UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE

INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROGRAMACIÓN II

BLOQUE I de prácticas

Prácticas 3 y 4

Semestre de primavera curso 2013/14

INDICE

Introducción

A lo largo de las prácticas 3 y 4 del bloque I y las prácticas 5 y 6 del bloque II de prácticas de esta asignatura, codificaremos una versión simplificada del juego Arkanoid, de arcade, que fue desarrollado originalmente en 1986.

El juego consiste en mantener una bola dentro de un área de juego delimitada por tres muros estáticos (izquierda, superior y derecha), y por un elemento con movimiento horizontal llamado pala, que es movido por el jugador para evitar que la bola salga del área de juego por la parte inferior. La bola está en continuo movimiento, rebotando en los muros y en la pala y variando su trayectoria en cada rebote. En el área de juego hay bloques llamados ladrillos que desaparecerán al ser tocados una cierta cantidad de veces por la bola y que también provocan cambios en su trayectoria al rebotar la bola en ellos.

El juego finaliza cuando no haya ningún ladrillo en el área de juego o cuando la bola haya salido del área de juego (por la parte inferior).

A continuación se muestra una pantalla de la **aplicación completa** a desarrollar donde todas las columnas de ladrillos son estáticas (Figura 1), donde el juego ha finalizado y se ha derribado un único ladrillo. Se observan tres muros de color negro en los lados izquierdo, superior y derecho del área de juego, la pala de color amarillo situada en la parte inferior de dicha área, una serie de ladrillos estáticos de varios colores relacionados con el número de impactos necesarios para ser derribados. Los ladrillos de color rojo requieren 3 impactos, los de color amarillo requieren 2 impactos y los de color verde sólo 1 impacto para ser derribados, y la bola que ha salido de la zona de juego por el lado inferior-izquierdo:

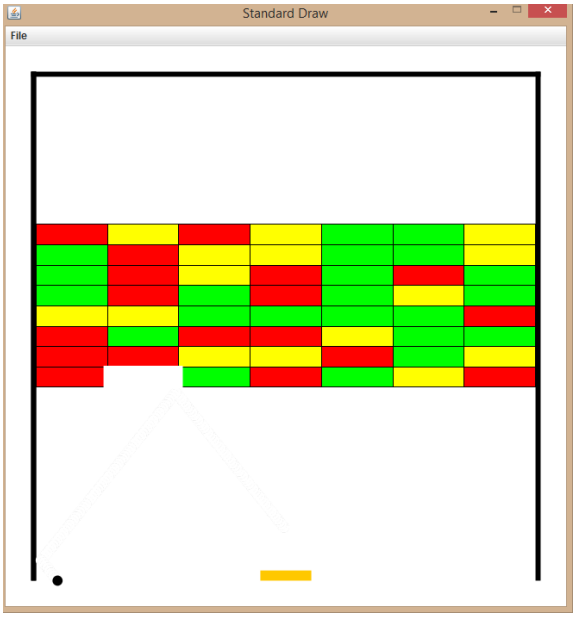

Figura 1

En la figura 2 se muestra una pantalla de la **aplicación completa** donde hay columnas de ladrillos estáticos y columnas de ladrillos dinámicos (señalados por las flechas) con un movimiento oscilatorio horizontal. La dinámica del juego no cambia respecto a la versión anterior. Los ladrillos dinámicos son una parte opcional de la práctica.

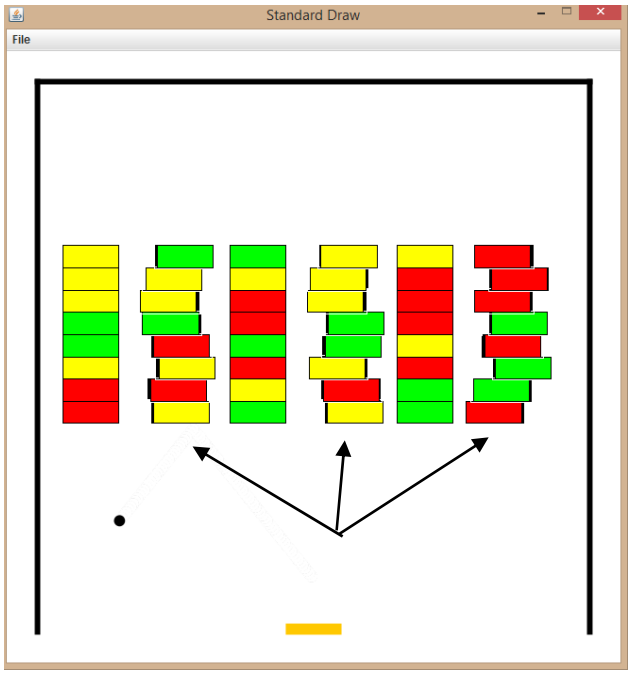

Figura 2

Entorno de desarrollo

Clase Vector

Para representar diversas magnitudes físicas en el juego, tales como la velocidad de la bola, la posición y las dimensiones de los diversos elementos se utilizarán vectores. En esta práctica todos los vectores serán de dos dimensiones.

Para manejar los vectores se utilizará la clase `Vector`. Esta clase se entrega completamente codificada al alumno y permite manejar vectores de un número arbitrario de dimensiones y realizar algunas operaciones básicas con ellos, tales como suma, resta, producto escalar, giro, obtención del módulo, etc. Obsérvese que para esta práctica solo será necesario utilizar un subconjunto de la funcionalidad de esta clase.

La clase `Vector` se encuentra disponible en Moodle y únicamente tiene que añadirla a su proyecto como un fichero de código fuente más.

Clase StdDraw

Para el desarrollo de estas prácticas se utilizará la clase `StdDraw`, que puede ser descargada desde http://introcs.cs.princeton.edu/java/stdlib/StdDraw.java.html. Esta clase proporciona métodos para crear y pintar en una ventana puntos, líneas rectas y curvas, y figuras geométricas, como por ejemplo círculos o rectángulos, y salvar, en un fichero, los elementos pintados. Esta clase se tiene que añadir al proyecto como un fichero de código fuente más.

Práctica 3

Los objetivos que se pretenden cubrir con el desarrollo de esta práctica son los siguientes:

1. Familiarizarse con un entorno integrado de desarrollo de programas en Java de uso común.
2. Desarrollar una aplicación en lenguaje Java, de dificultad baja, aplicando los conceptos básicos de programación orientada a objetos.
3. Crear un proyecto de programación en un IDE.
4. Trabajar con la clase `StdDraw`.
5. Documentar código fuente en Java con Javadoc.

Prerrequisitos

Haber realizado las prácticas 1 y 2 de esta asignatura.

Revisar la documentación de la clase `StdDraw`, que está accesible a través del URL http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html. En particular, consulte los métodos `filledCircle`, `filledRectangle`, `setCanvasSize`, `setPenColor` y `show`, ya que son los que necesitará utilizar en estas prácticas.

Revisar la documentación de la clase Vector para conocer los métodos que implementa (disponible en Moodle). En particular, consulte el constructor y los métodos `plus`, `minus`, `cartesian`, `times`, `angle` y `rotate`.

Trabajo a realizar

Durante esta sesión, el alumno deberá codificar una aplicación que cree el área de juego y sus muros delimitadores (izquierda, superior y derecha), donde además debe incluir otros muros dentro del área de juego a elección por parte del alumno, de forma similar a lo que se muestra en la Figura 3.

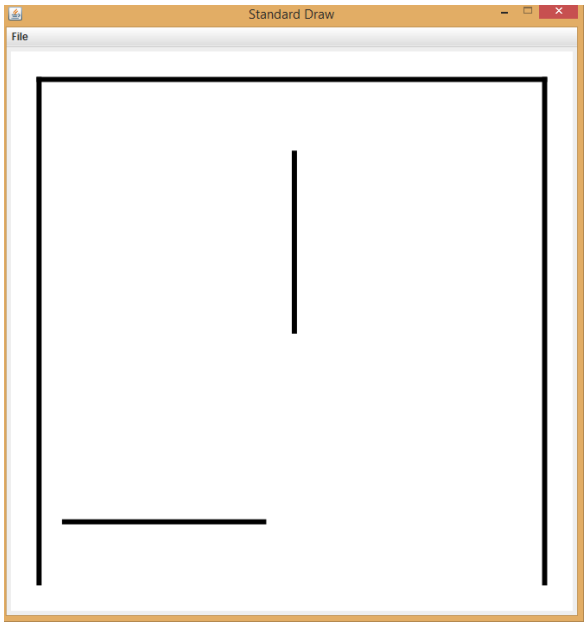

Figura 3.

Desarrollo de la sesión de trabajo

Diseño y codificación de una aplicación desarrollada en entorno gráfico.

Objetivos: codificar una aplicación Java con varias clases utilizando una biblioteca gráfica.

Descripción: el alumno tiene que familiarizarse con las ideas básicas de la biblioteca StdDraw y generar la documentación del programa con la herramienta javadoc. Así mismo, tiene que editar, compilar y ejecutar una aplicación que permita visualizar los elementos gráficos que aparecen en la figura 3, y cuya descripción UML se muestra a continuación (Figura 4):

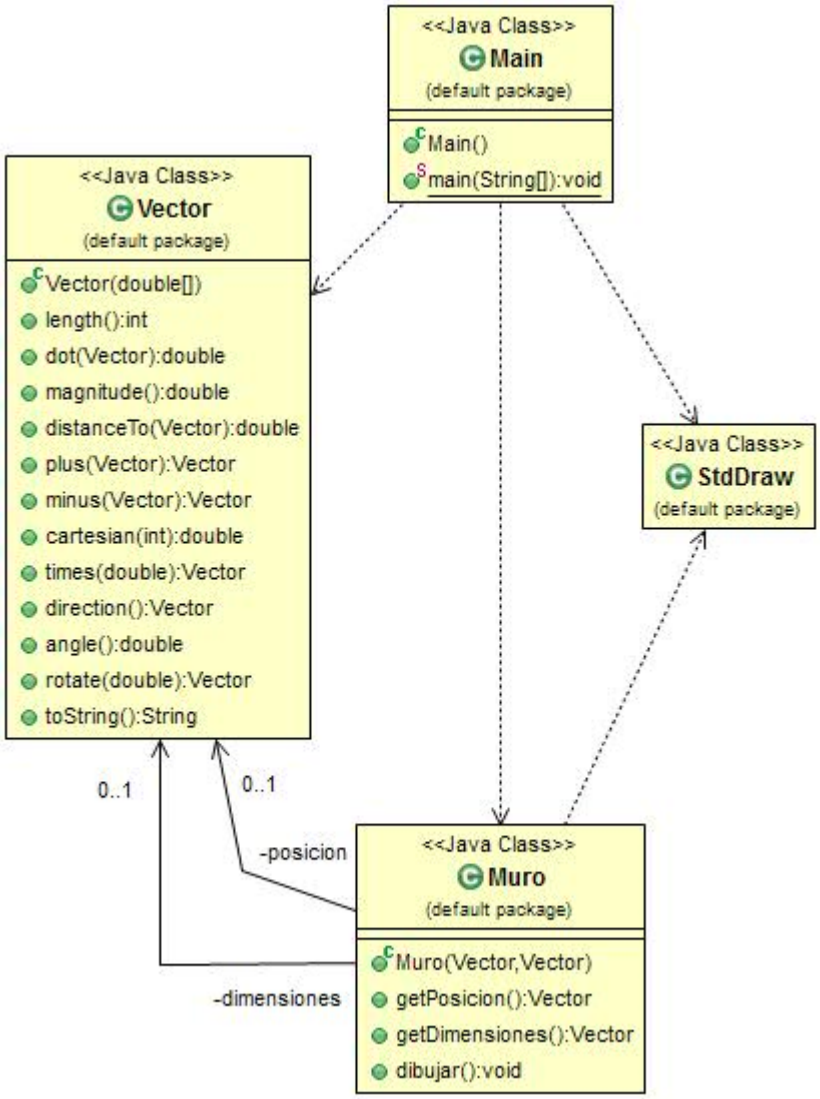

Figura 4

Para conseguir todo lo anterior, se organizará el trabajo en las siguientes fases:

Fase 1 (30 minutos, en el aula de laboratorio):

En esta fase, el alumno aprenderá a dibujar elementos gráficos con la clase `StdDraw`.

1. Cree un proyecto nuevo en Eclipse e incorpore la clase StdDraw.
2. Cree una clase llamada `AplicacionGrafica` con el método `main` y copie en él el siguiente código:

```
 public static void main(String[] args) {
   StdDraw.setCanvasSize(600, 600);
   StdDraw.square(.2, .8, .1);
   StdDraw.filledSquare(.8, .8, .2);
   StdDraw.circle(.8, .2, .2);

   StdDraw.setPenColor(StdDraw.BOOK_RED);
   StdDraw.setPenRadius(.02);
   StdDraw.arc(.8, .2, .1, 200, 45);

   // draw a blue diamond
   StdDraw.setPenRadius();
   StdDraw.setPenColor(StdDraw.BOOK_BLUE);
   double[] x = { .1, .2, .3, .2 };
   double[] y = { .2, .3, .2, .1 };
   StdDraw.filledPolygon(x, y);

   // text
   StdDraw.setPenColor(StdDraw.BLACK);
   StdDraw.text(0.2, 0.5, "black text");
   StdDraw.setPenColor(StdDraw.WHITE);
   StdDraw.text(0.8, 0.8, "white text");
}
```

3. Ejecute el programa y observe el resultado. Debería salir una ventana con varias formas gráficas. Observe que para especificar la posición y el tamaño de las formas gráficas se utiliza un espacio de coordenadas cartesianas entre (0,0) y (1,1), donde (0,0) es la esquina inferior izquierda de la ventana y (1,1) es la esquina superior derecha.

4. Haga que el programa anterior también dibuje un rectángulo relleno de color verde en la parte central de la ventana. Si lo desea, puede probar a dibujar otras formas gráficas.

Fase 2 (80 minutos, en el aula de laboratorio):

En esta fase el alumno tendrá que completar los elementos que aparecen en la figura 3.

1. Cree un proyecto de programación en la plataforma Eclipse.

2. Incorpore la clase Vector que está disponible en Moodle.

3. Codifique la clase `Muro`, que representa a los diferentes elementos que se visualizan en la Figura 3. Cada muro se representa mediante un rectángulo sólido (con relleno). Las dimensiones de un rectángulo pueden determinarse por su semialtura y su semianchura. Para situarlo en el plano puede usarse el vector que va desde el origen de coordenadas hasta su centro. En esta práctica, todos los rectángulos tendrán sus lados paralelos a los ejes de coordenadas. Por tanto, podremos representarlos mediante el vector que indica la posición de su centro, y el vector que va desde el centro al vértice superior derecho, tal como ilustra la figura 5. Los muros se mostrarán en la ventana mediante el método `dibujar()`. Se facilita el javadoc de esta clase en Moodle.

4. Codifique la clase `Main` que crea el área de juego y dibuja los diferentes elementos que aparecen en la Figura 3. Para ello, creará tantos objetos Muro como sean necesarios y los

guardará en una tabla de tipo `Muro[]`. El programa principal los dibujará recorriendo dicha tabla y llamando al método `dibujar()` de cada elemento de la tabla.

5. Genere la documentación completa de la aplicación con la herramienta javadoc.

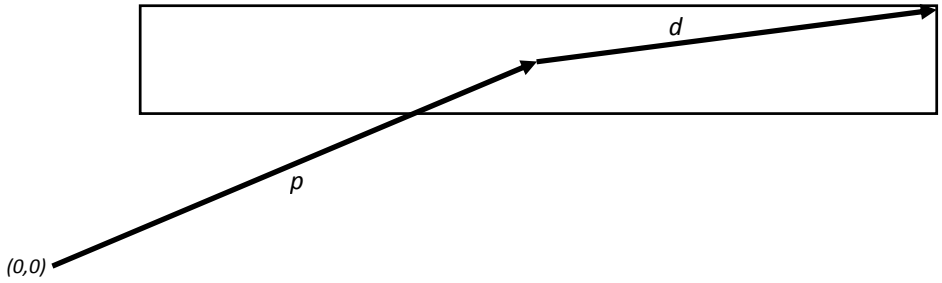

Figura 5

Plazos de realización, entrega y documentación a entregar

La práctica 3 se realizará en la semana lectiva 6 que comienza el 14 de marzo y finaliza el 20 de marzo.

Los archivos que son solución de la práctica 3 de este bloque de prácticas deberán entregarse a través de la plataforma Moodle de la siguiente manera:

1. El código fuente de las clases `Main (Main.java) y Muro (Muro.java)` debe estar ubicado en una carpeta llamada `src`.
2. La documentación generada por javadoc debe estar ubicada en la carpeta doc
3. Se deben comprimir las carpetas `src` y doc en un único fichero ZIP.
4. El nombre del fichero ZIP debe tener el formato:

   grupolaboratorio_B1_P3_apellidos_nombre.zip

   Por ejemplo, un alumno que se llamara Rafa Nadal y que estuviera en el grupo V4 subiría a la plataforma Moodle el fichero *V4_B1_P3_Nadal_Rafa.zip*
5. El fichero resultante del paso anterior será subido a la plataforma moodle. Sólo se admitirá un único fichero.
6. El plazo de entrega terminará 96 horas después de finalizar su grupo de laboratorio. Por ejemplo, para un alumno del grupo V4 que se imparte los viernes de 15:30 a 17:30 finaliza el martes a las 17:30. Tenga en cuenta que la plataforma Moodle registra la hora de entrega.

Estas reglas deben respetarse **obligatoriamente** para que las prácticas sean evaluadas.

Práctica 4

Los objetivos que se pretenden cubrir con el desarrollo de esta práctica son los siguientes:

1. Familiarizarse con un entorno integrado de desarrollo de programas en Java de uso común.
2. Desarrollar una aplicación en lenguaje Java, de dificultad baja, aplicando los conceptos básicos de programación orientada a objetos.
3. Crear un proyecto de programación en un IDE.

4. Trabajar con la clase `StdDraw`.
5. Documentar código fuente en Java con Javadoc.

Prerrequisitos

Haber realizado la práctica 3 de esta asignatura.

Revisar la documentación de la clase `StdDraw`, que está accesible a través del URL http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html. En particular, consulte los métodos , `filledCircle`, `filledRectangle`, `setCanvasSize`, `setPenColor` y `show`, ya que son los que necesitará utilizar en estas prácticas.

Revisar la documentación de la clase Vector para conocer los métodos que implementa (disponible en Moodle). En particular, consulte el constructor y los métodos `plus`, `minus`, `cartesian`, `times`, `angle` y `rotate`.

Trabajo a realizar

Durante esta sesión, el alumno deberá codificar una aplicación que utilice el área de juego creado en la práctica 3 (ver figura 3) y cree una bola que se moverá por el área de juego rebotando en todos los elementos existentes, tal como se muestra en la Figura 5.

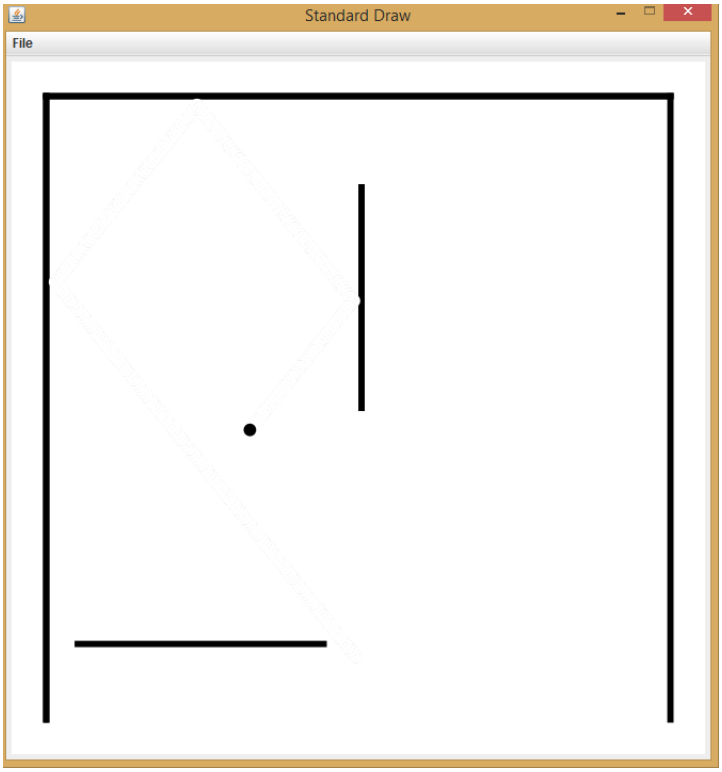

Figura 5.

En esta aplicación, una vez se ha dibujado el área de juego, habrá que crear una bola y lanzarla en una dirección cualquiera, con una cierta velocidad inicial. A continuación, un bucle del programa principal simulará el paso del tiempo, de forma que en cada iteración se considerará que ha transcurrido un cierto tiempo y se calculará la nueva posición de la bola en función de su velocidad y el tiempo transcurrido de una iteración a la siguiente. Tanto la posición de la bola como su velocidad se pueden representar

mediante sendos vectores. Como la bola se mueve, al dibujarla habrá que borrarla de la posición anterior y dibujarla en la nueva posición. Para borrar la bola es suficiente con volver a pintarla en la posición anterior con el color del área de juego.

Además, tras cada movimiento de la bola, el programa deberá comprobar si ha chocado contra algún muro, en cuyo caso deberá rebotar y cambiar su trayectoria. En este juego se considerará que los muros son elementos inmóviles, que todos los elementos son cuerpos rígidos y que los choques son perfectamente elásticos, por lo que en cada impacto entre la bola y un muro se producirá una reflexión y la bola mantendrá la magnitud de su velocidad, aunque la trayectoria se modificará según las leyes básicas de la reflexión. Para calcular el ángulo de reflexión, lo más sencillo puede ser trabajar con la velocidad como un vector en coordenadas polares. En este caso, tendremos que el módulo del vector nos indica la magnitud de la velocidad de la bola y el ángulo nos indica la dirección en la que se mueve. Trabajando con coordenadas polares, los giros se pueden hacer sumando un ángulo al ángulo original correspondiente a la velocidad.

Desarrollo de la sesión de trabajo

Diseño y codificación de una aplicación desarrollada en entorno gráfico.

Objetivos: codificar una aplicación Java con varias clases utilizando una biblioteca gráfica.

Descripción: el alumno tiene que familiarizarse con las ideas básicas de la biblioteca StdDraw y generar la documentación del programa con la herramienta javadoc. Así mismo, tiene que editar, compilar y ejecutar una aplicación que permita visualizar los elementos gráficos que aparecen en la figura 6, y cuya descripción UML se muestra a continuación (Figura 6):

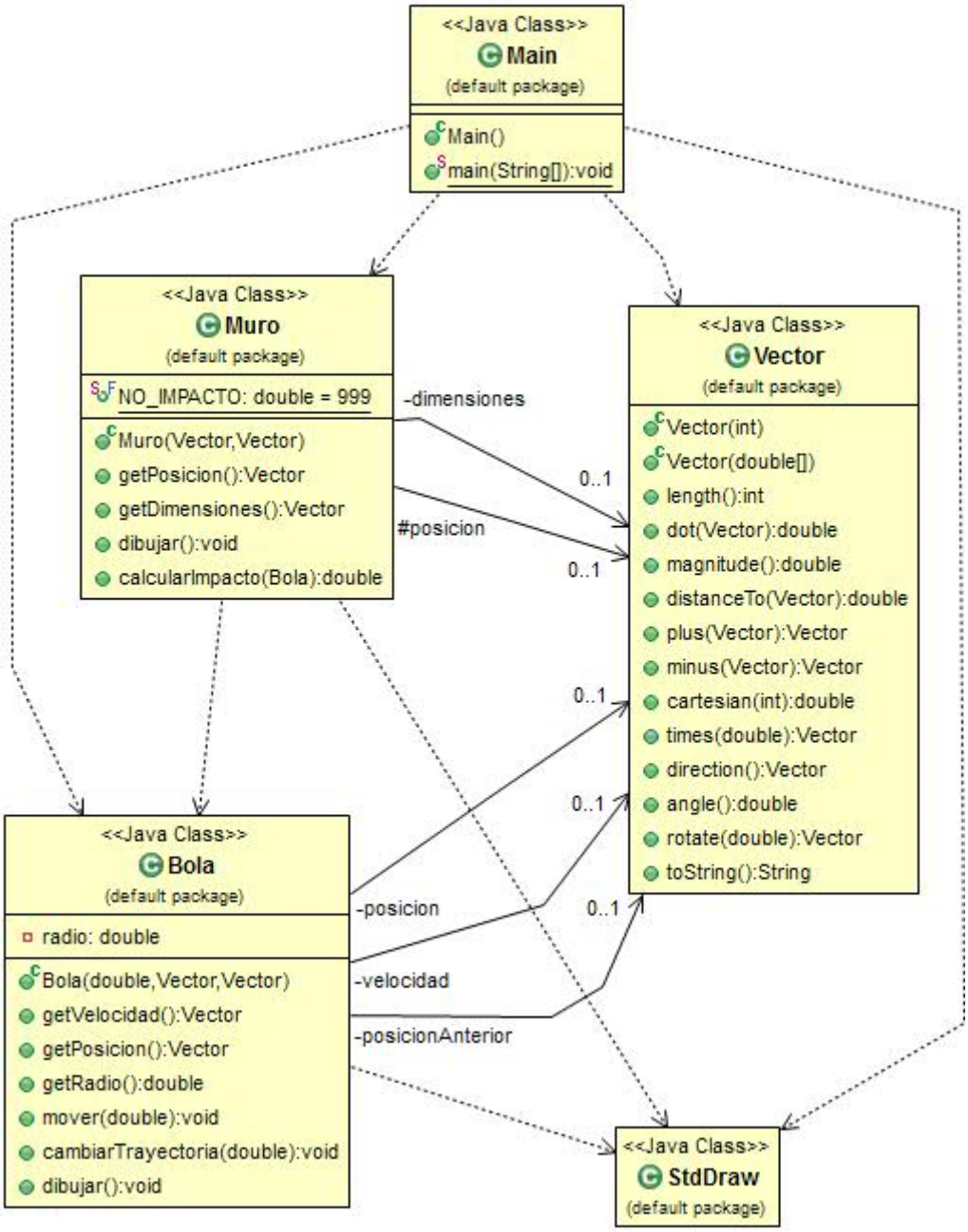

Figura 6

Para conseguir todo lo anterior, se organizará el trabajo en las siguientes fases:

Fase 1 (45 minutos, en el laboratorio)

En esta fase el alumno creará la clase Bola y aprenderá a moverla por la pantalla, aunque de momento sin que rebote en los muros. Para realizar esta fase se parte del programa desarrollado en la práctica anterior.

1. Cree una clase nueva llamada Bola con los métodos descritos en el Javadoc que se encuentra en Moodle, salvo el método `cambiarTrayectoria()`, que se añadirá un poco más adelante. Algunas recomendaciones son:

   - El método `mover()` debe guardar la posición de la bola antes del movimiento (para que pueda ser utilizada por el método `dibujar()`) y calcular la nueva posición tras el intervalo de tiempo especificado. A continuación se muestra una posible implementación del método, en la que se supone que la posición y velocidad actuales de la bola se encuentran almacenados en los atributos `posicion` y `velocidad` y que el atributo `posicionAnterior` guarda la posición anterior de la bola:

     ```
     public void mover(double dt) {
         posicionAnterior = posicion;
         posicion = posicion.plus(velocidad.times(dt));
     }
     ```

   - El método `dibujar()` primero debe borrar la posición anterior de la bola pintándola en el color del área de juego y después pintar la posición actual de la bola. Así se obtiene el efecto de movimiento. La bola se puede dibujar mediante un círculo (sólido).
   - Cuando ejecute el programa final, seguramente observará que la bola va dejando un rastro tras de sí. Puede eliminarlo haciendo que el borrado de la posición anterior pinte una bola un poco más grande de lo normal.

2. Modifique el programa principal para que, después de dibujar los muros, se cree un objeto Bola con una posición inicial en la zona central inferior de la ventana y una velocidad inicial de, por ejemplo, (-0.04, 0.05). A continuación, el programa entrará en un bucle en que se llame repetidamente a los métodos `mover()` y `dibujar()` de la bola. Se recomienda utilizar un valor en torno a 0,1 como intervalo de tiempo para el método `mover()`. El bucle terminará cuando la bola se salga del área de juego. En adelante, denominaremos a este bucle "bucle principal del juego". Cuando ejecute este programa, debería observar una bola que se mueve rectilíneamente por la pantalla, atravesando los muros, hasta que el programa se detiene cuando la bola alcanza algún extremo de la ventana.

Fase 2 (65 minutos, en el laboratorio)

En esta fase el alumno añadirá el efecto de rebotes a los muros y a la bola.

1. Añada el método `public double calcularImpacto (Bola bola)` a la clase `Muro`. Este método recibe un objeto Bola y debe hacer dos cosas:

   - Primero debe determinar si la bola ha impactado con el muro, es decir, si la bola se superpone, aunque sea parcialmente, con el área ocupada por el muro.
   - En caso de que haya impacto, debe calcular el ángulo de giro que habrá que aplicar al vector velocidad de la bola para desviar su trayectoria según el impacto.
   - En un apartado posterior se proporciona una posible codificación de este método.

2. Incorpore el método `public void cambiarTrayectoria (double anguloDesvio)` a la clase `Bola`. Este método gira el vector velocidad de la bola según el ángulo especificado. Así cambiará su trayectoria la próxima vez que se mueva.

3. Incorpore al bucle principal del juego un fragmento de código que recorra todos los muros invocando al método `calcularImpacto()` para saber si hay impacto con la bola y, en caso afirmativo, llame al método `cambiarTrayectoria()` de la bola para aplicar el desvío de trayectoria correspondiente.

Cuando ejecute el programa resultante de esta fase, debería observar que ahora la bola rebota en los muros.

NOTA: seguramente observará que los muros quedan "mordidos" en los puntos de impacto con la bola. También es posible que la velocidad de la bola sea demasiado rápida o demasiado lenta. Esto se podrá mejorar en la siguiente fase.

Fase 3 (1 hora, en casa, después de la sesión de laboratorio)

En esta fase, se dan algunas recomendaciones para mejorar ciertos detalles de la presentación gráfica del juego. En este punto conviene avisar de que no es un objetivo de la práctica obtener una presentación gráfica del juego perfecta, ni tampoco una precisión muy elevada en la mecánica del juego ni en la interacción con el usuario. Más allá de lo que se exige o recomienda en el enunciado de la práctica, el alumno puede dedicar todo el tiempo que desee a afinar este tipo de detalles, pero ese tiempo no se contempla dentro de la planificación de la práctica.

1. Para ajustar la velocidad de movimiento de la bola a algo adecuado para un humano se pueden probar varios valores para el intervalo de tiempo `dt` que se pasa al método `mover(double dt)` de la clase `Bola`. También se pueden insertar llamadas a `Thread.sleep()` en puntos estratégicos para que el programa demore su ejecución lo suficiente como para ir a un ritmo adecuado para el jugador humano.

2. A medida que haya más elementos gráficos en el juego, es posible que note que los métodos para dibujar de la clase `StdDraw` son relativamente lentos. Esto se puede corregir con el método `show()`. En concreto, puede deshabilitar la actualización gráfica de la ventana al inicio de cada iteración del bucle principal del juego llamando a `StdDraw.show(0)` y puede activarla de nuevo para que se actualice la ventana de golpe al final de cada iteración del bucle principal del juego llamando a `StdDraw.show(1)`.

3. Para eliminar o reducir los efectos de "mordisco" en los muros tras el impacto de la bola, se puede redibujar el muro en el que ha habido impacto. Tenga en cuenta que puede suceder que después de redibujar el muro vuelva a producirse el "mordisco" cuando la bola se mueva y borre la posición anterior. Otra posibilidad sería redibujar todos los muros en todas las iteraciones del bucle principal del programa, pero en ese caso se incrementaría sustancialmente el consumo de recursos de CPU por parte del juego. También hay otras posibilidades que el alumno puede plantearse, como hacer los movimientos y repintados en varias fases, pero se repite que no es un objetivo de la práctica obtener una presentación gráfica perfecta, por lo que no es imprescindible maquillar este aspecto de la presentación gráfica.

4. Si lo desea, opcionalmente puede añadir todos los detalles gráficos que desee a los elementos del juego (tramas, colores, imágenes…), aunque esto en principio no se tendrá en cuenta de cara a la evaluación de la práctica.

Detección del impacto de la bola contra los muros

A continuación se propone una forma de detectar cuándo impacta la bola contra un muro y por qué lado del muro lo hace.

Sea un muro y el vector d de sus dimensiones (este vector tiene su origen en el centro del rectángulo y su extremo en la esquina superior derecha) indicados en la figura 7. Si tomamos el ángulo α del vector d en coordenadas polares, podemos usarlo para dividir el espacio alrededor del muro en cuatro zonas, que corresponden a arriba, izquierda, abajo y derecha. Por ejemplo, todos los vectores originados en el centro del muro y con un ángulo en el intervalo $[\pi/2+\alpha, 2\pi-\alpha)$ estarían en la zona de abajo.

Si calculamos el vector a, como el que va del centro del muro al centro de la bola, entonces podemos comparar el ángulo en polares del vector a con los cuatro intervalos antedichos, y así determinar en qué zona se encuentra la bola.

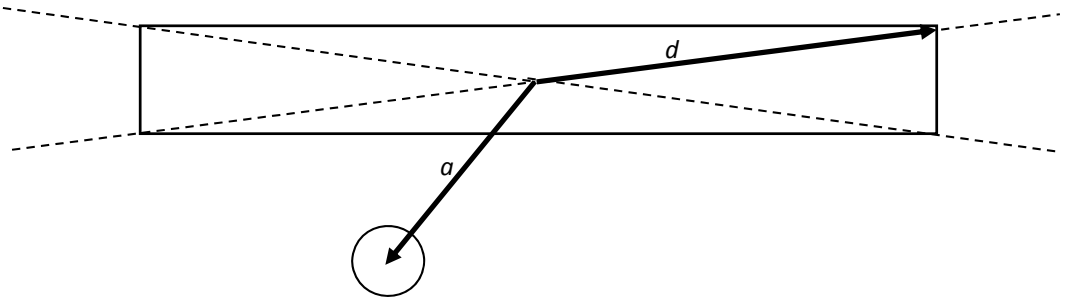

Figura 7

Para saber además si la bola está en impacto con el muro se deben comparar las posiciones cartesianas de los extremos de la bola con los extremos del muro. Sea (x_b, y_b) la posición de la bola en coordenadas cartesianas y sea r su radio, sea (x_m, y_m) el centro del muro y (x_d, y_d) las dimensiones del muro (semianchura, semialtura). Así, si la bola está en la zona de abajo, entonces hay impacto si se cumplen simultáneamente las tres siguientes condiciones:

$$y_b + r \geq y_m - y_d$$
$$x_b - r/2 \geq x_m - x_d$$
$$x_b + r/2 \leq x_m + x_d$$

La primera condición comprueba si el extremo superior de la bola alcanza o sobrepasa el extremo inferior del muro y las otras dos condiciones comprueban que el grueso de la bola se encuentre entre los dos extremos laterales del muro. Se divide el radio por 2 para no considerar impacto cuando la bola roza la esquina del muro.

Para el resto de las zonas, las condiciones serían análogas.

En el Moodle dispone de una implementación del método calcularImpacto que sigue el esquema descrito.

Plazos de realización, entrega y documentación a entregar

La práctica 4 se realizará en la semana lectiva 7 que comienza el 21 de marzo y finaliza el 27 de marzo.

Los archivos (todos) que son solución de la práctica 4 en este bloque de prácticas deberán entregarse a través de la plataforma Moodle de la siguiente manera:

1. El código fuente de las clases `Main` (`Main.java`), `Muro` (`Muro.java`) y `Bola` (`Bola.java`) debe estar ubicado en una carpeta llamada `src`.
2. La documentación generada por javadoc debe estar ubicada en la carpeta doc.
3. Las carpetas `src` y doc deben comprimirse en un único fichero ZIP.
4. El nombre del fichero ZIP debe tener el formato:

   grupolaboratorio_B1_P4_apellidos_nombre.zip

   Por ejemplo, un alumno que se llamara Rafa Nadal y que estuviera en el grupo V4 subiría a la plataforma Moodle el fichero *V4_B1_P4_Nadal_Rafa.zip*

5. El fichero resultante del paso anterior será subido a la plataforma moodle. Sólo se admitirá un único fichero.
6. El plazo de entrega finalizará 96 horas después de finalizar su grupo de laboratorio. Por ejemplo, para un alumno del grupo V4 que se imparte los viernes de 15:30 a 17:30 finaliza el martes a las 17:30. **Tenga en cuenta que la plataforma Moodle registra la hora de entrega.**

Estas reglas deben respetarse **obligatoriamente** para que las prácticas sean evaluadas.

Evaluación

Las prácticas 3 y 4 serán evaluadas conjuntamente con el resto de prácticas del bloque I al finalizar el mismo. **Esta evaluación se realizará en la semana 8 del 28 de marzo al 3 de abril.**

- El peso total de las prácticas 3 y 4 del bloque I, en relación a la nota final del bloque, se detalla a continuación.

Hito de evaluación	Porcentaje en el peso del bloque I
Práctica 3	30%
Práctica 4	30%