Documentación
ELO 329 - Tarea 2

Tarea 2

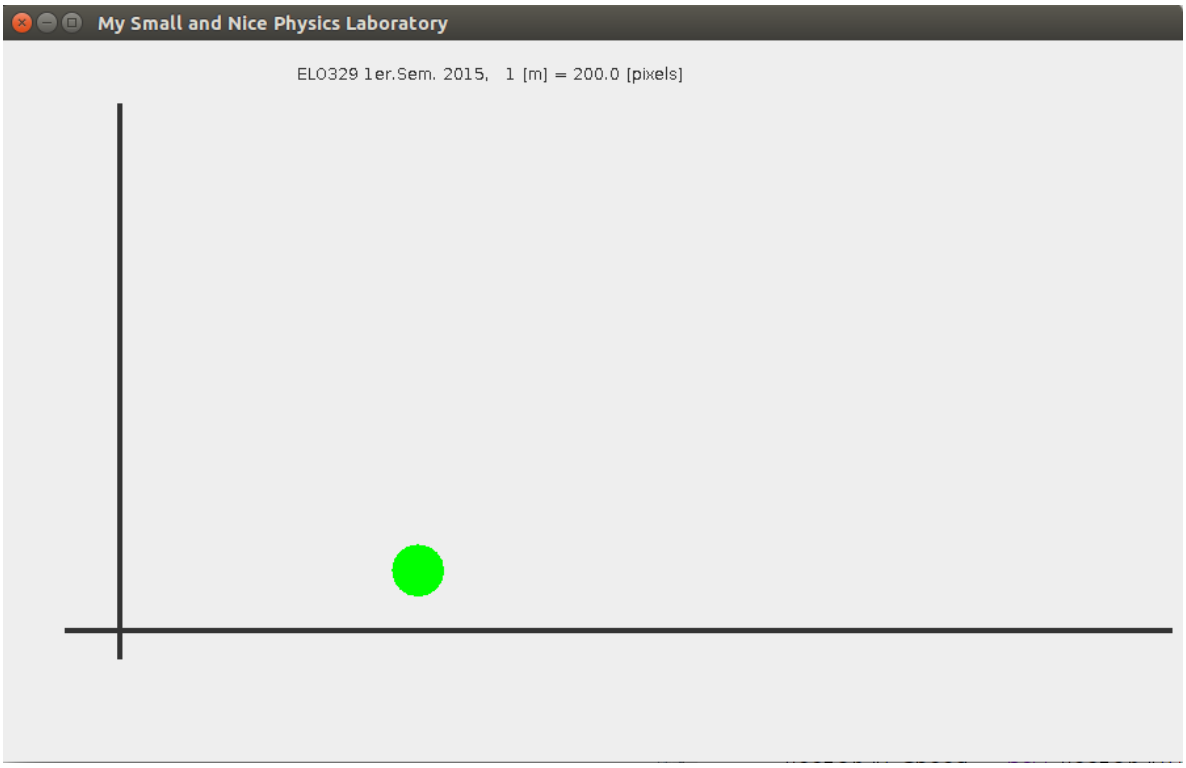
Descripción General

Como en la tarea 1, en esta tarea se modela la interacción de resortes, bolas, osciladores, amortiguadores y elásticos; pero esta vez se hace en forma gráfica. Como en la tarea 1, esta vez deberemos desarrollar cuatro etapas en un proceso de desarrollo iterativo e incremental. En esta metodología cada etapa debe generar una versión ejecutable de la solución.

Parte 1

Descripción:

Como primer paso, debemos implementar y completar el código base entregado para correr **PhysicsLab.java** proporcionado para esta etapa. Debemos conseguir que la clase **MyWorldView**, en particular, muestre gráficamente, en esta etapa, el movimiento de una bola con Movimiento Rectilíneo Uniforme, en una ventana como la siguiente:



Archivos:

1) **PhysicsLab.java**
   Corresponde a la clase main de la tarea, que se encarga de configurar y determinar lo necesario para la creación de la ventana gráfica y la disposición de los elementos en ella.

2) **MyWorld.java**
   Es el lugar donde se agregan los elementos y donde interaccionan entre sí. Esta clase se encarga de hacer la simulación, utiliza los parámetros entregados por el **PhysicsLab** para generar dicha simulación y además, se encarga de actualizar el estado de los objetos basándose en el estado general de la simulación. En **MyWorldView.java** se configura lo necesario para obtener la vista de **MyWorld.**

3) **MyWorldView.java**
   Define lo necesario para dibujar el entorno en que los objetos interaccionan. Además establece lo necesario para usar coordenadas métricas en vez de pixeles para graficar.

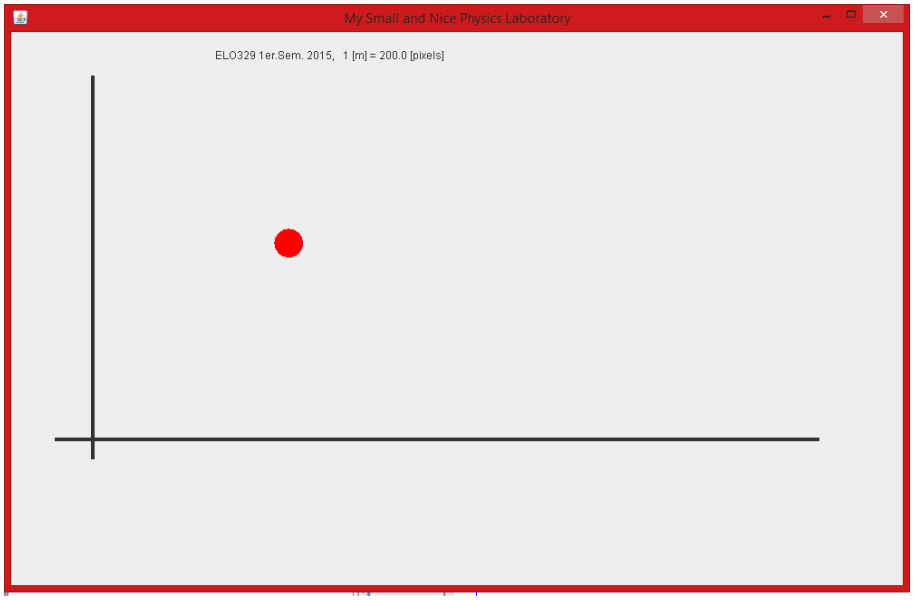
4) **PhysiscsElements.java**
   Se encarga de crear los elementos físicos. En esta primera etapa sólo tenemos a la pelota.

5) **Ball.java**
   Primer elemento físico implementado, por tanto, extiende de la clase **PhysicsElement**. Como elemento físico tiene características: id, masa, posición, velocidad. Además debe ser capaz de calcular su siguiente estado, de acuerdo al delta tiempo. Actualiza el estado. Define el color. **BallView** se encargará de dibujar la pelota para poder agregarla a la representación gráfica.

6) **BallView.java**
   Clase que establece los parámetros para el dibujo gráfico de la pelota. Como el color, el radio, obtiene la posición y el método Paint se encarga de dibujarla.

7) **Vector2D.java**
   Clase que establece lo necesario para trabajar y operar con vectores en 2 dimensiones. Se establece la suma, la multiplicación por escalar y la resta de vectores. La posición de la pelota es uno de los vectores usados en esta tarea.

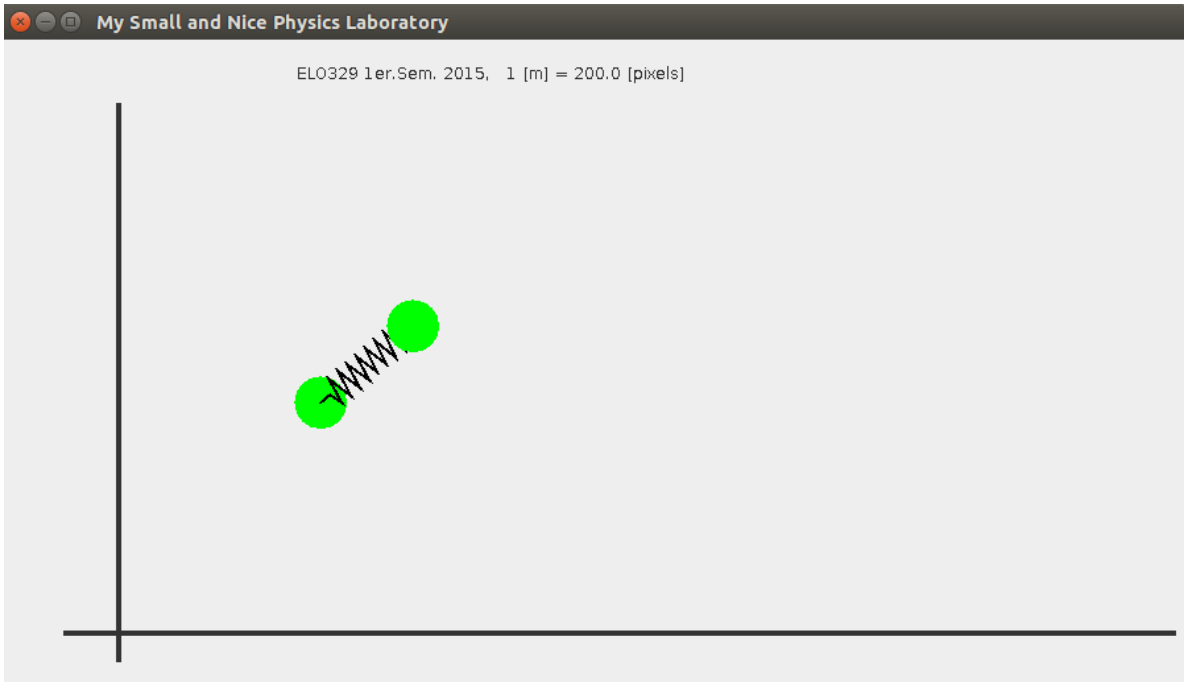
Imagen ejecución del programa.



Parte 2

Descripción:

En esta segunda parte debemos codificar el código entregado para permitir, al igual que en la primera parte, la simulación gráfica, en este caso, de dos pelotas, y ahora se incluye un resorte que está unido a estas dos pelotas. Debemos generar un escenario como el siguiente:



Archivos:

1) **PhysicsLab.java**
   En esta segunda parte, sigue siendo la clase main de la tarea. En esta segunda parte agrega las características necesarias para incorporar el resorte que debe estar unido a algún otro elemento, en este caso a dos pelotas.

2) **MyWorld.java**
   Es el lugar donde se agregan los elementos y donde interaccionan entre sí. Esta clase se encarga de hacer la simulación.

3) **MyWorldView.java**
   Define lo necesario para dibujar el entorno en que los objetos interaccionan. En este caso tendremos las dos pelotas y el resorte.

4) **PhysiscsElements.java**
   Clase abstracta encargada de crear los elementos físicos En esta segunda etapa tenemos el resorte y las pelotas, sin embargo, más adelante tendremos bolas, resortes, osciladores, amortiguadores y elásticos que todos extenderán de esta clase.

5) **Ball.java**
   Elemento ya conocido, extiende de la clase **PhysicsElement** y ahora, además, implementa de la interfaz **Attachable**. En este caso está conectada a un resorte, el cual ejerce una fuerza sobre la pelota, generando una aceleración que afectará en su velocidad y, por tanto, en su posición. De la misma forma hay que codificar para que se pueda acoplar un resorte.

6) **BallView.java**
   Clase que establece los parámetros para el dibujo gráfico de la pelota. El método Paint de esta clase se encarga de dibujar a la pelota utilizando ciertas características de ella como el color, el radio, y la posición.

7) **Spring.java**
   Nuevo elemento físico extiende de la clase **PhysicsElement**. Al igual la pelota, tiene características, largo, rigidez, la vista (que se usara en **Springview** para dibujarlo). Considerando la variación de su largo generará un vector fuerza que modificará y afectara en la posición de los objetos conectados a él, en este caso, las pelotas.

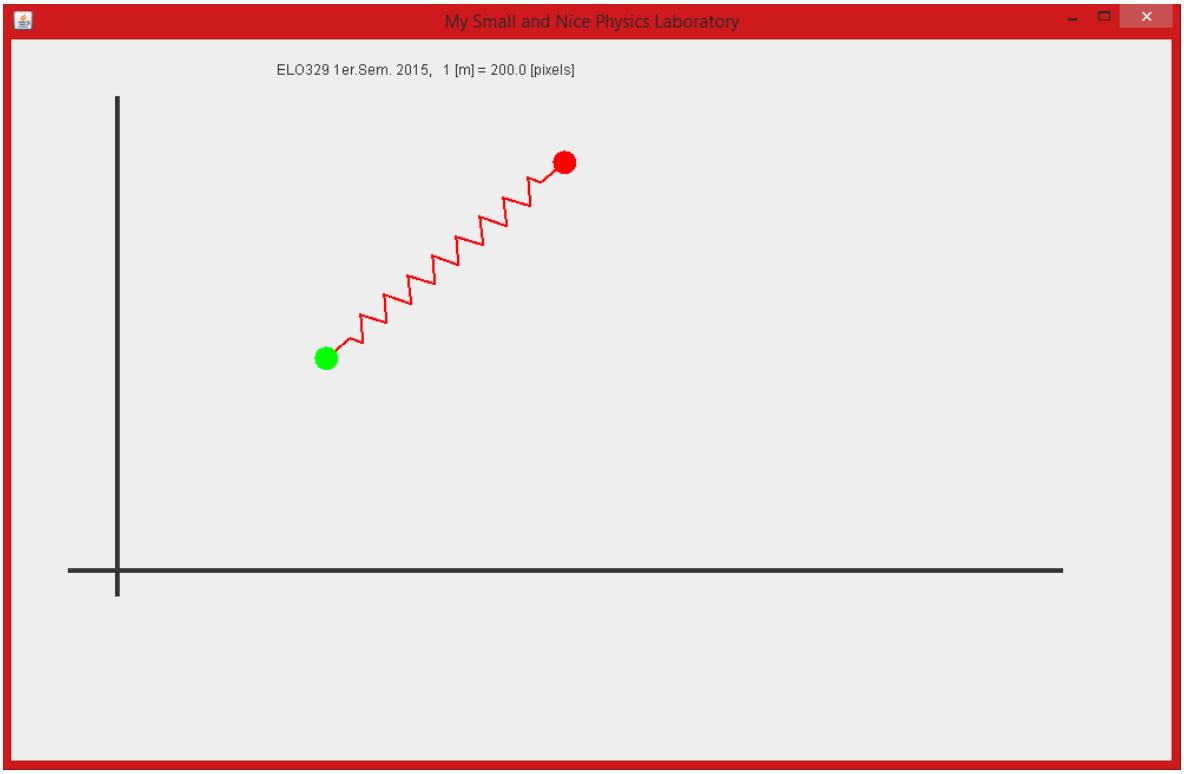
8) **SpringView.java**
   Esta clase se encarga de la vista del resorte, para ello usa el color y establece los determinados puntos para dibujar las líneas del resorte. Y sus diferentes formas a medida que se estira y se encoge.

9) **Vector2D.java**
   En esta clase se establece la suma, la multiplicación por escalar y la resta de vectores, las características necesarias para trabajar con vectores en 2 dimensiones. Dentro de los vectores a mencionar tenemos, la fuerza del resorte y la posición, tanto de la pelota como el resorte.

10) **Attachable.java**
    Interfaz que define los métodos que deben implementarse en este tipo de objetos. La pelota es uno de ellos. Son los objetos a los que se le puede conectar otro y no al revés.

11) **Simulateable.java**
    Interfaz de la que la pelota implementa sus métodos, puesto que su siguiente estado cambia como resultado de su estado anterior y su interacción con otros elementos. Esta Interfaz se crea para evitar métodos con implementaciones vacías.

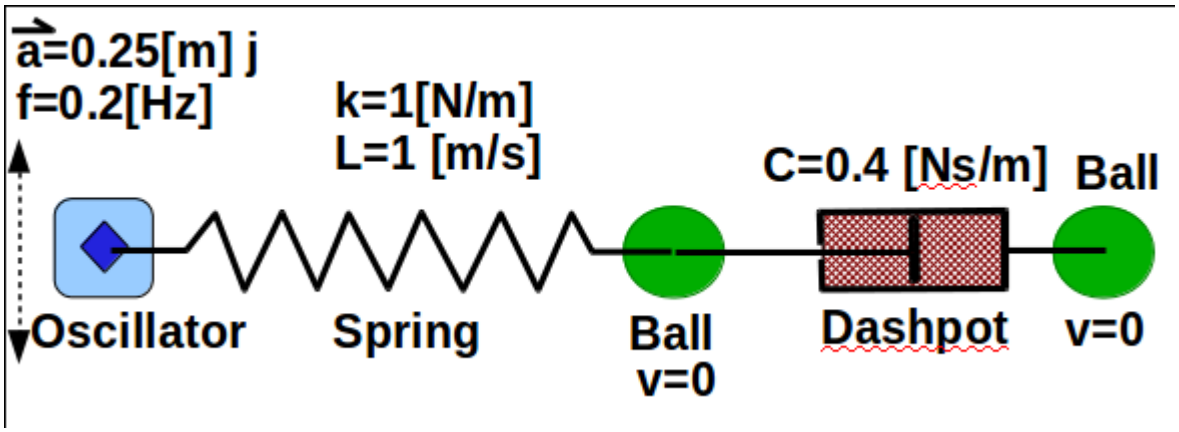
Imagen ejecución del programa



Parte 3

Descripción:

En esta tercera parte, además de las clases ya mencionadas debemos implementar la clase **Oscillator.java** que posee características similares a la pelota. Además tendremos otra clase llamada **Dashpot.java,** otro elemento físico que se comporta de la misma forma que el resorte. La representación gráfica debe ser de la siguiente situación física.



Archivos:

1) **PhysicsLab.java**
   Es el main de la tarea, se encarga de intervenir en el archivo **MyWorld.java** y lo crea, que es donde finalmente interaccionaran los elementos, que se verán gracias al **MyWorldView.java**. En esta tercera parte agrega las características necesarias para incorporar el oscilador y el amortiguador.

2) **MyWorld.java**
   Mundo virtual en el que interaccionan los elementos físicos, **MyWorldView** se encarga de representarlo gráficamente.

3) **MyWorldView.java**
   Define lo necesario para dibujar el entorno en que los objetos interaccionan. En este caso tendremos el oscilador, un resorte, dos pelotas y un amortiguador.

4) **PhysiscsElements.java**
   La función de esta clase es crear los elementos físicos. En esta etapa se incluyen dos nuevos elementos físicos que son el amortiguador (Dashpot) y el Oscilador (Oscillator)

5) **Attachable.java**
   El resorte y, ahora, el amortiguador pueden acoplarse a este tipo de elementos físicos. De esta Interfaz implementan la pelota y el oscilador.

6) **Connector.java**
   Nueva interfaz a la cual pertenecen los elementos físicos que se pueden unir a otros, es el caso del resorte y el amortiguador, éstos deben implementar los métodos de la interfaz en sus respectivas clases.

7) **Simulateable.java**
   Interfaz a la cual pertenecen tanto la pelota como el oscilador, puesto que su siguiente estado cambia como resultado de su estado anterior y su interacción con otros elementos.

8) **Vector2D.java**
   Clase que establece lo necesario para trabajar y operar con vectores en 2 dimensiones. En esta clase se establece la suma, la multiplicación por escalar y la resta de vectores.

9) **Ball.java**
   Es un elemento físico, extiende de la clase **PhysicsElement** e implementa **Simulateable** y **Attachable.** En esta parte tenemos dos pelotas, una conectada a un resorte y un amortiguador, y la otra al mismo amortiguador pero por el otro lado

10) **BallView.java**
    Esta clase se encarga de generar la vista de las pelotas, y dibujarla propiamente tal, considera valores como el color, el radio y la posición para ubicarlas en el mundo virtual generado.

11) **Spring.java**
    Elemento físico y, por tanto extiende de la clase **PhysicsElement** e implementa **Connector.** En esta tercera etapa se encuentra conectado a una pelota y a un oscilador, el oscilador va a afectar en la compresión/estiramiento del resorte, que a su vez afectará en la fuerza que éste ejerza sobre la pelota.

12) **SpringView.java**
    Clase que se encarga de generar la vista del resorte, su dibujo para que aparezca en la simulación y sus diferentes formas a medida que se estira y se encoge. Para ello establece los determinados puntos para dibujar las líneas del resorte, usa el color y traza las líneas y las formas a medida que cambia su forma.

13) **Oscillator.java**
    Nuevo elemento físico de esta tercera etapa, se comporta de la misma forma que una pelota hereda de **PhysicsElement** e implementa de la interfaz **Attachable** y **Simulateable**, lo que quiere decir, que se puede conectar a un **Spring** o a un **Dashpot.**

14) **OscillatorView.java**
    Esta clase define la vista que tendrá nuestro oscilador, la forma y los colores que podremos ver en la representación gráfica, también tiene un “radio”, con éste, y la posición se puede ubicar en el plano.

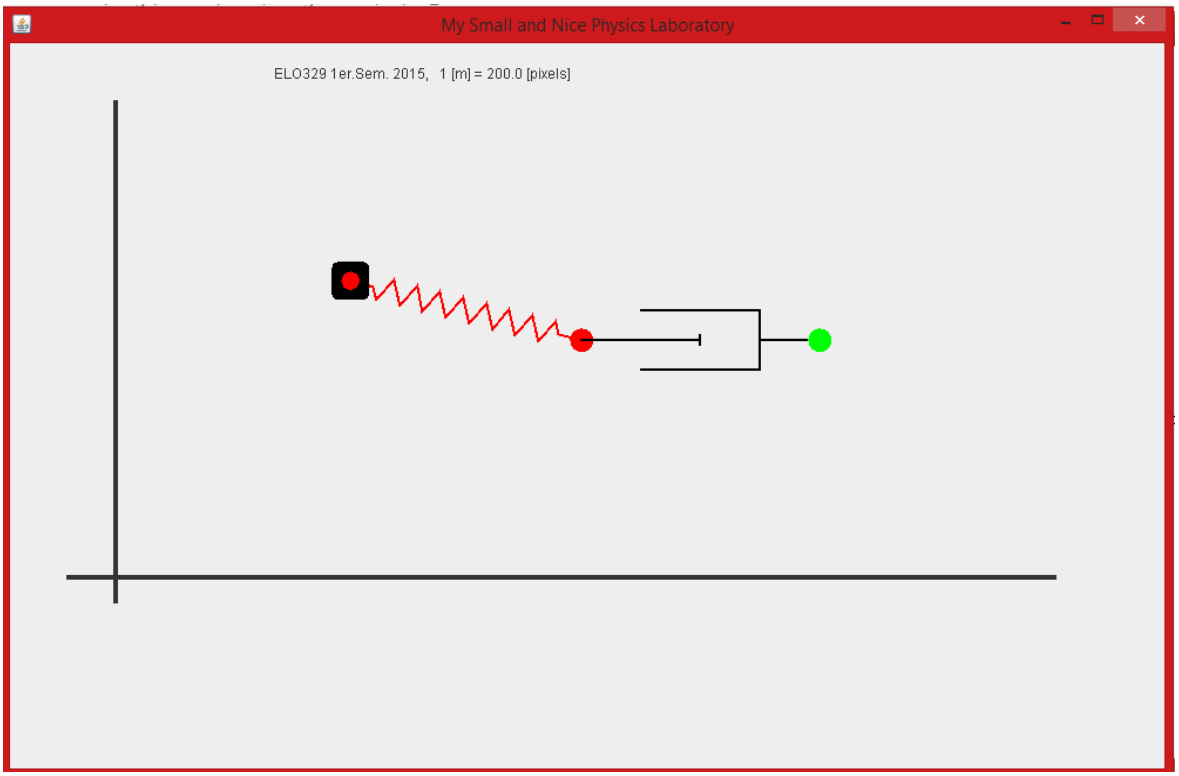
15) Dashpot.java

Nuevo elemento físico, por tanto extiende de **PhysicsElements**, además las características de un **Connector** al igual que el resorte, de hecho se comporta de la misma forma. Codificamos **attachEnd** para unirlo a pelotas, como en este caso, o eventualmente a un oscilador.

16) DashpotView.java

Al igual que los demás View, se encarga de dibujar y permitir la vista del amortiguador en nuestro programa. Que se verá afectado en su tamaño dependiendo del contacto que tenga los demás elementos físicos

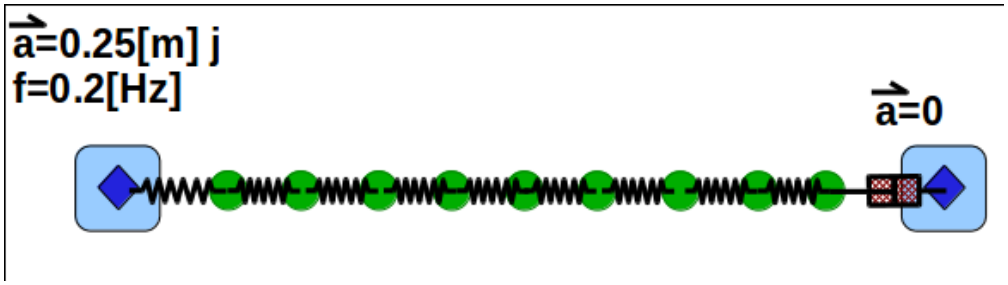
Imagen ejecución del programa:



Parte 4 + Extra Credit

Descripción:

Parte cuatro de la tarea, en esta etapa tendremos exactamente los mismos archivos que en la parte 4. En este caso específico debemos incorporar un menú con sólo un ítem (Configuration) asociado a una opción del menú (Insert), la idea es asemejar la configuración a una cuerda con un oscilador en uno de sus extremos. El otro extremo tiene un amortiguado sujeto de un lado a un punto fijo. Además de eso está la opción del menú ("Extra Credit"), que permite acceder a la parte extra de la tarea, donde tenemos un oscilador conectado a un elástico y éste, conectado a una pelota. Sólo el oscilador que tiene velocidad inicial en el eje x.



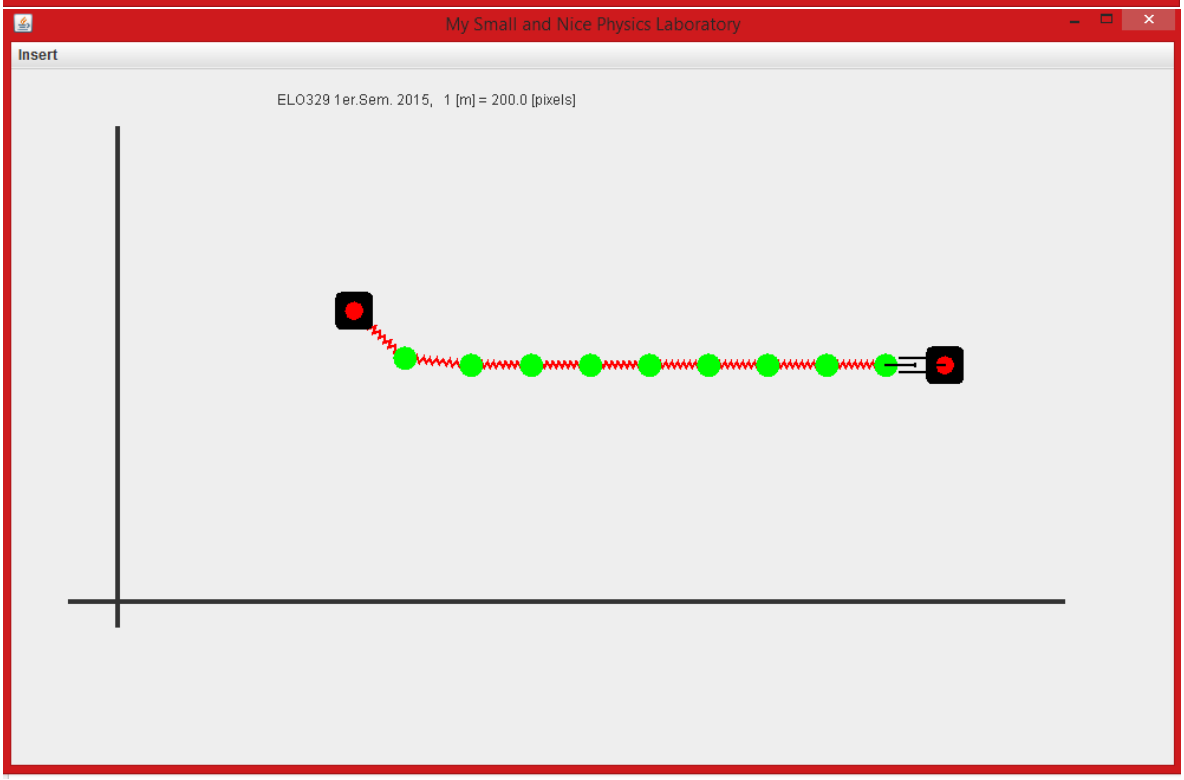
Archivos:

(Al ser los mismos archivos que en la etapa tres, sólo definiremos las clases nuevas que aparecen en el "Extra Credit" de la tarea, que son Elastic y ElasticView)

1) **Elastic.java**
   Elemento físico perteneciente al "Extra Credit" de la tarea. Extiende de **PhysicsElements** y al tener características comunes con el resorte, implementa también **Connector.**

2) **ElasticView.java**
   Se encarga de generar la vista de nuestro elemento físico. Está representado por una línea horizontal, que se alarga y se encoge imitando el comportamiento de un elástico.

Imagen ejecución del programa:



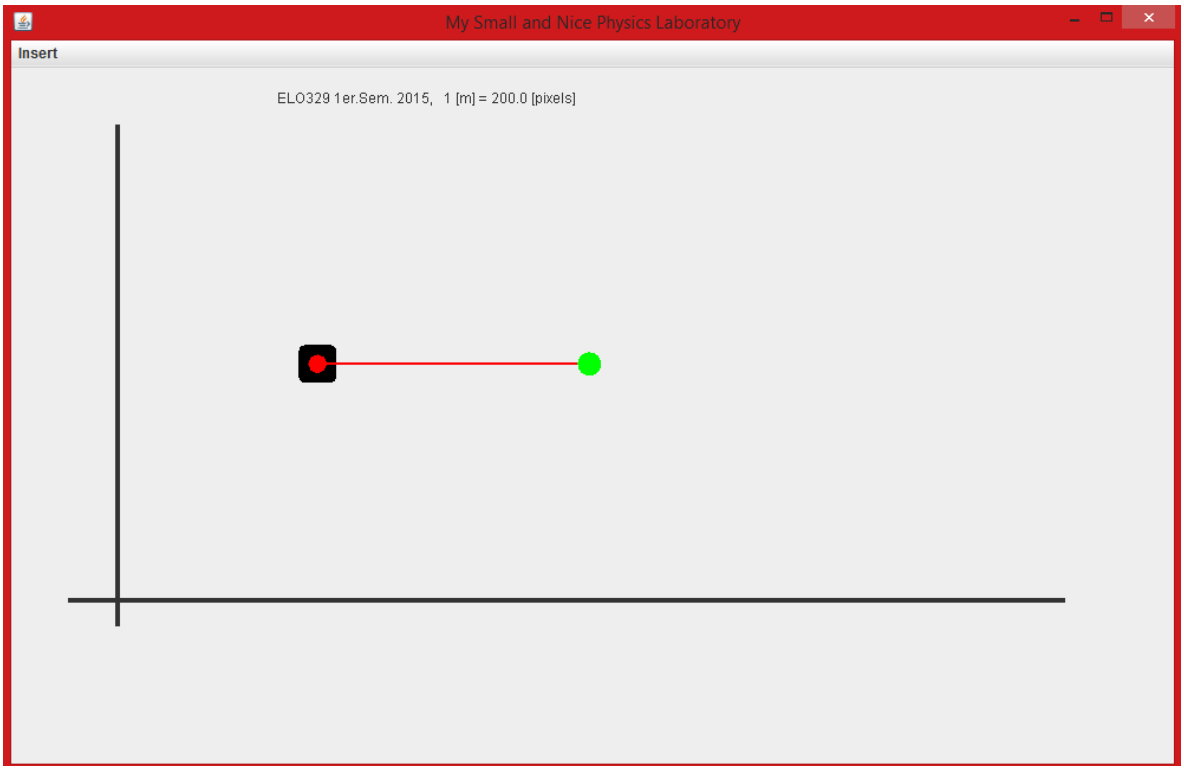


Diagrama de clases: