Prácticas de Programación Orientada a Objetos
Convocatoria de Febrero
Curso 2009/2010

Entrega 2

Objetivos formativos

- Objetivos formativos de la entrega 1.
- Identificación de clases, junto a sus atributos, métodos y constructores, así como las relaciones entre ellas.
- Comprender la utilidad de las clases abstractas como mecanismo de clasificación y reutilización de código.
- Aplicar la técnica del Diseño por Contrato en la definición de una clase. Utilizar correctamente excepciones *runtime* como mecanismo de control de las precondiciones.
- Incluir el control de estado en la definición de una clase. Utilizar correctamente excepciones *runtime* como mecanismo de control.
- Uso de las colecciones de Java.
- Definición de una jerarquía de herencia.
- Aplicación del concepto de interface Java.
- Comprender el código desarrollado por otros y utilizarlo dentro de una aplicación.
- Entender el propósito del mecanismo de excepciones y utilizarlo correctamente.
- Aplicar correctamente los patrones de diseño Estrategia, Composición y Método Plantilla.

Plazo de entrega: 15 de enero de 2010.

Valoración de la práctica: 60% de la nota final.

Diseño por Contrato

Aplica la técnica Diseño por Contrato al código de toda la práctica (entregas 1 y 2). En primer lugar, identifica y documenta en `JavaDoc` todas las precondiciones (estado y parámetros) de los métodos y constructores. A continuación utiliza las excepciones *runtime* de Java para notificar la violación de las precondiciones. En segundo lugar, el código que haga uso de métodos y constructores con precondiciones (rol cliente en el Diseño por Vontrato) ha de asegurar que se cumplen las precondiciones antes de hacer la llamada. Si el cliente no pudiera hacer cumplir una precondición debe tratar esta situación de forma adecuada. En general, sigue las recomendaciones del Tema 4 y Seminario 4.

Pruebas unitarias `JUnit`

En esta entrega no hay que implementar pruebas `JUnit`, ya que a través de la interfaz de usuario se podrá validar *visualmente* la funcionalidad de la aplicación.

Filtro de elementos

Añade a la clase **`Planta`** un método que permita consultar todos los elementos que cumplan una condición cualquiera. Ejemplos de condiciones podrían ser: elementos que se encuentren en casillas visibles, elementos de tipo autónomo, armas, etc. Utiliza ese método en el resto del código cuando sea necesario filtrar los elementos de una planta.

Elementos

A continuación se describen las características y funcionalidad de los elementos del juego.

1. Los elementos del juego se sitúan sobre las casillas de las plantas. En un momento determinado pueden estar sólo en una casilla o en ninguna.
2. La casilla sobre la que se sitúa un elemento puede ser consultada y modificada. Nótese que entre las clases Casilla y Elemento hay una relación de clientela bidireccional.
3. Sobre cualquier elemento podemos consultar la planta en la que se encuentra y las coordenadas x e y de la casilla sobre la que está situado.
4. A un elemento le pueden suceder tres situaciones destacables durante el transcurso del juego que determinan su estado (véase el punto siguiente):
    a. Escapar: un elemento escapa si consigue subir desde la última planta. Un elemento que escapa logra el éxito en el juego.
    b. Fracasar: un elemento queda destruido si cae desde la primera planta. Un elemento en esta situación ha fracasado en el juego.
    c. Desaparecer: ha sido desplazado de una casilla por otro elemento. En este caso el elemento desaparece del juego.

5. Un elemento puede encontrarse en uno de los siguientes estados:
   a. Estado “no activo” si aún no ha sido asignado a una casilla.
   b. Estado “activo” si tiene una casilla asociada.
   c. Estado “desaparecido” si ha sido expulsado de una casilla por otro elemento del juego.
   d. Estado “éxito” si ha escapado del escenario, es decir, ha conseguido situarse en un muelle de la última planta y ha salido del escenario.
   e. Estado “fracaso” si ha caído desde la primera planta por un hueco.
6. El estado de los elementos puede ser consultado.
7. Todos los elementos pueden moverse en alguna dirección. El movimiento traslada al elemento de su casilla actual a la casilla adyacente en la dirección indicada, si la hubiera. Asimismo, también puede moverse directamente a una casilla, siempre que ésta sea adyacente. Es decir, el movimiento en una dirección es equivalente al movimiento a una casilla adyacente.
8. Los elementos pueden determinar qué otro elemento se encuentra más cercano a él a partir de una colección de elementos. Para el cálculo de la distancia más cercana se empleará distancia euclídea. Utiliza las rutinas matemáticas de la clase `Math`. La fórmula se indica a continuación:

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

9. Los elementos pueden calcular qué casilla adyacente los acerca más a una posición determinada de la planta. Esta característica es utilizada para dirigirse a sus objetivos. De nuevo, para realizar este cálculo utiliza distancia euclídea.
10. Existen varios tipos de elementos: autónomos, armas, proyectiles y personajes.
11. Un elemento autónomo se caracteriza por realizar una acción que es solicitada periódicamente por el escenario del juego (véase apartado Escenario).
12. Las armas son elementos que pueden ser utilizados por los personajes. Por tanto, un arma puede tener un propietario que será un personaje. En caso de que tenga propietario, la planta, la casilla y las coordenadas x e y del arma serán las del propietario.
13. Las armas pueden ser activadas. La tarea que realizan en la activación depende del tipo de arma.
14. La antorcha es un arma que puede lanzar proyectiles (bolas de fuego) en una dirección. La dirección en la que lanza proyectiles es siempre fija para una misma antorcha. Cuando es activada sitúa un proyectil en la casilla adyacente a sus coordenadas en la dirección de disparo.
15. Las antorchas también son elementos autónomos. Mientras que están en el escenario sin ser cogidas por un personaje se mueven aleatoriamente a otra posición de la planta. Cuando se solicita que realicen su acción comprueban si ha transcurrido al menos 3 segundos desde el último movimiento. Si es así, la antorcha se mueve a otra casilla que no tenga ningún elemento. Utiliza el método `currentTimeMillis` de la clase `System` para controlar el tiempo transcurrido.
16. Un visor es un tipo de arma que cuando es activada muestra todas las casillas adyacentes a la posición en la que se encuentra.

17. Hay un tipo de arma especial denominada arma compuesta. Cuando un personaje coge esta arma puede ir acumulando cualquier otra arma que vaya encontrando por el camino. Es decir, un arma compuesta es una composición de otras armas. La activación de esta arma consiste en activar las armas de las que está compuesta.
18. Un proyectil es un elemento autónomo que se desplaza en una dirección. Cada vez que se solicita que realice su acción, se mueve una casilla en una dirección fija. Si el movimiento lo lleva a una posición fuera del escenario, el proyectil desaparece. Un proyectil colisiona con otro elemento si se sitúa en una casilla en la que ya hay otro elemento. En caso de colisión desaparecen tanto el elemento que se encontraba en la casilla como el proyectil.
19. Las bolas de fuego que lanzan las antorchas son proyectiles. Se caracterizan por estar limitadas en el número de casillas que pueden avanzar. Cuando agoten el número de movimientos desaparecerán. Este límite está en función de las dimensiones de la planta y la dirección de movimiento. Si la dirección es arriba o abajo, será la mitad del tamaño en Y de la planta, mientras que si la dirección es derecha o izquierda, será la mitad del tamaño en X de la planta.
20. Los personajes son elementos autónomos. Se caracterizan por coger y activar armas del escenario.
21. Cuando un personaje se mueve a una casilla, aparte de las acciones que haría cualquier elemento, comprueba si en la casilla a la que se dirige hay un arma. En este caso, coge el arma y la saca de la casilla.
22. La acción de coger un arma consiste en indicar al arma que el personaje es su propietario. Además, se pueden producir los siguientes casos:
    - El jugador no tiene arma. En este caso simplemente coge el arma.
    - El jugador tiene un arma y coge un arma compuesta. El arma que tenía pasa a formar parte del arma compuesta.
    - El jugador tiene un arma compuesta y coge una nueva. Esta arma pasa a formar parte del arma compuesta.
    - El jugador tiene un arma que no es compuesta y coge una nueva. En tal caso, pierde la que ya tuviera (desaparece) y toma la nueva.
23. El jugador es un tipo de personaje. El jugador puede realizar dos tipos de acciones: moverse a una casilla adyacente o activar su arma. Ninguna de estas dos acciones las realiza por iniciativa propia, sino bajo la indicación del usuario de la aplicación (véase apartado Interfaz Gráfica). El jugador debe recordar qué acción debe realizar para efectuarla cuando deba realizar la acción que caracteriza a los elementos autónomos.
24. Hay un tipo especial de personajes que denominamos enemigos. Cada vez que realizan su acción, intentarán avanzar una casilla hacia el arma que “vean” más cercana. No todos los enemigos tiene la misma visión. Una vez que consigan un arma se quedarán quietos.
25. El enemigo vidente es capaz de ver todas las armas de la planta, aunque estén en casillas ocultas.
26. El enemigo loco sólo puede ver las armas que se sitúan en casillas visibles. Mientras no consigue un arma se comporta como cualquier otro enemigo. Sin embargo, cuando consigue un arma enloquece. En esta situación, cada vez que realiza su acción se mueve aleatoriamente a una casilla adyacente y activa el arma al azar. Es decir, siempre se mueve a una casilla y en algunas ocasiones activa el arma. Es posible que en sus movimientos erráticos pueda coger otras armas.

Escenario

El escenario del juego está formado por varias plantas del mismo tamaño. La funcionalidad básica que ofrece un escenario es la siguiente:

- Construir un escenario a partir del número de plantas y las dimensiones (tamaños en X e Y). El escenario se encarga de construir todas las plantas y conectarlas (relaciones inferior y superior).
- Consultar el número de plantas y las dimensiones en X e Y.
- Reemplazar una casilla existente por otra nueva. Se indicará el índice de la planta en el escenario (la primera planta tiene índice 0) y las coordenadas X e Y dentro de esa planta.
- Situar un elemento en cualquier casilla. Igual que en el caso anterior se hará referencia a la planta donde se sitúa el elemento por índice.
- Establecer y consultar el jugador.
- Consultar la planta activa. Se entiende por planta activa aquella en la que está situado el jugador.
- Consultar el índice de la planta activa.

Un escenario puede estar en varios estados:

- Estado "configuración": se encuentra en este estado tras la construcción. En este estado se pueden situar casillas y elementos en el escenario.
- Estado "completo": se alcanza este estado aplicando el método *setCompleto*. Este método sólo puede aplicarse si el escenario está en "configuración". El jugador sólo se puede establecer en estado "completo".
- Estado "preparado": se alcanza este estado cuando se establece el jugador. A partir de este estado puede ser consultado el jugador.
- Estado "arrancado": el escenario se encuentra en este estado si se aplica el método *arrancar*. Este método sólo puede aplicarse si el escenario está "preparado". La planta activa sólo puede ser consultada en este estado.
- Estado "finalizado": este estado se alcanza desde el estado anterior aplicando el método *finalizar*. El juego finaliza por éxito o fracaso del jugador. Una vez llegado a este estado el escenario no puede volver a utilizarse, es decir, no puede cambiar de estado.

El escenario es el motor del juego. Se encarga de accionar a todos los elementos autónomos y mostrar aleatoriamente algunas casillas de la planta activa. Para realizar esta funcionalidad hará uso de un temporizador (véase apartado Temporizador) con determinado periodo de aviso (propiedad del escenario). Al aplicar el método *arrancar* el escenario pone en marcha un temporizador. El temporizador se detiene cuando se aplica el método *finalizar*.

El temporizador avisa periódicamente al escenario para que anime el juego. La animación de un escenario consiste en las siguientes tareas:

- Recorrer los elementos autónomos de la planta activa y aplicarles un método para que realicen su acción. La aplicación de la acción sobre los elementos autónomos se hará de forma secuencial. De este modo se asegura que sólo un elemento puede realizar su acción en un momento determinado, evitando así problemas de concurrencia. Nótese que sólo es posible aplicar la acción sobre un elemento si está en estado activo (véase apartado Elementos).

- Visualizar casillas. El escenario deja visibles unas cuantas casillas de la planta activa durante el periodo de notificación del temporizador. Cada vez que recibe un aviso, oculta todas las casillas de la planta activa. Seguidamente, selecciona aleatoriamente varias casillas (el porcentaje de casillas visibles es una propiedad del escenario) y las muestra. Estas casillas quedarán visibles hasta el siguiente aviso del temporizador.

Configuración del juego

El escenario del juego se define utilizando un fichero de configuración. El formato del fichero es el siguiente:

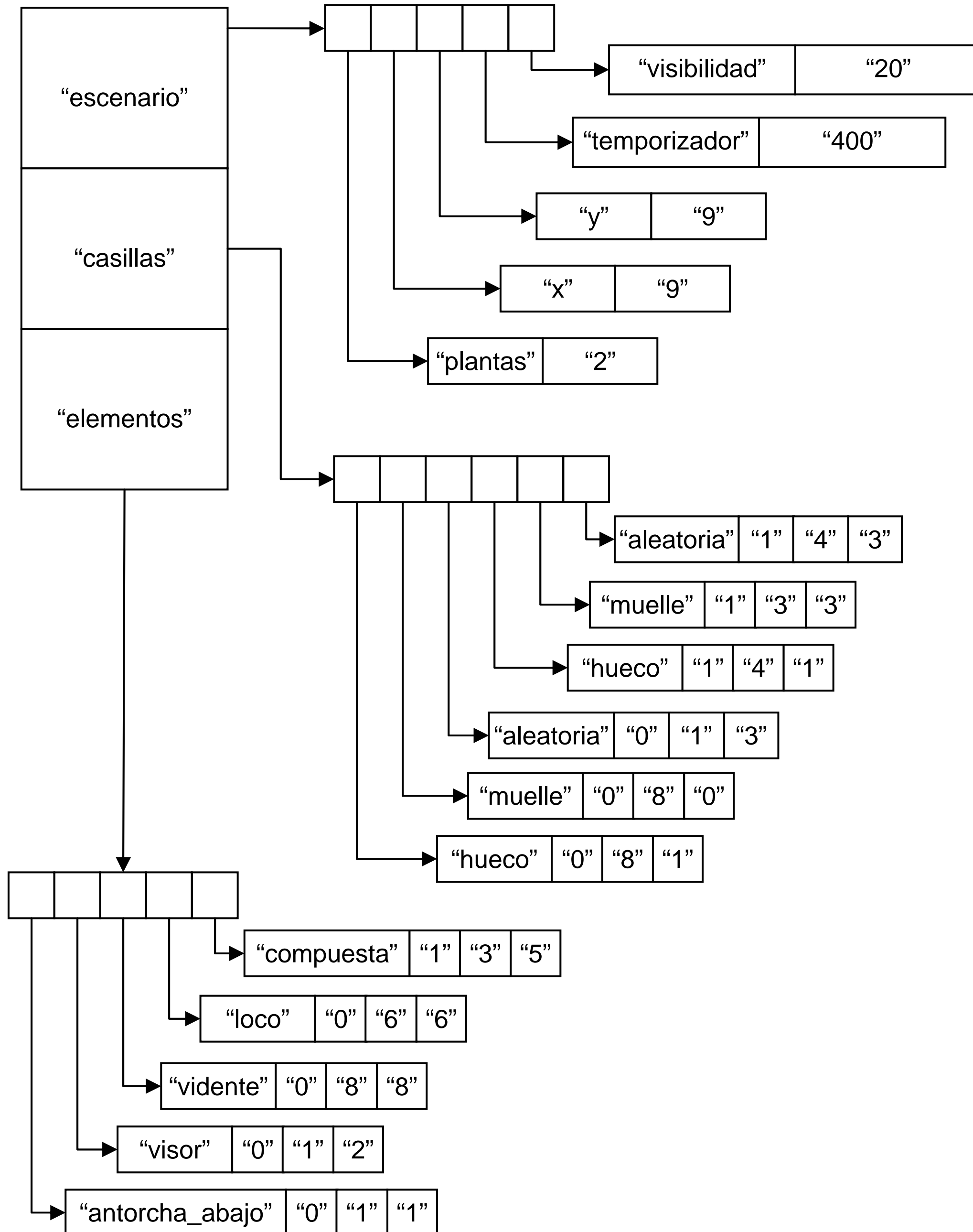
```
# Esto es un comentario que no se procesa
# Las líneas vacías también son ignoradas

# Declaración del escenario.

# Número de plantas y dimensiones

escenario plantas 2
escenario x 9
escenario y 9

# Periodo de notificación del temporizador en milisegundos

escenario temporizador 400

# Porcentaje de casillas visibles en una planta

escenario visibilidad 20

# Casillas especiales: tipo (planta, x, y)

casilla hueco (0, 8, 1)
casilla muelle (0, 8, 0)
casilla aleatoria (0, 1, 3)

casilla hueco (1, 4, 1)
casilla muelle (1, 3, 3)
casilla aleatoria (1, 4, 3)

# Elementos: tipo (planta, x, y)

elemento antorcha_abajo (0, 1, 1)
elemento visor (0, 1, 2)
elemento vidente (0, 8, 8)
elemento loco (0, 6, 6)

elemento compuesta (1, 3, 5)
```

El fichero de configuración procesa tres tipos de declaraciones:

- Propiedades del escenario: corresponden a las líneas que comienzan por “escenario”. Estas propiedades se declaran con un nombre (por ejemplo, “plantas”) seguidas de un valor entero.
- Casillas: son declaradas por líneas que comienzan por “casilla”, seguidas del tipo de casilla (por ejemplo, “muelle”) y la planta y posición dentro de la planta.
- Elementos: se declaran comenzando por “elemento”, a continuación el tipo (por ejemplo, “vidente”) y la situación en el escenario.

Se proporciona una biblioteca de código encargada de procesar el fichero de configuración y construir una estructura de datos con la información recogida en el fichero. Se recomienda consultar la documentación `JavaDoc` que acompaña a la biblioteca. A continuación se muestra la estructura de datos correspondiente al ejemplo de fichero de configuración anterior.

`Map<String, List<List<String>>>`



En la figura anterior se puede observar que los datos del fichero de configuración son representados por cadenas, aunque algunos datos expresen números. Por tanto, es necesario convertir cadenas en enteros. Utiliza el método de clase `Integer.parseInt()` para convertir una cadena en un entero.

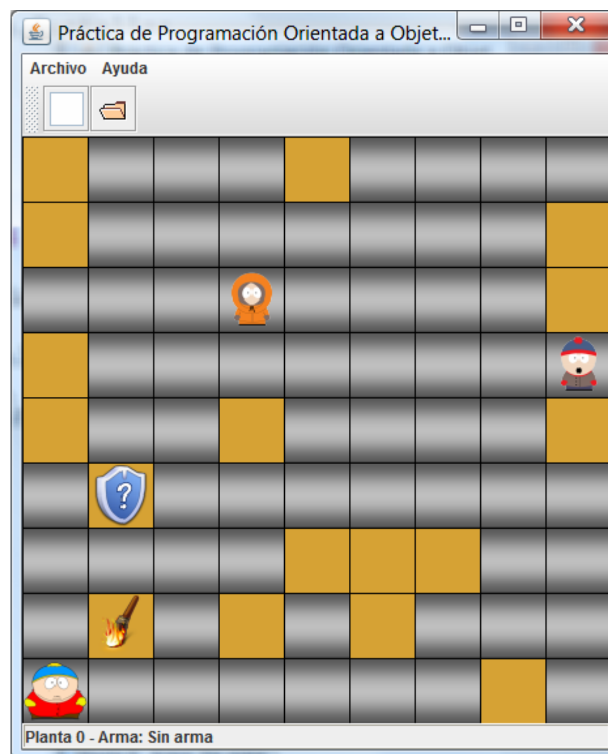
El procesamiento del fichero de configuración no controla los nombres de las propiedades del escenario, los nombres de los tipos de casillas o elementos. Sólo tiene en cuenta que para esos datos se declare una cadena alfanumérica que puede contener el carácter "_". Así mismo, sólo se controla que los valores numéricos sean enteros mayores o iguales que cero.

Por tanto, la biblioteca de configuración no comprueba la corrección semántica del fichero. Por ejemplo, es responsabilidad del programador comprobar que las inserciones de las casillas y elementos sean correctas, es decir, que el número de planta y las coordenadas sean correctas en el escenario. En caso de error, fallaría la precondición del diseño por contrato de las operaciones que añaden casillas y elementos al escenario. Tampoco se controla que falten propiedades del escenario. Trata estos errores siguiendo las recomendaciones del Tema 4 y Seminario 4.

Por último, nótese que en el fichero de configuración no se define la entidad jugador. Se asume implícitamente que existirá un jugador que se situará en la casilla (0, 0) de la primera planta. Por tanto, la tarea de creación del jugador es responsabilidad del programador de la aplicación.

Interfaz gráfica

La interfaz del videojuego será gráfica. Se ofrece una biblioteca de clases documentadas en `JavaDoc` que proporcionan la interfaz gráfica del juego. La biblioteca de código implementa la clase `Pantalla` que se encarga de realizar todas las tareas relativas a la visualización del juego: construcción de una ventana, cálculo de las dimensiones de dibujo, dibujo de las casillas, dibujo de los elementos, captura de las teclas con las acciones del usuario, etc. Por tanto, el programador de la aplicación no debe tratar con la representación gráfica.

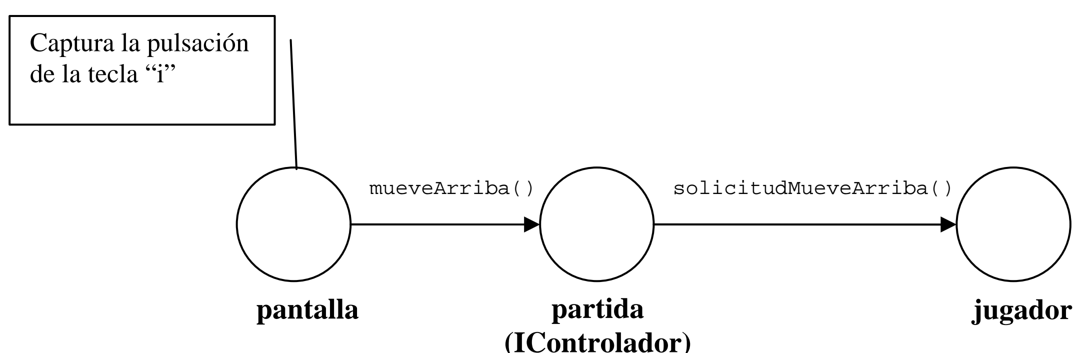


Sin embargo, la clase `Pantalla` requiere información del juego para realizar la visualización gráfica. Por ejemplo, la pantalla necesita conocer la posición de las casillas y elementos, y la imagen que se empleará para su representación. Los requisitos de información necesarios para la visualización del juego son expresados en la biblioteca gráfica utilizando dos interfaces: `IControlador` y `Dibujable`.

Es responsabilidad del programador de la aplicación la correcta implementación de esas interfaces. La clase que implemente la interfaz `IControlador` (clase `Partida`) será la encargada de proporcionar la información a la pantalla (colección de casillas, etc.) y además recibirá mensajes por parte de la pantalla notificándole la pulsación de las teclas (mueve hacia arriba, mueve hacia abajo, etc.).

Cualquier objeto que vaya a ser representado gráficamente en el juego debe implementar la interfaz `Dibujable`. A través de esa interfaz se proporciona la posición y el nombre de la imagen con la será representado en la pantalla.

Uno de los aspectos más destacados del uso de la biblioteca gráfica es el flujo de mensajes que se origina con la pulsación de una tecla hasta que el objeto jugador tiene conocimiento de la acción que ha de realizar. Tal como se ha comentado anteriormente, la clase `Pantalla` es la responsable de tratar con los eventos gráficos relacionados con la pulsación de las teclas. Supongamos que se pulsa la tecla “i” (arriba). La pantalla maneja la pulsación de tecla y se encarga de comunicar el suceso al juego a través del objeto `IControlador`. En la aplicación la interfaz `IControlador` sería la clase `Partida`. Por tanto, un objeto de esta clase recibiría el mensaje `mueveArriba` indicando que el usuario acaba de pulsar la tecla de movimiento hacia arriba. El objeto de la clase `Partida` (`IControlador`) actúa como mediador en el juego, recibiendo mensajes de la interfaz gráfica y proporcionando información. En este caso, el mensaje de movimiento hacia arriba debe conducirlo al objeto jugador que está interesado en la pulsación de teclas. Por tanto, la partida utiliza la referencia al jugador (podría almacenarla como atributo) para comunicarle que debe moverse hacia arriba la próxima vez que deba realizar su acción (el jugador es un elemento autónomo). Para recibir esa notificación la clase `Jugador` ofrece el método `solicitudMueveArriba` y lo implementa recordando en un atributo que la próxima vez que sea accionado debe moverse hacia arriba. La secuencia de eventos puede resumirse del siguiente modo:



Los detalles de uso de la biblioteca gráfica pueden ser consultados en la documentación `JavaDoc`. Se recomienda consultar la documentación a partir del fichero `index.html` que incluye un resumen de uso de la librería y proporciona los enlaces a la documentación de la clase `Pantalla` y de las interfaces.

Por último, en cuanto a la **visualización de los elementos del juego** debe tenerse en cuenta que todos los elementos siempre serán dibujados, salvo las armas. Éstas se visualizarán en función de la casilla en la que se encuentren. Es decir, si están en una casilla oculta no se visualizarán. Un objeto *dibujable* no se representa si el nombre de su imagen es la cadena vacía.

Temporizador

El uso de un temporizador es esencial para el desarrollo del juego. El escenario está encargado de activar los elementos autónomos y de mostrar casillas al azar. Se recomienda el uso de los temporizadores de Java para programar esta funcionalidad. En la documentación de la clase `javax.swing.Timer` se puede encontrar más información sobre el uso de temporizadores.

La clase `javax.swing.Timer` implementa los temporizadores en Java. Seguidamente se describen sus características:

- Para el uso de la clase `Timer` hay que importar la clase `javax.swing.Timer` y el paquete `java.awt.event`.
- Los objetos `Timer` se encargan de avisar periódicamente a objetos que implementen la interfaz `ActionListener`.

```
public interface ActionListener{
        void actionPerformed(ActionEvent e);
}
```

- La implementación del método `actionPerformed` define lo que harán los objetos cuando llegue el aviso del temporizador.

A continuación se muestra un ejemplo de un marcador de segundos que recibe avisos del `Timer` cada 1000 milisegundos para actualizar la cuenta de segundos. En este ejemplo se introduce una espera hasta la finalización del conteo de segundos. **En la implementación del videojuego no hay que realizar ninguna espera tras arrancar el temporizador.**

```
import javax.swing.Timer;
import java.awt.event.*;

class MarcaSgdos implements ActionListener{
        private int sgdos = 0;

        public void actionPerformed(ActionEvent e){
                sgdos++;
                System.out.println(sgdos);
        }

        public int getSgdos(){
                return sgdos;
        }
}
public class TestTimer {
        public static void main(String[] args) {

                //milisegundos que transcurren entre avisos
                int delay = 1000;
                MarcaSgdos oyente = new MarcaSgdos();
                Timer crono = new Timer(delay, oyente);
                System.out.println("Crono conectado...");

                crono.start();

                // Espera hasta que se alcance el límite de segundos.
                while (oyente.getSgdos()<Integer.parseInt(args[0])){}

                crono.stop();
                System.out.println("Crono parado ...");

        }
}
```

Ajustes del juego

El comportamiento del juego depende del valor definido por dos propiedades del escenario que serán definidas en el fichero de configuración. A continuación se dan recomendaciones sobre el ajuste de estos valores:

- **Periodo del temporizador del escenario**: el periodo de notificación establecido para el temporizador condiciona el ritmo del juego y por tanto la dificultad. Establecer un valor de 200 milisegundos significa que cada segundo se aplican 5 acciones a cada elemento autónomo y que las casillas se mostrarán sólo durante 200 milisegundos. Por tanto, este factor determina la velocidad del juego. Se recomienda ajustar esta propiedad para establecer la dificultad del juego.
- **Visualización de las casillas**: en cada aviso del temporizador algunas casillas de la planta activa son mostradas. Esta propiedad representa el porcentaje de casillas que se mostrarán en cada etapa de visualización. El juego será más difícil cuando menor sea el porcentaje.

Empaquetado de la aplicación

Junto al proyecto de programación hay que entregar la aplicación empaquetada para ser distribuida. La aplicación se empaqueta en el fichero **`muelles.jar`** de manera que pueda lanzarse la aplicación desde el explorador de ficheros del sistema operativo utilizando ese fichero.

Téngase en cuenta que la aplicación requiere de dos bibliotecas de código para su funcionamiento. Por tanto, la aplicación será distribuida en una carpeta con los siguientes ficheros:

- `muelles.jar`: corresponde con el ejecutable de la aplicación.
- `lib/muelles-vista.jar`: biblioteca para la representación gráfica. Se sitúa en la carpeta local “lib”.
- `lib/muelles-configuracion.jar`: biblioteca para el procesamiento de los ficheros de configuración. También se sitúa en la carpeta “lib”.
- Carpeta “`imagenes`”: contiene las imágenes que representan las casillas y elementos del juego.
- Ficheros de configuración: al menos un fichero de configuración.

Además, las librerías tendrán que enlazarse en el fichero `MANIFEST` al crear el empaquetado, de manera que el contenido del mismo debe ser:

```
Manifest-Version: 1.0
Main-Class: muelles.rutaClasePrincipal
Class-Path: lib/muelles-configuracion.jar lib/muelles-vista.jar
```

Suponiendo que las librerías se encuentran dentro del directorio de la aplicación, en la carpeta `lib`.