Inertia Simulator

Amber Colletti
Brandon Conway
Aaron Setser
Matthew Shea
Yi-Chin Sun

Principles of Software Engineering
Vanderbilt University, Fall 2011

Table of Contents

Section I: Purpose

The inertia simulator is designed to teach 5th grade students about the properties of inertia by modeling how inertia affects objects of various size and shape inside a bus. The students will be able to choose what object to put in the bus (box or ball) and change the weight of the object, as well as the speed and acceleration/deceleration of the bus.

Section II: Requirements

- Allow user to create and drag objects across screen
- Allow user to define relationships between objects
- Allow user to start and stop simulation
- Allow user to delete objects/relationships
- Allow user to define attributes of all object(s)
- We will implement a simulation based on the user's input
- We will create useful error messages and useful hints (perhaps an XML to send to research group)
- Saving project
- Opening existing project

Section III: Design Approach

We began our approach by creating several use cases that demonstrated high level scenarios, such as creating a simulation or loading in an existing simulation. From these use cases, we built our first draft of a class diagram. In this diagram, we outlined all the major classes, such as our Graphical User Interface, Physics Engine, and Managers. We chose the class diagram as it is a classic diagram that allows us to easily visualize how all of our pieces will come together in the final product.

From this class diagram, we created all other diagrams utilized in our project. First we designed several sequence diagrams to cover almost any scenario a user would encounter. We chose the sequence diagrams because they are simple, yet incredibly useful when trying to visualize the flow of our project. In addition, looking at several of our sequence diagrams allowed us to find flaws with the class diagram that we were able to immediately fix. Once we fixed both the class diagram and sequence diagrams, we created our communication diagram. We found this useful in creating our project as it combines the benefits of a use case diagram and sequence diagram. Our graphical user interface benefited from this diagram, as it showed us what kind of inputs our user would need in order to properly invoke the methods involved with creating an object.

Section IV: Class Overviews

The following are general overviews of each class. The relationships between classes can be seen in the class diagram (Appendix III) and the classes can be reviewed in greater detail in the attached Javadoc Documentation (Appendix V).

edu.vu.vuse.cs278.g3.gui

Contains all classes that render the user interface. Interface files are designed using the interface editor in NetBeans and should be edited as in NetBeans.

MainWindow

Class that implements the main display window for the program. This window contains the NetLogo display window that will be modified by the physics engine.

EditObjectUI

Class that implements the edit object window. This window modifies the objects in the model and commits them to the NetLogo display.

SoftwareUI

Class that implements the create object window. This window is responsible for creating objects and properly adding them to the model.

edu.vu.vuse.cs278.g3.engine

Contains all classes that operate the physics engine. This packages consists of three files that are entirely responsible for running the physics simulation.

PhysicsEngine

Class that implements the physics engine. This class is actually a wrapper around an internally defined QueueExecutor that accepts Runnables and executes them on a single thread.

PhysicsFormulas

Class that contains flexible implementations of physics formulas used to calculate new positions and properties for objects. This class contains static methods that act as these formulas.

PhysicsActions

Class that contains higher level actions for running the physics engine. These actions are implemented as internal static Runnables for execution on the PhysicsEngine. Each Runnable should define one specific action.

edu.vu.vuse.cs278.g3.model

Contains all classes used to represent the physics data model. The two primary classes are the ObjectManager and RelationshipManager.

ObjectManager

Singleton class that handles the creation, editing, and deletion of all objects. Objects should only be edited through this class.

PhysicsObject

Base class that provides universal functionality for all objects involved in the physics engine. Subclasses should be sure to call the relevant PhysicsObject super functions if they choose to override them.

SquareObject

Class that extends the PhysicsObject to provide dimensions that make it a square, such as width and height.

RoundObject

Class that extends the PhysicsObject to provide dimensions that make it a circle, such as radius.

BusObject

Class that extends the PhysicsObject to provide dimensions that make it a circle, such as length.

RelationshipManager

Class that keeps a map of the relationships between objects. The map associates references to objects with a RelationshipType.

RelationshipTypes

Enumeration that is used to describe relationship types. The actual implementation is done in the PhysicsActions, but this is used to switch on to make the code cleaner

Section V: Operation

The program has three primary components: The graphical user interface, the physics engine, and the managers.

Graphical User Interface

The GUI accesses the physics engine and managers to control the objects and the simulation. The GUI also contains the Netlogo window which is used to display the simulation.

Physics Engine

The physics engine consists of a *QueueExecutor* and *Runnable* actions to execute on said executor. The *QueueExecutor* implements a thread that reads a queue of *Runnable*s and executes them sequentially. Each *Runnable* accesses the managers and performs some action on them, such as accelerate or move. These changes are then passed to the Netlogo window which updates the display.

Managers

The managers contain the Java representations of the Objects and Relationships. They provide basic functionality such as creating, editing and deleting objects while maintaining data integrity between the Java objects and the Netlogo objects that they are bound to.

For more details on how the individual parts work together, please see the class diagram in Appendix III.

Section VI: Walkthrough

Mixed Group Walkthrough (30 minutes)

Documents Examined: Original Class Diagram, All Sequence Diagrams

Fault #1: Physics Engine was not well developed and needed to be expanded to more properly detail the behavior.

Fault #2: User interface was not well developed and required additional dialogs to be developed.

Fault Rate: 4 faults/hour (both faults determined to be major faults)

Original Group Walkthrough (1 hour)

During the original group walkthrough, the only faults we found in our documents were minor faults, since we fixed the faults found in the mixed group walkthrough

Documents Examined: Updated Class Diagram, All Sequence Diagrams, and Communication Diagram

Fault #1: User Interfaces needed to be in three separate classes, as opposed to one in our last class diagram because three different windows were needed.

Fault #2: Ball, box, and bus attributes are similar (all ints); would be better to abstract the attributes for ball, box, and bus.

Fault #3: Sequence diagrams included classes that had been removed from the class diagram.

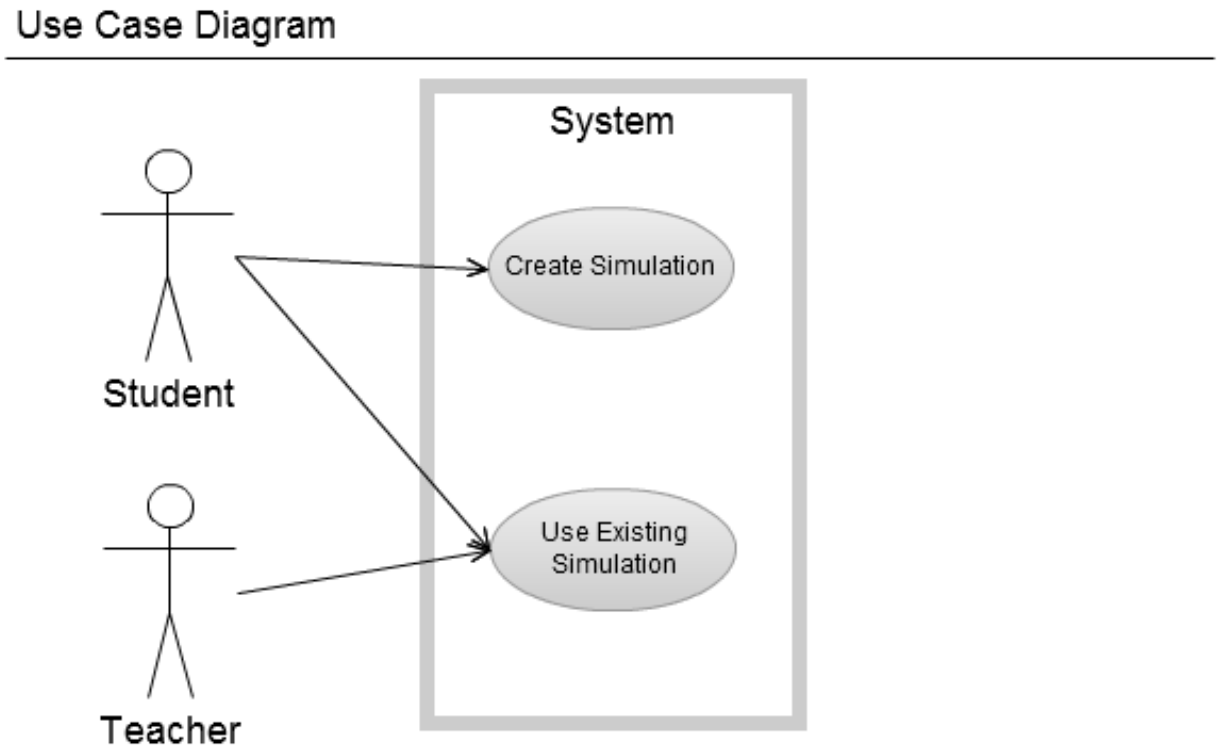
Fault #4: Ground object not necessary since it does not add anything to the simulation; makes more sense for it to be part of UI instead of a separate object

Fault Rate: 4 faults/hour

Discussion

The fault rates for our project stayed constant between our mixed group walkthrough and our original group walkthrough. The difference between the two walkthroughs was the severity of the faults we found. Our mixed group walkthrough found only major faults that needed to be fixed immediately, while the original group walkthrough found only minor faults that still needed to be addressed but did not cause major breakdowns in our project.

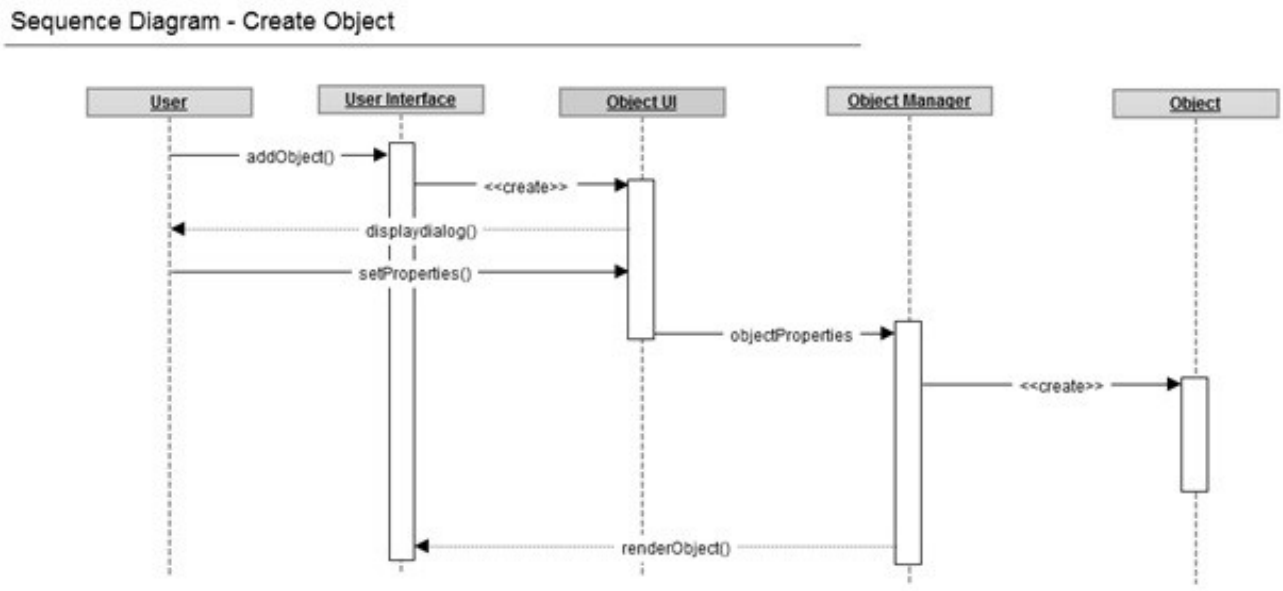
Appendix I: Use-Case Diagram



This diagram describes potential use cases for both the student and the teacher.

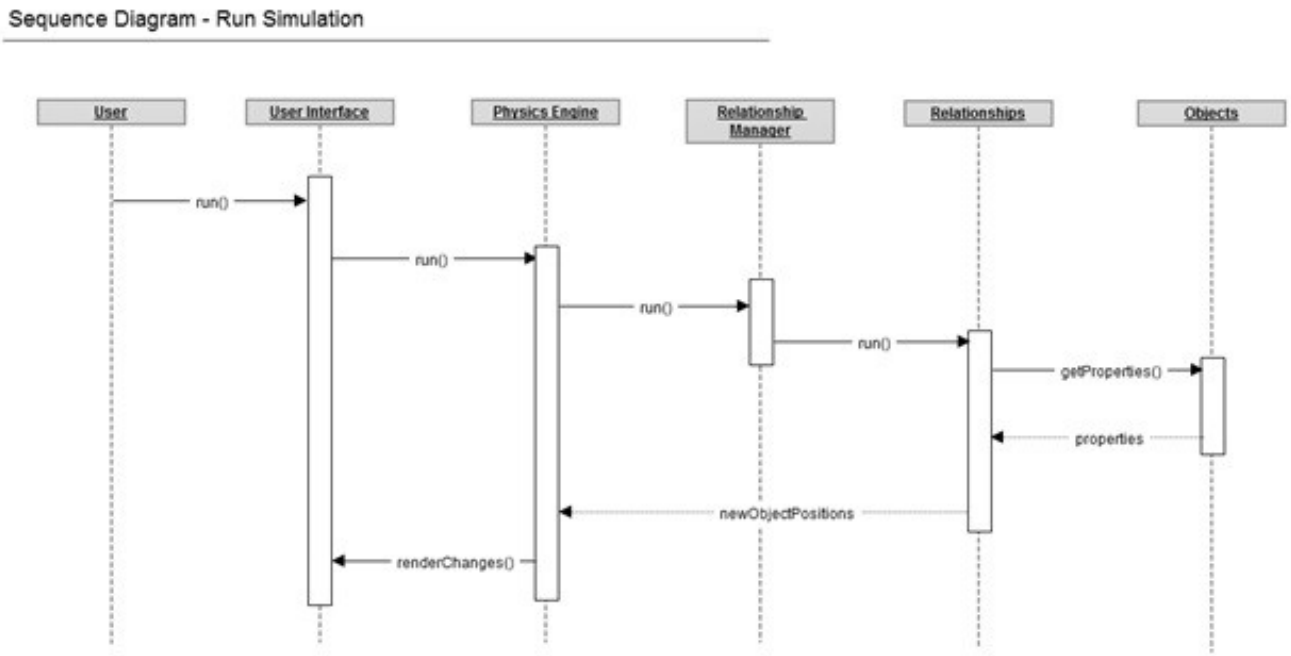
Appendix II: Sequence Diagrams

Create Object



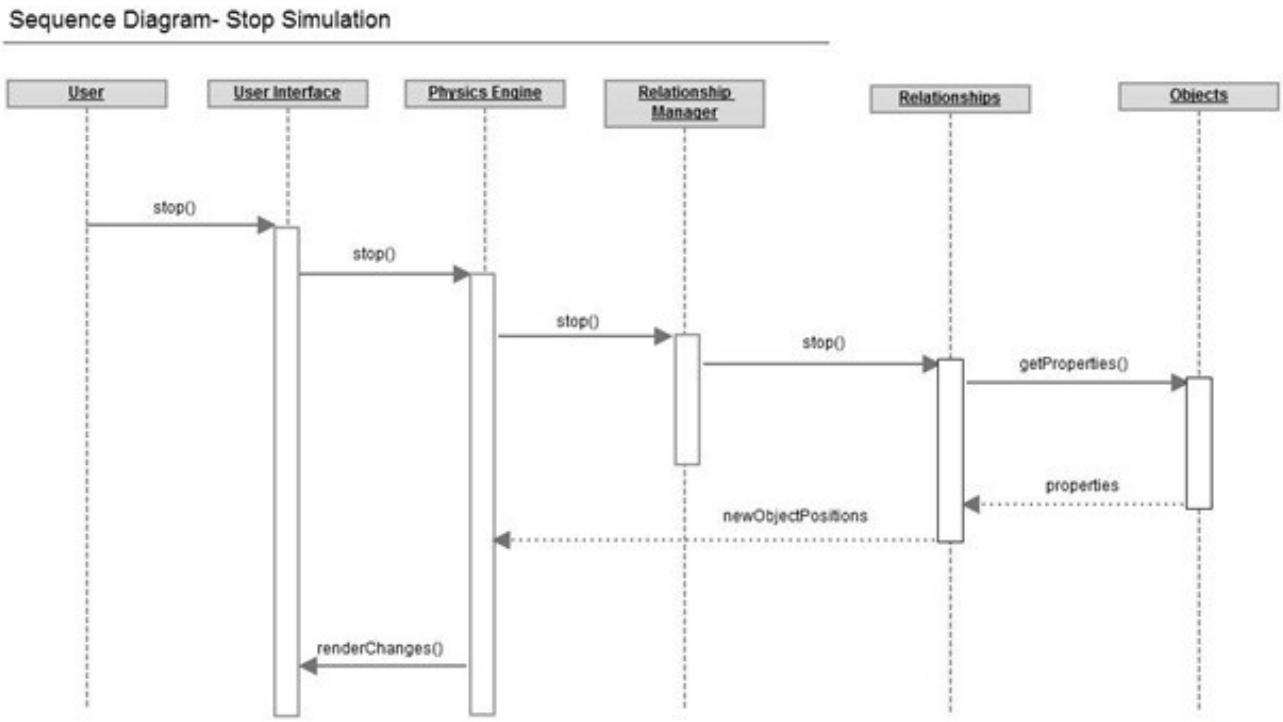
This diagram describes the methods and events taken when the user wishes to create an object.

Run Simulation



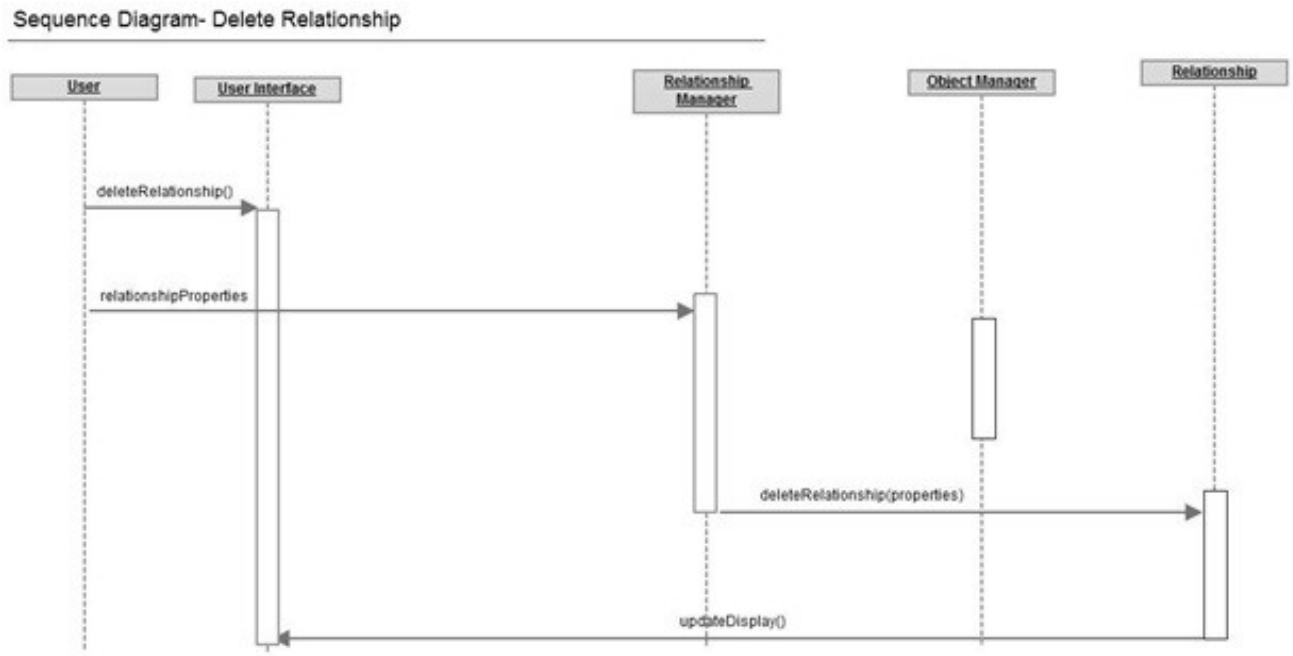
This diagram describes the methods and events taken when the user wishes to start the simulation

Stop Simulation



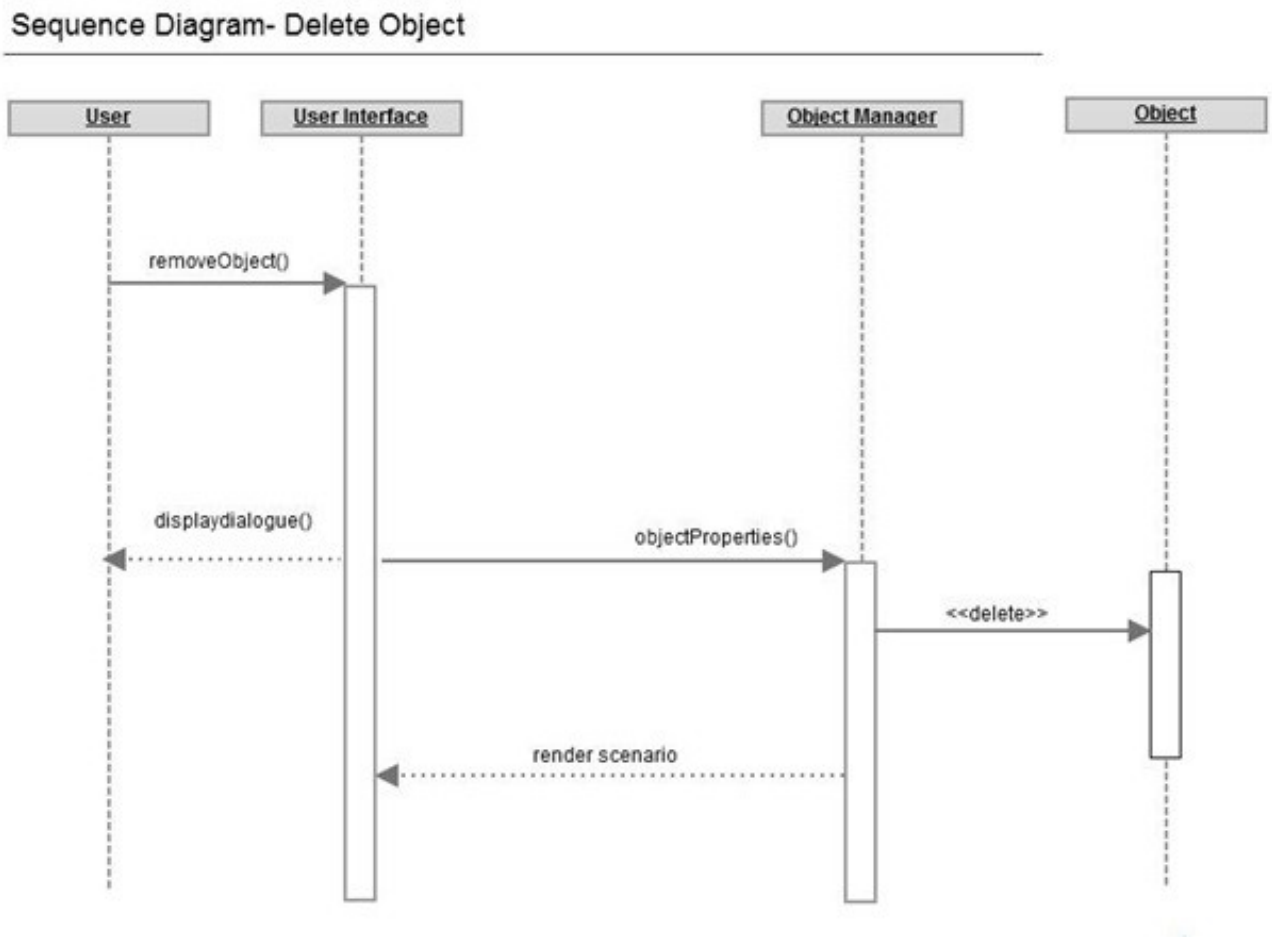
This diagram describes the methods and events taken when the user wishes to stop the simulation.

Delete Relationship



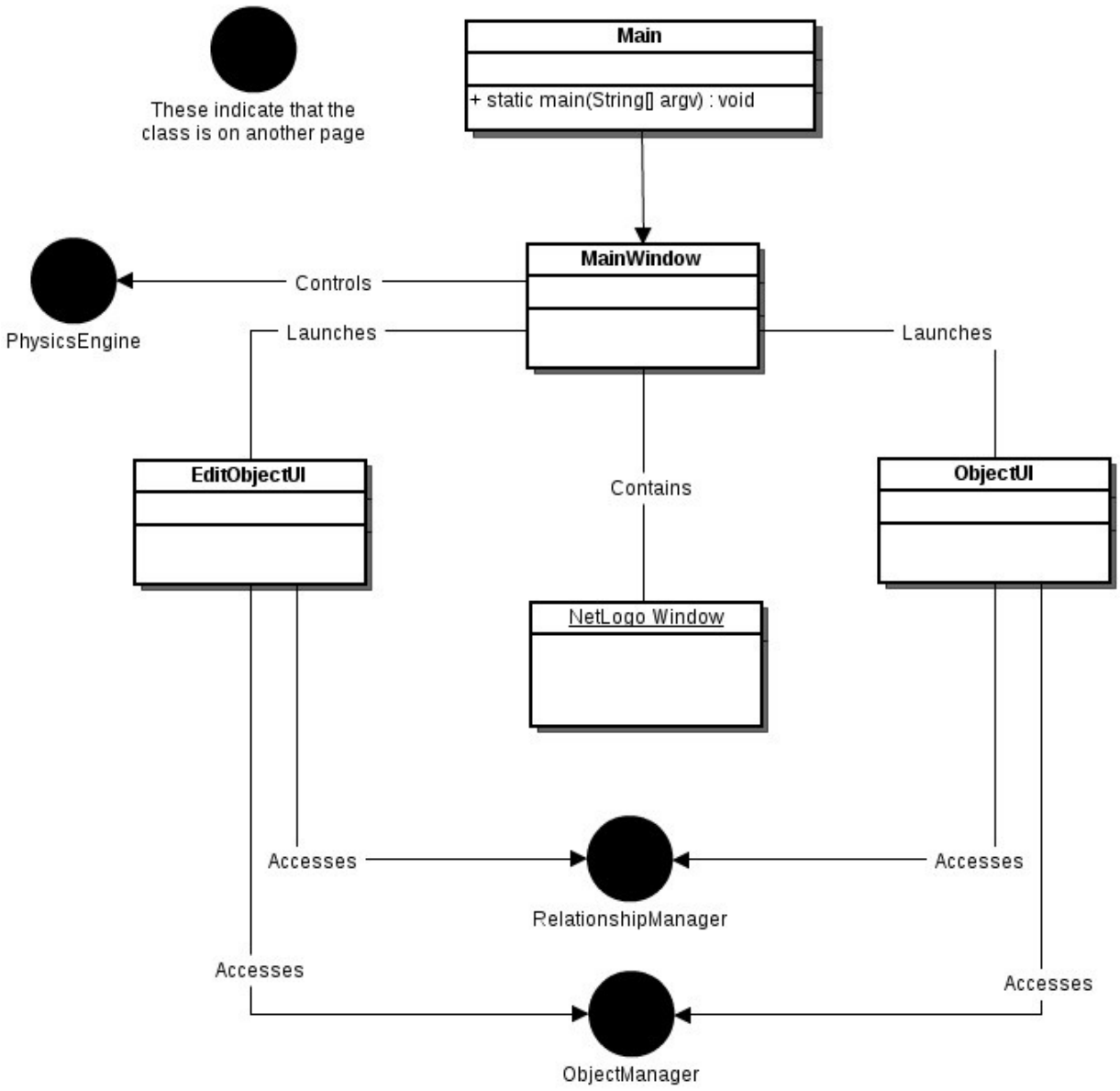
This diagram describes the methods and events taken when the user wishes to delete a relationship.

Delete Object

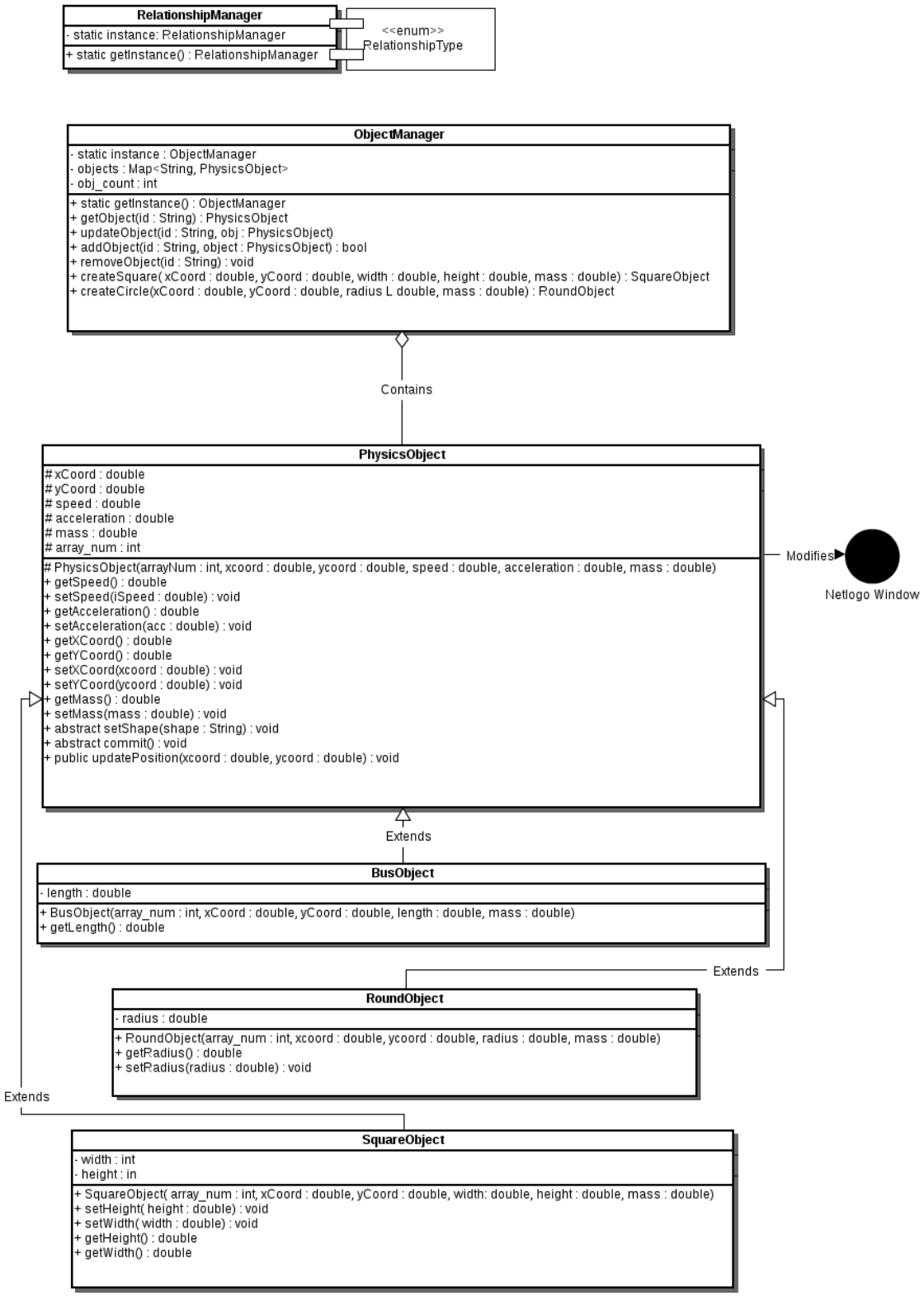


This diagram describes the methods and events taken when the user wishes to delete an object.

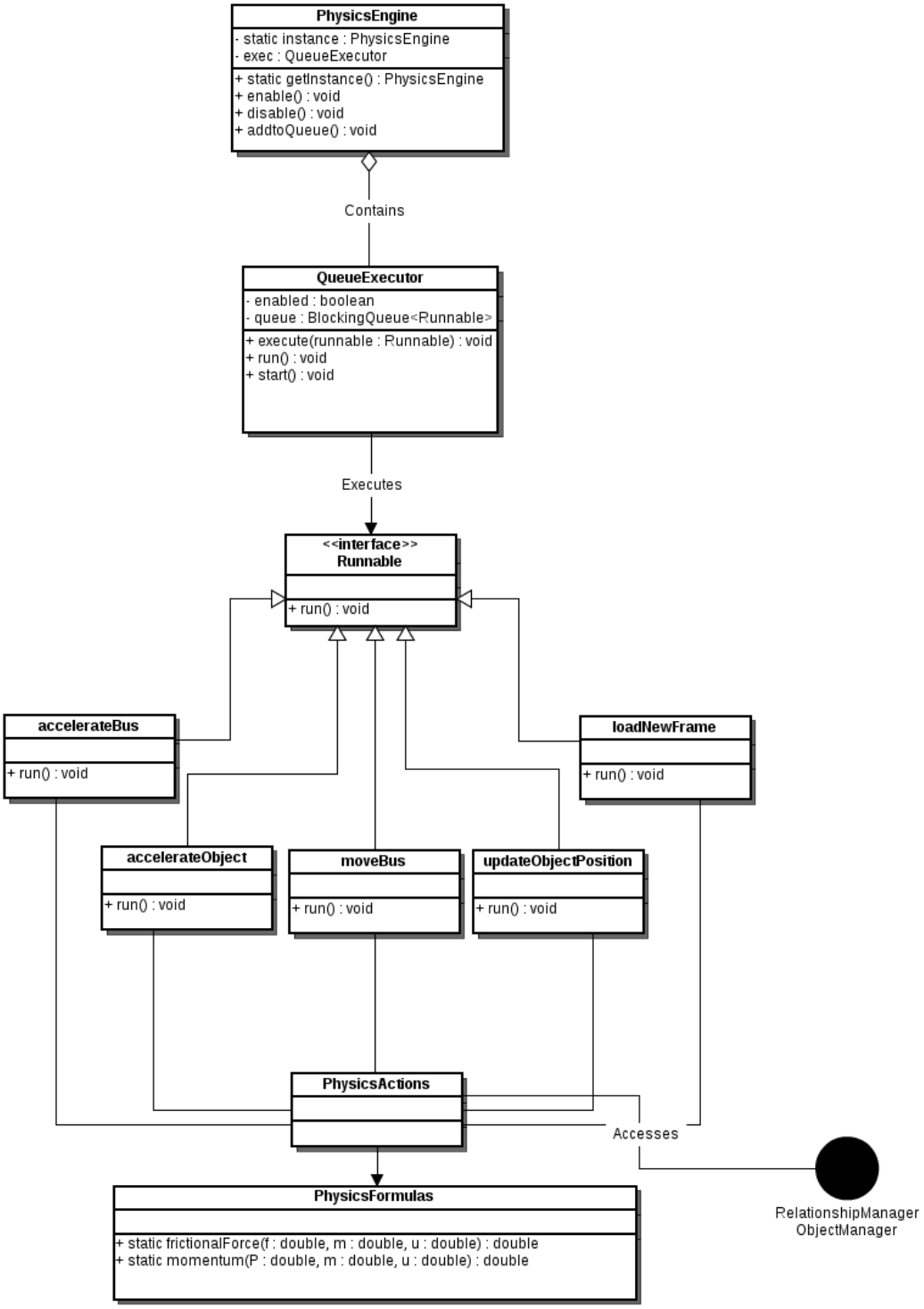
Class Diagram- Main and User Interfaces



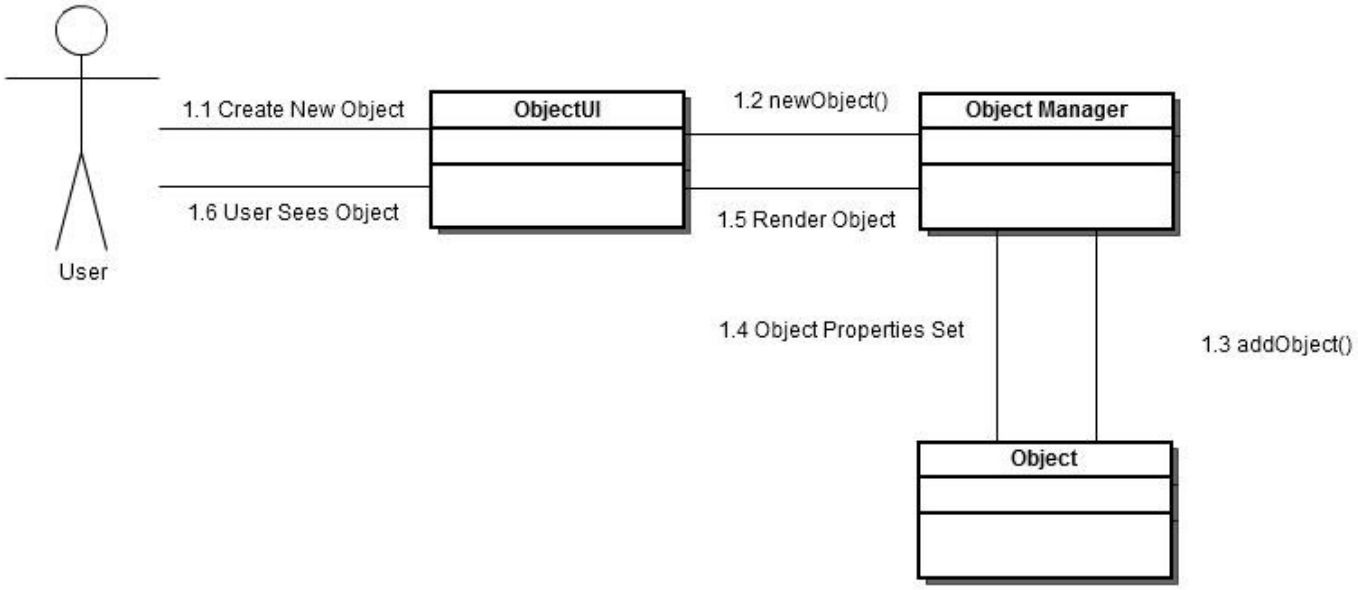
Managers Class



Physics Classes



Appendix IV: Communication Diagram



This diagram describes the methods and communications used to create an object.

Table Of Content

Class PhysicsActions

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions
```

< Constructors >

public class **PhysicsActions**
extends java.lang.Object

Class that contains inner classes that implement the Runnable interface. All inner classes should implement the Runnable interface and be designed as one specific action for the PhysicsEngine work queue.

Author:
Matthew Shea

Constructors

PhysicsActions

```
public  PhysicsActions()
```

Class PhysicsActions.accelerateBus

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions.accelerateBus
```

All Implemented Interfaces:
java.lang.Runnable

< Constructors > < Methods >

public static class **PhysicsActions.accelerateBus**
extends java.lang.Object
implements java.lang.Runnable

Calculates the new speed for the bus based on the current speed and the acceleration.

Author:
Matthew Shea

Constructors

PhysicsActions.accelerateBus

```
public  PhysicsActions.accelerateBus()
```

Methods

run

```
public void run()
```

Class PhysicsActions.accelerateObject

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions.accelerateObject
```

All Implemented Interfaces:
java.lang.Runnable

< Constructors > < Methods >

---

public static class **PhysicsActions.accelerateObject**
extends java.lang.Object
implements java.lang.Runnable

Accelerates the object based on the calculated friction and speed of the object

Author:
Matthew Shea

Constructors

PhysicsActions.accelerateObject

```
public  PhysicsActions.accelerateObject()
```

Methods

run

```
public void run()
```

Class PhysicsActions.loadNewFrame

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions.loadNewFrame
```

All Implemented Interfaces:
java.lang.Runnable

< Constructors > < Methods >

---

public static class **PhysicsActions.loadNewFrame**
extends java.lang.Object
implements java.lang.Runnable

This Runnable adds the sequence of other runnables to the physics engine and then itself at the end. This is added to the PhysicsEngine's work queue on the enable() call.

Author:
Matthew Shea

Constructors

PhysicsActions.loadNewFrame

```
public  PhysicsActions.loadNewFrame()
```

Methods

run

```
public void run()
```

Class PhysicsActions.moveBus

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions.moveBus
```

All Implemented Interfaces:
java.lang.Runnable

< Constructors > < Methods >

---

public static class **PhysicsActions.moveBus**
extends java.lang.Object
implements java.lang.Runnable

The bus itself should never move on the screen. We will simulate movement by adjusting the position of the background.

Author:
Matthew Shea

Constructors

PhysicsActions.moveBus

```
public  PhysicsActions.moveBus()
```

Methods

run

```
public void run()
```

Class PhysicsActions.updateObjectPosition

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsActions.updateObjectPosition
```

All Implemented Interfaces:
java.lang.Runnable

< Constructors > < Methods >

---

public static class **PhysicsActions.updateObjectPosition**

extends java.lang.Object
implements java.lang.Runnable

Updates the object's position on the netlogo display

Author:
Matthew Shea

Constructors

PhysicsActions.updateObjectPosition

`public` **`PhysicsActions.updateObjectPosition`**`()`

Methods

run

`public void` **`run`**`()`

Class PhysicsEngine

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsEngine
```

< Methods >

---

public class **PhysicsEngine**
extends java.lang.Object

This class is a singleton wrapper around a QueueExecutor that runs the physics calculations and updates the display. Most of the work will be done on this QueueExecutor.

Author:
syddraf

Methods

addtoQueue

`public void` **`addtoQueue`**`(java.lang.Runnable runnable)`

Adds an item to the queue for the thread to execute.

Parameters:

runnable - PhysicsAction for the engine to perform

---

disable

`public void` **`disable`**`()`

Calling this method will allow the engine to finish it's current operation, and then suspend and flush the queue.

---

enable

`public void` **`enable`**`()`

This method will allow the engine to perform operations in its run queue.

---

getInstance

`public static` `PhysicsEngine` **`getInstance`**`()`

This method returns the singleton PhysicsEngine object.

Returns:

The singleton PhysicsEngine

Class PhysicsFormulas

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.engine.PhysicsFormulas
```

< Constructors > < Methods >

---

public class **PhysicsFormulas**
extends java.lang.Object

Constructors

PhysicsFormulas

```
public  PhysicsFormulas()
```

Methods

frictionalForce

```
public static java.lang.Double frictionalForce(java.lang.Double f,
                                               java.lang.Double m,
                                               java.lang.Double u)
                                throws
PhysicsFormulas.TooManyNullArgumentsException
```

Implements the formula F = um; Pass one parameter as null and the calculated value of that parameter will be returned.

Parameters:

f - The frictional force applied to the object
m - The mass of the object
u - The coefficient of friction

Returns:

The returned value is the calculated value of the parameter that was passed as null.

Throws:

edu.vu.vuse.cs278.g3.engine.PhysicsFormulas.TooManyNullArgumentsException - Thrown if more than one argument is null.

---

momentum

```
public static java.lang.Double momentum(java.lang.Double P,
                                        java.lang.Double m,
                                        java.lang.Double v)
                                throws
PhysicsFormulas.TooManyNullArgumentsException
```

This function implements the formula for an objects momentum.

Parameters:

P - The momentum of the object.
m - The mass of the object.
v - The velocity of the object.

Returns:

The value returned is the argument that was set to null in the arguments.

Throws:

edu.vu.vuse.cs278.g3.engine.PhysicsFormulas.TooManyNullArgumentsException - Thrown if more than one argument is null.

Class PhysicsFormulas.TooManyNullArgumentsExc

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--edu.vu.vuse.cs278.g3.engine.PhysicsFormulas.TooManyNullArgumentsEx
```

All Implemented Interfaces:
java.io.Serializable

< Constructors > < Methods >

---

public static class **PhysicsFormulas.TooManyNullArgumentsException**
extends java.lang.Exception

This exception indicates that too many null arguments were passed to one of the above PhysicsFormulas.

Author:
Matthew Shea

Constructors

PhysicsFormulas.TooManyNullArgumentsException

```
public  PhysicsFormulas.TooManyNullArgumentsException(int number)
```

Methods

getError

```
public java.lang.String getError()
```

Class EditObjectUI

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--java.awt.Window
                |
                +--java.awt.Frame
                    |
                    +--javax.swing.JFrame
                        |
                        +--edu.vu.vuse.cs278.g3.gui.EditObjectUI
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

< Constructors > < Methods >

---

public class **EditObjectUI**
extends javax.swing.JFrame

This class holds the code for the Edit Oject Dialog box.

Author:

Amber Maria

Constructors

EditObjectUI

```
public  EditObjectUI()
```

The constructor for the class. Since this is Edit Object, we want to "fill in" the fields for the user to see what the current attributes are.

Methods

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args - the command line arguments

Class MainWindow

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--java.awt.Window
                |
                +--java.awt.Frame
                    |
                    +--javax.swing.JFrame
                        |
                        +--edu.vu.vuse.cs278.g3.gui.MainWindow
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

< Methods >

---

public class **MainWindow**
extends javax.swing.JFrame

This class handles the main window of the program where the NetLogo is embedded.

Author:

Amber Maria

Methods

command

`public void` **`command`**`(java.lang.String arg)`

Sends arg to embedded NetLogo

Parameters:

arg -

---

getInstance

`public static` `MainWindow` **`getInstance`**`()`

Gets the single instance of the Main Window

Returns:

instance of Main Window

Class ObjectUI

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--java.awt.Window
                |
                +--java.awt.Frame
                    |
                    +--javax.swing.JFrame
                        |
                        +--edu.vu.vuse.cs278.g3.gui.ObjectUI
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

< Constructors > < Methods >

---

public class **ObjectUI**
extends javax.swing.JFrame

Class that handles the Add Object dialog

Author:

Amber

Constructors

ObjectUI

`public` **`ObjectUI`**`()`

Constructor. Calls initComponents to declare and initialize all fields of the dialog and sets the sliders to a neutral value.

Methods

main

`public static void` **`main`**`(java.lang.String[] args)`

Parameters:

args - the command line arguments

Class BusObject

```
java.lang.Object
    |
    +--PhysicsObject
        |
        +--edu.vu.vuse.cs278.g3.model.BusObject
```

< Methods >

---

public class **BusObject**
extends PhysicsObject

Methods

commit

```
public void commit()
```

Overrides:

commit in class PhysicsObject

---

getLength

```
public double getLength()
```

---

setShape

```
public void setShape(java.lang.String shape)
```

Overrides:

setShape in class PhysicsObject

---

updatePosition

```
public void updatePosition(double xcoord,
                           double ycoord)
```

Overrides:

updatePosition in class PhysicsObject

Class ObjectManager

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.model.ObjectManager
```

< Methods >

public class **ObjectManager**
extends java.lang.Object

The ObjectManager serves to manage all objects involved in the model. It's two primary functions are maintaining a map of all objects involved in the model and facilitating the creation of the objects.

Author:

Matthew Shea

Methods

addObject

```
public boolean addObject(java.lang.String id,
                         PhysicsObject object)
```

Adds an object to the manager with the specified id.

Parameters:

id - the UNIQUE identifier of the object
object - The object reference

Returns:

True if the object was successfully added and False if it already exists.

---

createBus

```
public BusObject createBus(double _xCoord,
                           double _yCoord,
                           double _length,
                           double mass)
```

Creates a BusObject with the specified parameters and returns the reference.

Parameters:

_xCoord - The x position of the bus
_yCoord - The y position of the bus
_length - The length of the bus

Returns:

Object reference

createCircle

```
public RoundObject createCircle(double _xCoord,
                                 double _yCoord,
                                 double _radius,
                                 double mass)
```

Creates a RoundObject with the specified parameters and returns the reference.

Parameters:

_xCoord - The x position of the circle
_yCoord - The y position of the circle
_radius - The radius of the circle

Returns:

Object reference

---

createSquare

```
public SquareObject createSquare(double _xCoord,
                                 double _yCoord,
                                 double _width,
                                 double _height,
                                 double mass)
```

Creates a SquareObject with the specified parameters and returns the reference.

Parameters:

_xCoord - The x position of the square
_yCoord - The y position of the square
_width - The width of the object
_height - The height of the object

Returns:

Object reference

---

getInstance

```
public static ObjectManager getInstance()
```

Returns the singleton instance of the ObjectManager.

Returns:

Singleton instance of the ObjectManager

---

getObject

`public PhysicsObject getObject(java.lang.String id)`

Returns the reference to the object associated with the id

Parameters:

id - identifier of the object

Returns:

The reference to the object or null if the object is not found.

---

removeObject

`public void removeObject(java.lang.String id)`

Remove the object associated with the specified unique id.

Parameters:

id - Unique ID of the object to remove.

---

updateObject

```
public void updateObject(java.lang.String id,
                         PhysicsObject obj)
```

Replaces the specified key with the new object.

Parameters:

id - Identifier
obj - New Object

Class PhysicsObject

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.model.PhysicsObject
```

Direct Known Subclasses:
BusObject, RoundObject, SquareObject

< Fields > < Constructors > < Methods >

---

public abstract class **PhysicsObject**
extends java.lang.Object

Fields

acceleration

`protected double` **`acceleration`**
The acceleration of the object, measured in pixels per frame per frame

---

array_num

`protected int` **`array_num`**
The number in the array of the Netlogo object

---

mass

`protected double` **`mass`**
The mass of the object in kg

---

speed

`protected double` **`speed`**
The speed of the object, measured in pixels per frame

---

xCoord

`protected double` **`xCoord`**
The x coordinate on the Netlogo display of the object

---

yCoord

`protected double` **`yCoord`**
The y coordinate on the Netlogo display of the object

Constructors

PhysicsObject

```
protected  PhysicsObject(int arrayNum,
                         double xcoord,
                         double ycoord,
                         double speed,
                         double acceleration,
                         double mass)
```

Creates a PhysicsObject with the specified parameters.

Parameters:

arrayNum - The number in the Netlogo array
xcoord - The xcoord in the Netlogo display
ycoord - The ycoord in the Netlogo display
speed - The speed of the object. Positive indicates rightward motion.
acceleration - The acceleration of the object.
mass - The mass of the object.

Methods

commit

```
public abstract void commit()
```

Commits the changes to the NetLogo backend to update the graphical display. You do not need to call this function unless the POSITION has changed.

---

getAcceleration

```
public double getAcceleration()
```

Returns the acceleration of the object

Returns:

The object's acceleration

---

getMass

```
public double getMass()
```

Returns the mass of the object

Returns:

The object's mass

---

getSpeed

`public double` **`getSpeed`**`()`

Returns the speed of the object

Returns:

The object's speed

---

getXCoord

`public double` **`getXCoord`**`()`

Returns the x coordinate of the object

Returns:

The object's X coordinate

---

getYCoord

`public double` **`getYCoord`**`()`

Returns the y coordinate of the object

Returns:

The object's Y coordinate

---

setAcceleration

`public void` **`setAcceleration`**`(double acc)`

Set's the object's acceleration

Parameters:

acc - A double indicating the acceleration of the object.

---

setMass

`public void` **`setMass`**`(double mass)`

Sets the mass of the object

Parameters:

mass - The new mass of the object

---

setShape

```
public abstract void setShape(java.lang.String shape)
```

Immediately changes the object's shape. Must exist in netlogo shapes library.

Parameters:

shape -

---

setSpeed

```
public void setSpeed(double iSpeed)
```

Sets the speed of the object.

Parameters:

iSpeed - The speed in pixels/frame

---

setXCoord

```
public void setXCoord(double xcoord)
```

Sets the x coordinate of the object

Parameters:

xcoord - x coordinate in pixels

---

setYCoord

```
public void setYCoord(double ycoord)
```

Sets the y coordinate of the object

Parameters:

ycoord - y coordinate in pixels

---

updatePosition

```
public void updatePosition(double xcoord,
                           double ycoord)
```

Updates the position with the specified xcoords and ycoords

Parameters:

xcoord -
ycoord -

Class RelationshipManager

```
java.lang.Object
    |
    +--edu.vu.vuse.cs278.g3.model.RelationshipManager
```

< Methods >

public class **RelationshipManager**
extends java.lang.Object

Singleton object to manage the list of Relationships

Author:

Matthew Shea

Methods

getInstance

```
public static RelationshipManager getInstance()
```

Returns the singleton instance of this type

Returns:

Singleton instance

---

getRelationship

```
public RelationshipTypes getRelationship(java.lang.String object1,
                                         java.lang.String object2)
```

This function fetches a relationship between two objects. The order of the objects is important. For example, getRelationship("a", "b") is not the same as getRelationship("b", "a")

Parameters:

object1 - The string id of the first object
object2 - The string id of the second object

Returns:

The relationshipType, if available, or NO_RELATIONSHIP if not.

---

setRelationship

```
public void setRelationship(java.lang.String object1,
                            java.lang.String object2,
                            RelationshipTypes type)
```

This function sets a relationship between two objects. The order of the objects is important. For example, getRelationship("a", "b") is not the same as getRelationship("b", "a")

Parameters:

object1 - The string id of the first object
object2 - The string id of the second object
type - The RelationshipType to define

Class RelationshipTypes

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--edu.vu.vuse.cs278.g3.model.RelationshipTypes
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

---

public final class **RelationshipTypes**
extends java.lang.Enum

Enumeration that defines possible relationship types

Author:

Matthew Shea

Fields

ABOVE_RESTRAINED

```
public static final RelationshipTypes ABOVE_RESTRAINED
```

ABOVE_UNRESTRAINED

```
public static final RelationshipTypes ABOVE_UNRESTRAINED
```

BEHIND_ATTACHED

public static final RelationshipTypes **BEHIND_ATTACHED**

INSIDE_RESTRAINED

public static final RelationshipTypes **INSIDE_RESTRAINED**

INSIDE_UNRESTRAINED

public static final RelationshipTypes **INSIDE_UNRESTRAINED**

NO_RELATIONSHIP

public static final RelationshipTypes **NO_RELATIONSHIP**

Methods

valueOf

public static RelationshipTypes **valueOf**(java.lang.String name)

---

values

public static edu.vu.vuse.cs278.g3.model.RelationshipTypes[] **values**()

Class RoundObject

```
java.lang.Object
    |
    +--PhysicsObject
        |
        +--edu.vu.vuse.cs278.g3.model.RoundObject
```

< Methods >

---

public class **RoundObject**
extends PhysicsObject

Methods

commit

```
public void commit()
```

Overrides:

commit in class PhysicsObject

---

getRadius

```
public double getRadius()
```

---

setRadius

```
public void setRadius(double radius)
```

---

setShape

```
public void setShape(java.lang.String shape)
```

Overrides:

setShape in class PhysicsObject

---

updatePosition

```
public void updatePosition(double xcoord,
                           double ycoord)
```

Overrides:

updatePosition in class PhysicsObject

Class SquareObject

```
java.lang.Object
    |
    +--PhysicsObject
        |
        +--edu.vu.vuse.cs278.g3.model.SquareObject
```

---

public class **SquareObject**
extends PhysicsObject

Methods

commit

`public void commit()`

Overrides:

commit in class PhysicsObject

---

getHeight

`public double getHeight()`

---

getWidth

`public double getWidth()`

---

getXCoord

`public double getXCoord()`

Overrides:

getXCoord in class PhysicsObject

---

getYCoord

`public double getYCoord()`

Overrides:

getYCoord in class PhysicsObject

---

setHeight

```
public void setHeight(double height)
```

---

setShape

```
public void setShape(java.lang.String shape)
```

Overrides:

setShape in class PhysicsObject

---

setWidth

```
public void setWidth(double width)
```

---

updatePosition

```
public void updatePosition(double xcoord,
                           double ycoord)
```

Overrides:

updatePosition in class PhysicsObject

INDEX

S

U

V

X

Y