes	handDetectionThread: Thread	ControlPanel(MainScreen screen) startAI() stopAI()

Table 2: Information of Control Panel

a. MovingImagePanel:

- Attributes:

+ x_1, x_2 , y: Coordinates for drawing two copies of the image to create a continuous scrolling effect. x1 and x2 are the x-coordinates, and y is the y-coordinate.

+ imageWidth: Width of the image display on screen

+ imageHeight: Height of the image display on screen

+ step: Determines the movement step size of the image in each timer tick. A positive value moves the image to the right, and a negative value moves it to the left.

- Methods:

+ MovingImagePanel(s : String): Initializes the MovingImagePanel with an image and sets up the timer.

+ loadImage(String s): Loads the image from the given file path and sets initial positions for scrolling.

+ SetImage(imagePath : String): Changes the image to a new one and repaints the panel.

+ adjust(velocity : int): Adjusts the scrolling speed of the image based on the given velocity.

+ setTimerInterval(interval : int): Sets the delay interval of the timer.

+ stopTimer()/startTimer() : Stop/start the timer if it is running.

+ paintComponent(g : Graphics): Customizes the painting of the panel to draw the scrolling images.

+ actionPerformed(e : ActionEvent): andles the timer's action events to update the positions of the images.

b. BackgroundPanel:

- Attributes:

+ x, y: The required width and height of the image

- Methods:

+ BackgroundPanel(backgroundImage): Initializes the attributes of class

+ paintComponent(g : Graphics) : Customizes the image of the panel

c. ControlPanel:

- Attributes:

+ handDetectionThread: Manages the execution of the HandDetection runnable, which is responsible for detecting hand movements using the webcam. This thread runs the hand detection logic in parallel to the main application.

- Methods:

+ ControlPanel(screen : MainScreen): Constructor to initialize the control panel

+ startAI(): Starts the AI-based hand detection feature.

+ stopAI() : Stops the AI-based hand detection feature and cleans up resources.

3. Force Visualization:

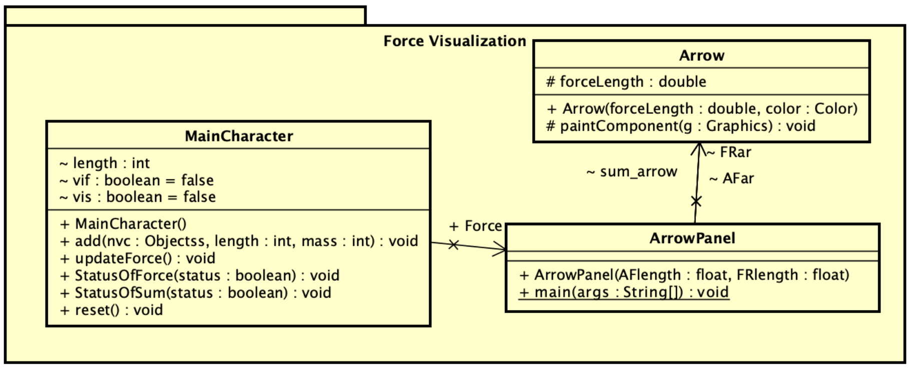

Figure 6: *Force Visualization.*

Class name	Class usage	Attributes	Method
Arrow	Controls the visibility of arrows of forces	forceLength : double ArrowforceLength : double color : Color	paint(Component(g : Graphics) : void)
MainCharacter	Controls the objects which is under consideration	length : int vf : boolean = false vis : boolean = true	MainCharacter() addForce(Objects, length : int, mass : int) : void updateForce() : void StatusOfForce(status : boolean) : void StatusOfRunstatus : boolean) : void reset() : void
ArrowPanel	Visualizes the magnitude of forces		ArrowPanel(Aflength : float, Fllength : float)

Table 3: Information of Force Visualization

a. MainCharacter:

- Attributes:

- length :Stores the length (or side length) of the main character object

- vif : Indicates the visibility status of the individual force arrows (applied force and friction force). When true, these arrows are visible on the screen.

- vis : Indicates the visibility status of the sum of forces arrow. When true, this arrow is visible on the screen.

- Methods:

+ MainCharacter(): Constructor to initialize the MainCharacter panel.

+ add(nvc : Objectss, length : int, mass : int) :Adds a new object object to the MainCharacter panel.

+ updateForce() : Updates the force arrows displayed on the MainCharacter panel based on the current forces acting on the main character.

+ StatusOfForce(status : boolean) : Sets the visibility of the individual force arrows (applied force and friction force).

+ StatusOfSum(status : boolean) : Sets the visibility of the sum of forces arrow.

+ reset() : Resets the force arrows displayed on the MainCharacter panel.

b. Arrow:

- Attributes:

+ forceLength : Represents the length of the force arrow. This value determines how long the arrow will be when it is drawn, and it is used to visually indicate the magnitude of the force.

- Methods:

+ Arrow(forceLength : double, color : Color): Constructor to initialize the Arrow object with a specified force length and color.

+ paintComponent(g : Graphics) : Overrides the paintComponent method to custom draw the arrow on the panel.

c. ArrowPanel:

+ ArrowPanel(AFlength : float, FRlength : float): Constructor to initialize the ArrowPanel with specified lengths for the applied force and friction force.

4. Applied Force:

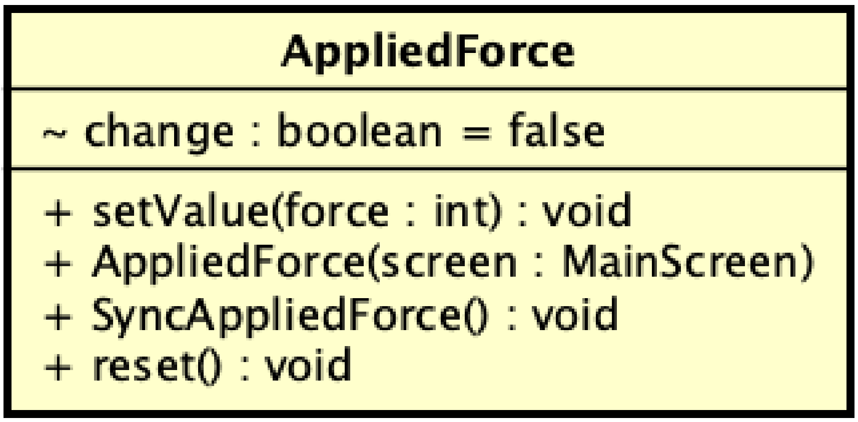

Figure 7: *Applied Force*

Class name	Class usage	Attributes	Method
Applied Force	Controls the applied force	change: boolean = false	setValue(force : int) : void AppliedForce(screen : MainScreen) SyncAppliedForce(): void reset(): void

Table 4: Information of Applied Force

- Attributes:

change :A flag indicating whether there has been a change in the applied force value (either through the slider or text field). It helps in synchronizing the applied force value with the main character.

- Methods:

+ setValue(force : int) : Sets the value of the slider to a specified force value within the range of -500 to 500.

+ AppliedForce(screen : MainScreen): Constructor that initializes the AppliedForce panel with a specified MainScreen object.

+ SyncAppliedForce() : Synchronizes the applied force value from the text field to the main character.

+ reset() : Resets the slider value to zero.

5. Characters:

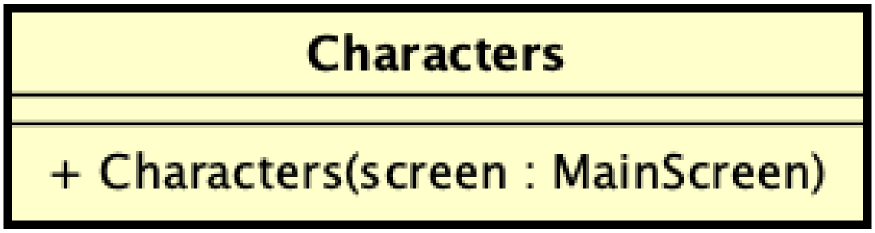

Figure 8: *Characters.*

Class name	Class usage	Attributes	Method
Characters	Represent a panel containing the cube-shaped and cylinder-shaped objects which can be initialized with desired parameters		Characters(screen : MainScreen)

Table 5: Information of Characters

- Methods: + Characters(screen : MainScreen) :Create a user interface component within a MainScreen that allows users to select and add either a square or a circle character with specified dimensions and mass

6. Surface:

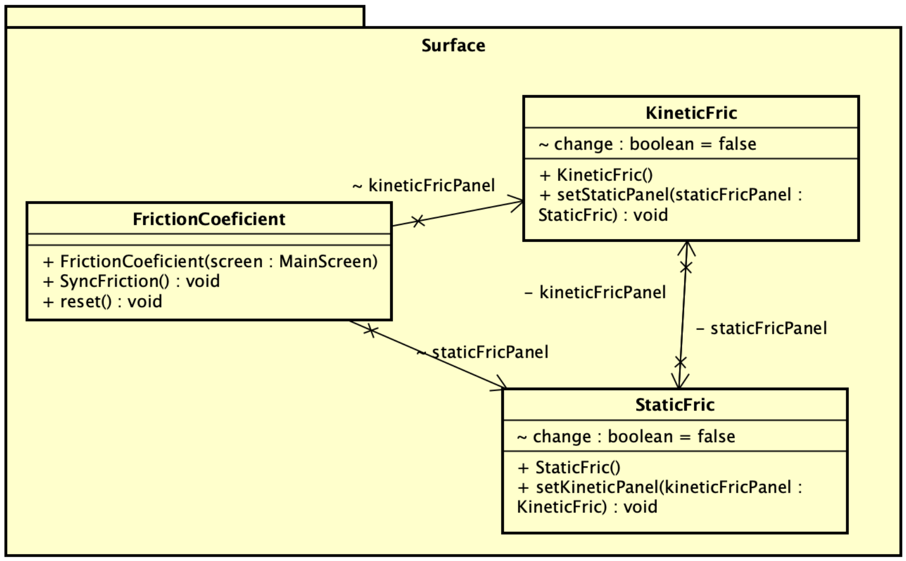

Figure 9: *Surface*

Class name	Class usage	Attributes	Method
FrictionCoefficient	Controls the friction coefficient		FrictionCoefficient(screen: MainScreen) SyncFriction() : void reset() : void
KineticFric	Controls the kinetic friction coefficient	change: boolean = false	KineticFric() setStaticPanel(staticFricPanel: StaticFric) : void
StaticFric	Controls the static friction coefficient	change: boolean = false	StaticFric() setKineticPanel(kineticFricPanel: KineticFric) : void

Table 6: Information of Friction Coefficient

a. Friction Coeficient

- Methods:

+ FrictionCoeficient(screen : MainScreen): Constructor that initializes the friction coefficient panel with a reference to the main screen. Sets up the layout to be a grid with two rows. Adds the staticFricPanel and kineticFricPanel to the panel. Synchronizes values between the static and kinetic friction panels.

+ SyncFriction() :Synchronizes the friction values between the static and kinetic panels. Updates the main character's static and kinetic friction values based on the inputs from the panels.

+ reset() : Resets the slider values in both the static and kinetic friction panels to their initial values.

b. Kinetic Fric:

- Attributes:

change : A boolean indicating whether the value has changed (change == true)

- Methods:

+ KineticFric(): Constructor that sets up the kinetic friction panel with a label, slider, and text field. Adds change listeners to the slider and text field to handle value changes and validation.

+ setStaticPanel(staticFricPanel : StaticFric) : Sets a reference to the staticFricPanel to ensure that the kinetic friction value is always less than the static friction value.

c. StaticFric:

- Attributes:

change : A boolean indicating whether the value has changed (change == true)

- Methods:

+ StaticFric(): Constructor that sets up the static friction panel with a label, slider, and text field. Adds change listeners to the slider and text field to handle value changes and validation.

+ setKineticPanel(kineticFricPanel : KineticFric) : Sets a reference to the kineticFricPanel to ensure that the static friction value is always greater than the kinetic friction value.

7. Control Air Mode:

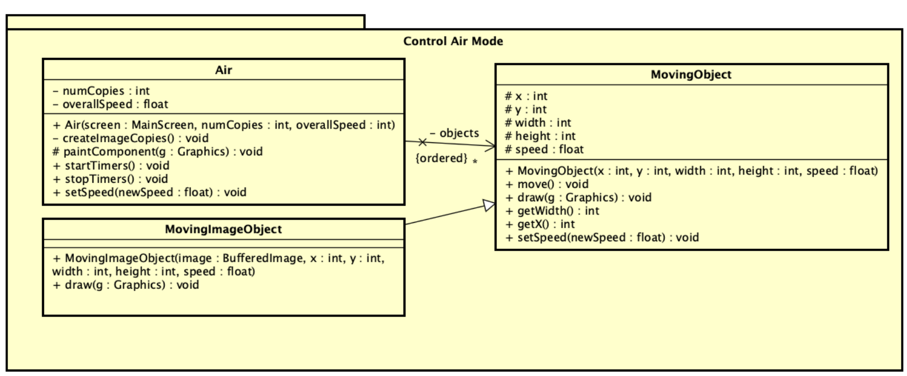

Figure 10: *Control Air Mode*

Class name	Class usage	Attributes	Method
Air	Simulates the air environment	numCopies: int overallSpeed: float	Air(screen: MainScreen, numCopies: int, overallSpeed: int) createImageCopies(): void paintComponent(g: Graphics): void startTimers(): void stopTimers(): void setSpeed(newSpeed: float): void
MovingObject	Creates the numerous copies of image and controls their's move	x: int y: int width: int height: int speed: float	MovingObject(x: int, y: int, width: int, height: int, speed: float) move(): void draw(g: Graphics): void getWidth(): int getX(): int setSpeed(newSpeed: float): void
MovingImage-Object	Creates the object of air image		MovingImageObject(image: BufferedImage, x: int, y: int, width: int, height: int, speed: float) draw(g: Graphics): void

Table 7: Information of Control Air Mode

a. Air:

- Attributes:

- numCopies : Number of copies of the image to create.

- overallSpeed : Reference to the speed of the object.

- Methods:

+ Air(screen : MainScreen, numCopies : int, overallSpeed : int): Constructor initializes the Air panel with a specified number of image copies and overall speed. Sets up timers, initializes the objects list, and sets initial properties.

+ createImageCopies() : Loads an image from a URL and creates numCopies of it. Randomly places copies on the panel with an adjusted speed based on overallSpeed.

+ paintComponent(g : Graphics) : Overrides JPanel's paintComponent method to draw all objects (MovingObject) onto the panel.

+ startTimers() : Starts two timers: timerCreate(Creates new image copies periodically.), timerRun(Moves existing objects and updates their positions). Also adjusts the air friction for the main character based on its speed.

+ stopTimers() : Stops and resets both timers. Resets air friction for the main character.

+ setSpeed(newSpeed : float) : Updates overallSpeed and adjusts the speed of existing objects accordingly.

b. Moving Image Object:

- Methods:

+ MovingImageObject(image : BufferedImage, x : int, y : int, width : int, height : int, speed : float): Constructor initializes the image object with specific attributes, including the image to be drawn.

+ draw(g : Graphics) : Overrides MovingObject's draw method to draw the image instead of a rectangle.

c. Moving Object:

- Attributes:

+ x, y: Position coordinates.

+ width ,height : Dimensions of the object.

+ speed : Speed of movement.

- Methods:

+ MovingObject(x : int, y : int, width : int, height : int, speed : float): Constructor initializes the object with position, dimensions, and speed.

+ move() : Updates the position of the object based on its speed

+ draw(g : Graphics) : Draws the object using graphics context g. For MovingObject, this draws a rectangle (default behavior).

+ getter and setter of some attributes.

8. Control Parameters

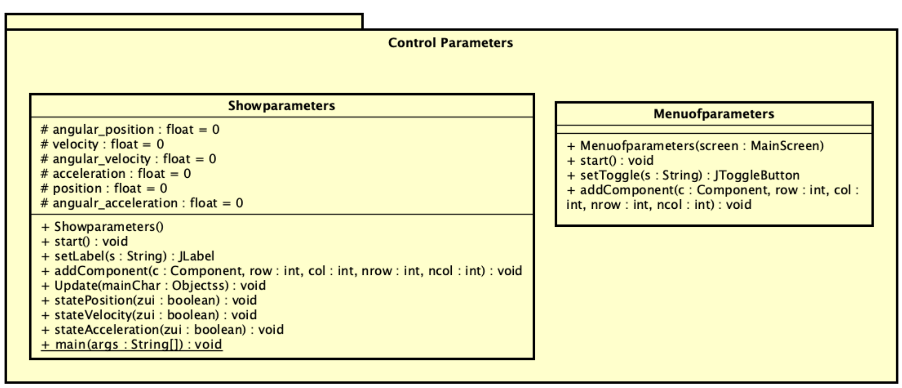

Figure 11: *Control Parameters*

Class name	Class usage	Attributes	Method
ShowParameters	Represents a panel containing the parameters of object's move	angular_position: float = 0 velocity: float = 0 angular_velocity: float = 0 acceleration: float = 0 position: float = 0 angular_acceleration: float = 0	ShowParameters() start(): void setLabels(): String[] JLabel addComponent(c: Component, row: int, col: int, nrow: int, ncol: int): void UpdateMainChar(objs: Objects): void statePosition(vis:boolean): void stateVelocity(vi: boolean): void stateAcceleration(vi: boolean): void main(args: String[]): void
Menuofparameters	Represents the buttons controlling the visibility of parameters		Menuofparameters(screen: MainScreen) start(): void setToggles(s: String[]): JToggleButton addComponent(c: Component, row: int, col: int, nrow: int, ncol: int): void

Table 8: Information of Control Parameters

a. Showparameters:

- Attributes:

+ angular position : Representing angular position.

+ velocity : Representing velocity.

+ angular velocity : Representing angular velocity.

+ acceleration : Representing acceleration.

+ position : Representing position.

+ angualr acceleration : Representing angular acceleration.

- Methods:

+ Showparameters(): Constructor that initializes the layout and components of the panel.

+ start() : Method to initialize and configure labels for parameters.

+ setLabel(s : String) : Creates and configures a JLabel with specified text.

+ addComponent(c : Component, row : int, col : int, nrow : int, ncol : int) : Adds a component to the panel with specified layout constraints.

+ Update(mainChar : Objectss) : Updates the displayed values based on the mainChar object provided.

+ statePosition(zui : boolean) : Controls visibility of position-related labels.

+ stateVelocity(zui : boolean) : Controls visibility of velocity-related labels.

+ stateAcceleration(zui : boolean) : Controls visibility of acceleration-related

b. Menuofparameters:

- Methods:

+ Menuofparameters(screen : MainScreen): Constructor that sets up the layout, initializes components, and sets listeners.

+ start() : Initializes and configures the toggle buttons (JToggleButton) for different parameters.

+ setToggle(s : String) : Creates and configures a JToggleButton with specified text.

+ addComponent(c : Component, row : int, col : int, nrow : int, ncol : int) : Adds a component to the panel with specified layout constraints.

4.3 Model

4.3.1 OOP technique

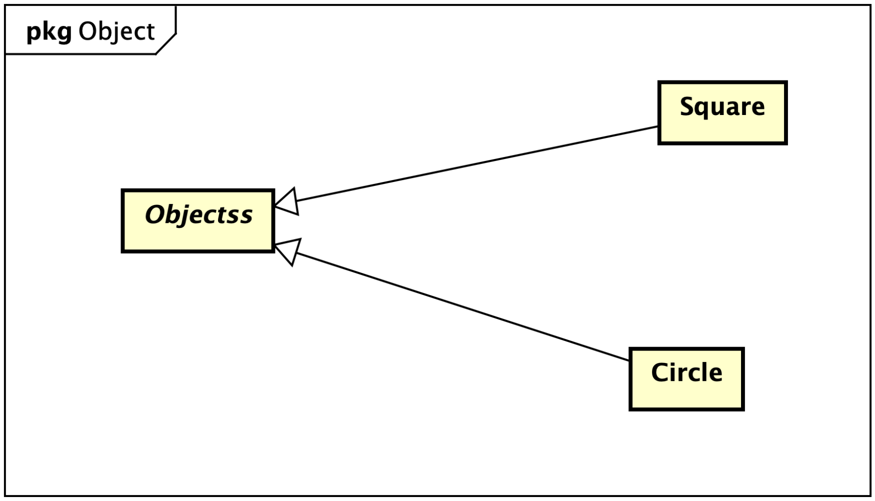

Figure 12: *Model diagram simple*

- **Encapsulation**
  - **Attributes (fields):** The attributes of the **'Objectss'** class are declared with **'protected'** access level, protecting them from direct access from outside the class (except by subclasses and classes in the same package).

```
protected float position = 0, speed = 0, acceleration ,
                time = (float) 0.01, friction = 0,
                appliedForce = 0, staticfric = 0,
                kineticfric = 0, airfric = 0;
protected int mass = 0, side;
protected float gamma = 0, omega = 0, theta = 0;
```

  - **Getter and Setter methods:** These methods allow controlled access and modification of the attribute values. This helps protect the data and ensures that changes follow the defined rules.
  - **Logic methods:** The class also contains methods to perform logical operations on the attributes.

- **Inheritance**

  In Java, we define a subclass using the keyword **"extends"**, and we used it in the Object package as follows:

```
public class Circle extends Objectss {...}
public class Square extends Objectss{...}
```

The superclass **'Objectss'** defines common attributes and behaviors for objects in the simulation. The subclasses **'Circle'** and **'Square'** inherit from Objectss and extend its functionality to suit their specific characteristics. This eliminates redundancy by defining shared properties like mass, position, velocity, and acceleration in the superclass. The subclasses override methods like **calculateFriction()** to customize their behavior based on their specific characteristics. Inheritance promotes code reuse and modularity, allowing for more efficient and organized code.

- **Abstraction**

  The Objectss class being declared as an abstract class demonstrates abstraction. This allows defining abstract methods that subclasses need to implement. The calculateFriction method serves as an example of an abstract method, where subclasses must provide a specific implementation. Using abstraction helps define a common interface and provides extensibility for subclasses.

  ```
  public abstract class Objectss extends JButton {...}
  ...
  public abstract void calculateFriction(float AppliedForce,
            float staticCoeffient, float kinetiCoefficient);
  ```

- **Polymorphism**

  Polymorphism allows objects of different classes to be treated interchangeably through a shared interface. (“Polymorphism in OOPs- Logicmojo”). In the provided code, the calculateFriction method implemented in the Circle and Square subclasses exemplifies polymorphism. Despite having the same method name, each subclass provides its own implementation, tailored to its specific characteristics and requirements. This flexibility enables objects of different classes to be used interchangeably within the context of calculateFriction, promoting code reusability and adaptability.

4.3.2 Object

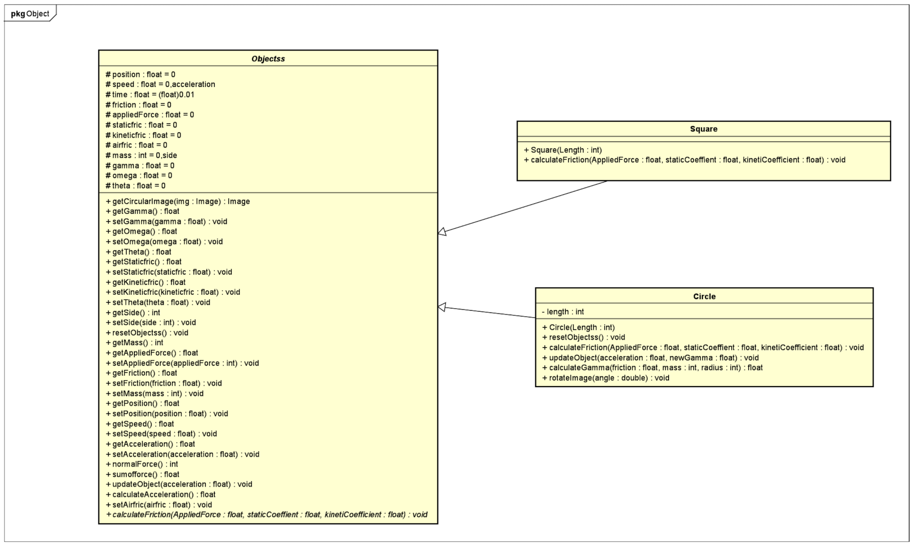

Figure 13: *Object diagram detail*

Class name	Clase usage	Attributes	Method	OPP technique
Objectss	Base abstract class for simulation objects	- side - mass - position - speed - acceleration - step	- getter & setter for attributes - normalForce() - sumofforce() - updateObject() - calculateFriction()	- Inheritance - Encapsulation - Abstraction - Polymorphism
Square	Represents a cube-shaped object in the simulation	Inherits attributes from Objectss	calculateFriction()	- Inheritance - Polymorphism

Circle	Represents a cylinder-shaped object in the simulation	Inherits attributes from Objectss - gamma - theta -omega	- getter & setter for attributes - resetObjectss() - calculateFriction() -updateObject() - calculateGamma() - rotateImage()	- Inheritance - Encapsulation - Polymorphism

Table 9: *Object detail*

- **Objectss class:**
  - It is an abstract class that serves as the superclass for Square and Circle.
  - **Attributes:**
    - position
    - speed
    - acceleration
    - time
    - friction
    - appliedForce
    - staticfric
    - kineticfric
    - airfric
    - mass
    - side
    - gamma
    - omega
    - theta
  - **Method:**
    - It provides methods to get and set the values of above attributes.
    - normalForce(): calculate the normal force of object
    - sumofforce():calculates the total force acting on the object, equal to the sum of the applied force and the friction force.
    - updateObject(float acceleration): calculate the acceleration, speed, and position of the main object based on the acceleration.
    - calculateFriction(float AppliedForce, float staticCoefficient, float kinetiCoefficient): calculate the friction applying on object based on the applied force and friction coefficients

- **Square class:**
  - **Attributes:**
    Inherits attributes from Objectss.

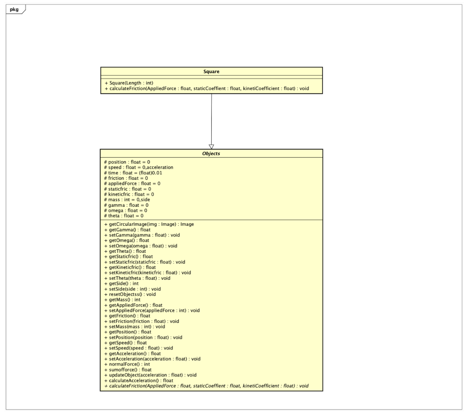

Figure 14: *Square diagram detail.*

- **Method:**

  - It extends the Objectss class and represents a cube-shaped object.

  - The *'Square(int Length)'* constructor is used to initialize a square-shaped object in a simulation, inheriting from the superclass Objectss.

  - It overrides the *'calculateFriction(float AppliedForce, float staticCoeffient, float kinetiCoefficient)'* method to calculate the friction.

- **Circle class:**

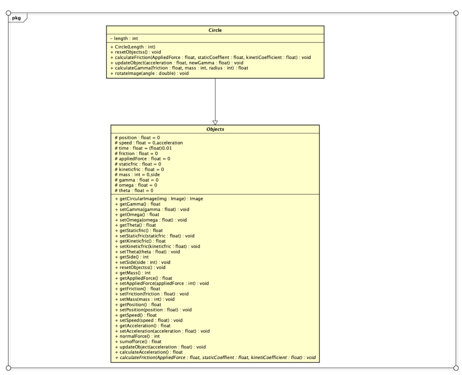

Figure 15: *Circle diagram detail.*

- **Attributes:**

- **Method:**

  - It extends the Objectss class and represents a cylinder-shaped object.

  - It introduces additional properties such as gamma, theta and omega.

- The *'Circle(int Length)'* constructor is responsible for initializing a circular-shaped object within a simulation. - The *'resetObjectss()'* reset properties: gamma, theta and omega to default values ('0').

- It overrides the *'calculateFriction(float AppliedForce, float staticCoeffient, float kinetiCoefficient)'* method to calculate the friction.

- The *'updateObject(float acceleration, float newGamma)'* method is used to update the object's state based on the provided acceleration and rotational speed (*'newGamma'*).
- The *'calculateGamma(float friction, int mass, int radius)'* method calculates the angular acceleration of the cylinder.

- The *'rotateImage(double angle)'* method calculates the angular attributes of the cylinder based on the applied force, gamma and updates them.

4.4 Other classes

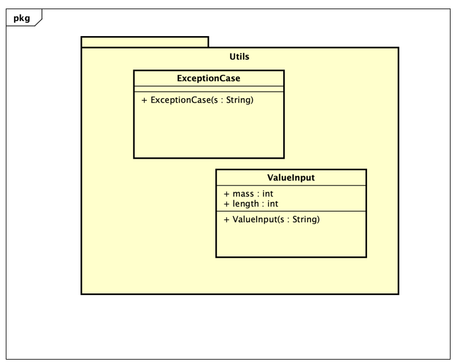

Figure 16: *Other classes.*

- **ExceptionCase class:**

  The 'ExceptionCase()' class is designed to create a simple error message window that displays a specified error message when an exception occurs.

- **ValueInput class:**

  The 'ValueInput()' class facilitates user input of *'mass'* and *'length'* values through a graphical dialog interface. It ensures that the entered values meet specified criteria and provides feedback to the user accordingly. After valid input, it allows the main application to access the *'mass'* and *'length'* values for further processing.

5 AI model

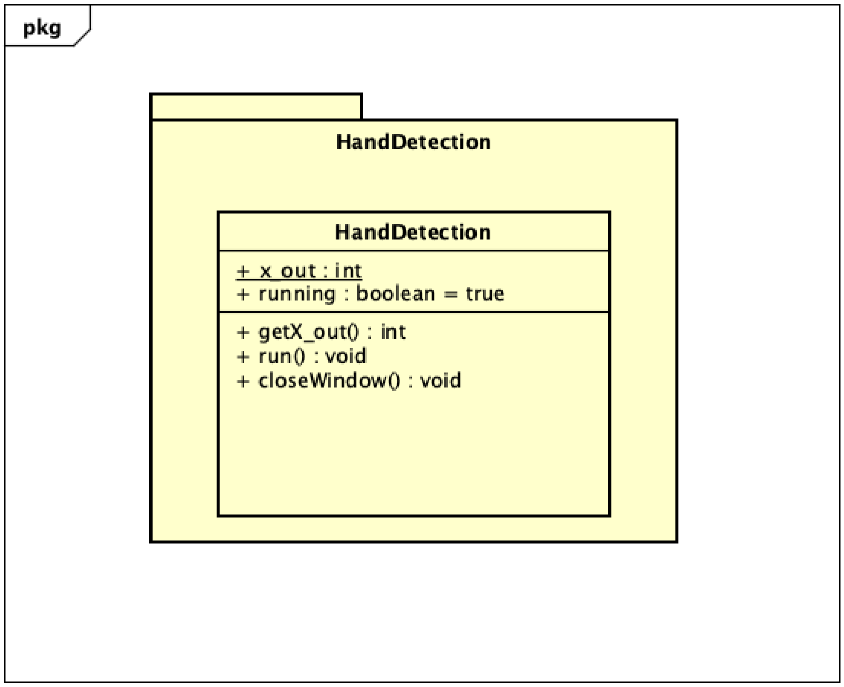

Figure 17: *AI model*

Attributes	**Method**
- x_out - running	- getX_out() - run() - closeWindow()

Table 10: *Hand detection detail.*

- This class implements the **'Runnable'** interface and performs hand detection in a video stream from a camera.
- **'x_out'**: static variable to store the x-coordinate of the detected hand.
- **'getX_out()'**: method to get the value of **'x_out'**.r
- The **'run()'** method is responsible for continuously capturing frames from the camera, detecting handa in each frame using OpenCV, and displaying the processed frames in a window named "Hand Detection". It runs in a loop until the **'running'** flag is set to **'false'**.
- The **'closeWindow'** method is responsible for terminating the camera capture and closing the display window used for showing the hand detection results.

‘HandDetection’ class is to perform real-time hand detection using a webcam feed with the help of OpenCV library.

List of Tables

List of Figures