Curso 2017-2018. Cuatrimestre de otoño. 17 de Enero de 2018

Programación Orientada a Objetos (POO). Examen final. Duración del examen: 2h 30min

Publicación de notas provisionales: 23/1/2018.

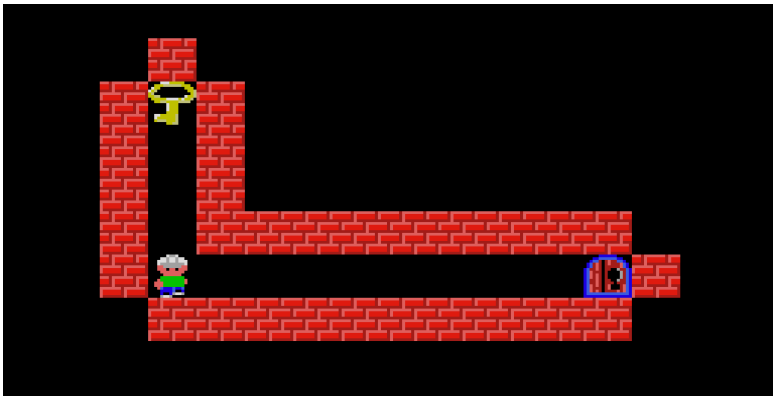
Revisión de examen:25/1/2018. Hora: 15:30 Lugar: C6-E101

El diagrama de clases UML (incompleto en términos de métodos de las clases) que se muestra a continuación pretende ser el correspondiente al núcleo de una aplicación que permite jugar a un antiguo videojuego llamado "Griel's quest for the Sangraal". **Leed la explicación que sigue con el diagrama de clases de la última página al lado**.
El juego presenta un recinto delimitado por paredes y cerrado por una puerta, y un personaje.

El recinto consta de varias celdas, cada una de las cuales tiene una coordenada [fila,columna]. Una celda puede estar libre o no. Si no está libre, en una celda puede estar el personaje controlado por el jugador o un “objeto en escena” .

Existen dos tipos de “objetos en escena”: los **obstáculos** (como Roca, Fuego, Hielo y Puerta) y los **Items** (como Pico, Agua, Soplete y Llave). Los ítems pueden ser recogidos por el jugador (quien podrá guardar UN SOLO ítem) y le sirven para eliminar los obstáculos en su camino hacia la Puerta. Cada ítem tiene uno o varios poderes (interface PoderItem): por ejemplo, el Agua puede tener entre otros el poder ApagaFuegos. Así, cuando el jugador pretenda eliminar el obstáculo Fuego presente en una celda, puede usar el Agua con su poder ApagaFuegos si dispone del ítem Agua porque lo ha recogido previamente.

El jugador debe conseguir que el personaje llegue y abra la puerta para pasar al siguiente nivel superando los obstáculos con los ítems presentes en la escena.



En el videojuego original, el jugador mueve su personaje utilizando las teclas cursor por el recinto cerrado para que éste pueda apoderarse de los diferentes ítems, cosa que hace al caer en una celda en la que haya uno de éstos, para así eliminar los distintos obstáculos. Cuando el personaje llega a la puerta con la llave en su poder, se pasa al siguiente nivel: el programa presenta al jugador un nuevo recinto cerrado más complejo con mayor número de obstáculos e ítems.

En este examen se os pedirá la realización de determinados métodos de la aplicación. **NINGUNO DE ELLOS TENDRÁ QUE VER CON LA INTERFICIE GRÁFICA DEL JUEGO**.

La figura de la última página del enunciado muestra el diagrama de clases de una primera aproximación al programa final. No os preocupéis por su tamaño. Solo tendréis que trabajar con una parte de los componentes del diagrama.

NOTA 1: Al resolver las preguntas del examen podéis suponer implementados los métodos inmediatos getXXX() y setXXX() que acceden a los atributos de las clases. **No es necesario que los implementéis a no ser que se indique lo contrario**.

NOTA 2: Si durante la resolución de las preguntas necesitáis de métodos más complejos que los de acceso directo a los atributos de una clase mencionados en la nota anterior, **DEBERÉIS IMPLEMENTAR DICHOS MÉTODOS**.

PREGUNTA 1 (1,25 PUNTOS) A partir del diagrama de clases UML de la figura, escribid para las clases e interfaces `ControlJuego`, `Nivel`, `Celda`, `Jugador`, `ObjetoEnEscena`, `Obstaculo`, `Item`, `PoderItem`, `Fuego`, `Agua` y `ApagaFuegos` lo siguiente: la parte de código de la definición que comienza con el "public class…." o "public interface" tal y como se indica a continuación:

1. **Para las clases** la parte de código de la definición que comienza con el "public class…." y el código correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones mostradas en el diagrama.
2. **Para los Interfaces** la parte de código de la definición que comienza con el "public interface…." y el código correspondiente a las cabeceras de las funciones especificadas en dichas interfaces.

NO debéis añadir nada más en el código en vuestras respuestas a esta pregunta. **IMPORTANTE**: el nivel contendrá una matriz bidimensional de celdas (relación "contiene"). Observad además que lo realmente importante de los poderes que tiene un ítem es ver si el ítem en cuestión tiene o no tiene un cierto poder, más que el orden en el que se guardan o el acceso rápido a uno de ellos.

```
public class ControlJuego {
  private List<Nivel> niveles ;
  private InterfazGrafica gui ;
  private int nivelActual = 0;
}
public class Nivel {
  private Jugador jugador;
  private Celda[][] celdas;
  private int nFilas;
  privte int nColumnas;
}
public class Celda {
  private int fila;
  private int columna;
  private ObjetoEnEscena objeto;
}
public class Jugador{
  private Celda celda ;
  private Item ítem;
}
public interface ObjetoEnEscena {
  public boolean puedePasar(Jugador jugador);
  public String getArchivoImagen();
  public boolean pasa(Jugador j, Celda c);
}
public abstract class Obstaculo implements ObjetoEnEscena {
}
public abstract class Item implements ObjetoEnEscena {
  protected Set<PoderItem> poderes;
}
public abstract class PoderItem {
}
public class Fuego extends Obstaculo {}
public class Agua extends Item {}
public class ApagaFuegos extends PoderItem {}
```

PREGUNTA 2 (0,75 puntos) Implementad los siguientes constructores:

1. El de Celda.

```
public Celda(int fila, int columna) {
        this.fila = fila;
        this.columna = columna;
        this.objeto = null;
    }
```

2. El de Nivel. Cuando acaba el trabajo de dicho método, el objeto Nivel debe tener un recinto con todas sus celdas creadas. Todas las celdas estarán vacías. El número de filas y columnas del recinto deben pasarse como argumentos.

```
public Nivel(int nFilas, int nColumnas) {
  this.nFilas = nFilas;
  this.nColumnas = nColumnas;
  this.celdas = new Celda[this.nFilas][this.nColumas];
  for(int i=0;i<nFilas;i++){
    for(int j=0;j<nColumnas;j++){
      this.celdas[i][j] = new Celda(i,j) ;
    }
  }
}
```

3. El de Item.

```
public Item() {
  this.poderes = new HashSet<PoderItem>();
}
```

PREGUNTA 3 (1 punto) Implementad el siguiente método en la clase **Nivel**:
`public void depositaObjetosEnNivel(char[][] mapa) throws NoObjectKnownException`

Este método debe depositar en las celdas del escenario del nivel los objetos y el jugador. Para ello se le pasa una matriz de caracteres de iguales dimensiones a las del recinto del nivel (las dimensiones están en los atributos `nFilas` y `nColumnas` de Nivel). En algunas de sus posiciones habrá un carácter diferente del carácter espacio en blanco. Esos caracteres indicarán que en la celda que ocupe la misma fila y columna del recinto del nivel debe aparecer el objeto identificado por dichos caracteres (o el jugador en caso de que el carácter sea 'J', tal y como se indica a continuación). El método debe recorrer las posiciones de `mapa`, inspeccionar sus contenidos e ir creando y depositando los objetos indicados en las celdas correspondientes del escenario del nivel.

Este método lanzará una excepción `UnknownObjectException` si se encuentra con un carácter que no se corresponde con ninguno de los que identifican algún objeto o al jugador. A continuación se muestran los caracteres que pueden aparecer en `mapa` y los objetos con los que se corresponden.

' ': (espacio en blanco). Celda vacía. '#': Pared, 'P': Puerta, 'L': Llave, 'S': Soplete, 'H': hielo, 'F': Fuego, 'A': Agua, 'T': Pico, 'R': Roca, 'J': Jugador

```
    public void depositaObjetosEnNivel(char[][] mapa) {
        for (int f = 0; f < nFilas; f++) {
            for (int c = 0; c < nColumnas; c++) {
                switch (mapa[f][c]) {
                    case 'J':
                        celdas[f][c].setJugador(new Jugador(celdas[f][c]));
                        break;
                    case 'A':
                        celdas[f][c].setObjeto (new Agua());
                        break;
                    case 'L':
```

```
                        celdas[f][c].setObjeto(new Llave());
                        break;
                    case 'T':
                        celdas[f][c].setObjeto (new Pico());
                        break;
                    case 'S':
                        celdas[f][c].setObjeto(new Soplete());
                        break;
                    case '#':
                        celdas[f][c].setObjeto(new Pared());
                        break;
                    case 'F':
                        celdas[f][c].setObjeto(new Fuego());
                        break;
                    case 'H':
                        celdas[f][c].setObjeto(new Hielo());
                        break;
                    case 'R':
                        celdas[f][c].setObjeto(new Roca());
                        break;
                    case 'P':
                        celdas[f][c].setObjeto(new Puerta());
                        break;
                    case ' ':
                        break; //Atributo objeto de Celda ya inicializado a null en constructor
                    default:
                       throw new UnknownObjectException() ;
                }
            }
        }
    }
```

PREGUNTA 4 (1,5 puntos) Implementad el método siguiente en la clase ControlJuego:
`public char[][] cargaMapa(String rutaFichero, int numLins, int numCols) throws IOException, ArchivoMalFormateadoException`

Este método debe leer del archivo cuya ruta se pasa por argumento, crear, rellenar y devolver la matriz bidimensional de caracteres que mostrarán el mapa del escenario de un nivel. En el archivo habrá (en principio) tantas líneas como filas en el escenario. Este valor se pasa por el argumento `numLins`. Cada línea contendrá (en principio) tantos caracteres como columnas haya en una fila del escenario. Este valor se pasa por el argumento `numCols`. Cada carácter de una fila indicará el contenido de la celda correspondiente del escenario.

En caso de que en el mapa haya algún problema de formateado (número erróneo de filas, número erróneo de caracteres en una fila o un carácter diferente a los mencionados en la pregunta anterior), el método debe lanzar la excepción `ArchivoMalFormateadoException`.

```
    public char[][] cargaMapa(String rutaFichero, int numLins, int numCols) throws IOException,
ArchivoMalFormateadoException {
        BufferedReader reader = new BufferedReader(new FileReader(rutaFichero));
       char[][] mapa = new char[numLins][numCols] ;
        int numLineaArchivo = 0;
        String leida = reader.readLine();
       numLineaArchivo ++;
        while(leida != null) {
          if(numLineaArchivo > numLineas) {
            throw new ArchivoMalFormateadoException("El archivo tiene más de " + numLins + "
```

```
filas");
          }
          if(leida.length()!=numCols){
            throw new ArchivoMalFormateadoException("La fila   " + numLineaArchivo + " tiene " +
leida.length() + " caracteres en lugar de " + numCols);
          }
           for(int i=0;i<leida.length();i++) {
             mapa[numLineaArchivo-1][i] = leida.charAt(i);
           }
// Otra forma: mapa[numLineaArchivo-1] = leida. toCharArray() ;
           leida = reader.readLine();
           numLineaArchivo++;
        }
        if(numLineaArchivo < numLineas) {
            throw new ArchivoMalFormateadoException("El archivo tiene menos de " + numLins + "
filas");
        }
        return mapa ;
    }
```

PREGUNTA 5 (1,25 puntos) Implementad el código de la clase **ApagaFuegos** de forma que durante la ejecución de todo el programa **SOLO se pueda crear un objeto de esa clase**. Para ello proponed soluciones para las siguientes preguntas:

1. Generad código para definir un atributo privado estático y final, de nombre INSTANCIA que sea una referencia a un objeto de la propia clase ApagaFuegos, e inicializadlo en la misma línea de la definición creando un nuevo objeto ApagaFuegos.

```
private static final ApagaFuegos INSTANCIA = new ApagaFuegos();
```

2. Implementad el siguiente método:

```
public static ApagaFuegos getInstance()
```

```
private static ApagaFuegos getInstance(){ return ApagaFuegos.INSTANCIA ;}
```

Este método debe devolver el atributo privado estático y final INSTANCIA antes creado.

Cuando se intente usar por primera vez algo de la clase ApagaFuegos, se ejecutará la línea que define INSTANCIA y crea el objeto ApagaFuegos. Después de eso, invocando al método estático getInstance() se tendrá acceso a ese único objeto ApagaFuegos.

3. Añadid código para evitar que puedan crearse otros objetos de la clase ApagaFuegos en código que no esté en la propia clase ApagaFuegos. **PISTA**: quienes asignan valores iniciales a los atributos de los nuevos objetos después del trabajo del operador new son los constructores. Pero los constructores son métodos y como tales son afectados por los modificadores de acceso.

```
private ApagaFuegos(){}
```

Ahora **implementad** el constructor de la clase **Agua**. Dicho constructor debe añadir el objeto ApagaFuegos a los poderes del objeto Agua construido.

```
public Agua() {
  poderes.add(ApagaFuegos.getInstance());
}
```

En las preguntas que siguen implementaréis métodos de varias clases que permiten al jugador pasar a través de casillas que contienen el ítem Agua y los obstáculos Fuego (que puede ser apagado con el ítem Agua) y Puerta (que puede ser abierta con el ítem Llave y que marca el final del nivel). La operativa descansa siempre sobre dos métodos

presentes en todo ObjetoEnEscena (esto es, en todo Item y Obstaculo): puedePasar() y pasaATraves(). Genéricamente, el primero determina si el jugador puede pasar a través del objeto presente en la celda a la vista del objeto y del ítem que el jugador lleva (si el objeto es el ítem Agua por ejemplo, y el jugador no lleva otro ítem, podrá pasar; si lleva otro ítem, no podrá puesto que no podrá cogerlo: el jugador solo puede llevar un ítem). El segundo hace que el jugador pase a través del objeto (si el objeto es el ítem Agua, "pasar a través" implicará que el jugador recogerá el ítem; si el objeto es Fuego implicará apagarlo con el ítem Agua, si es Puerta implicará abrirla con el ítem Llave). **Cuando se pase a través de cualquier objeto, el método pasaATraves() devuelve false, excepto cuando se pasa a través de la puerta. En ese caso el método devuelve true y ello indica el final del nivel.**

PREGUNTA 6 (1,25 puntos) Implementad los siguientes métodos de la clase **Item**:

`public boolean tienePoder(PoderItem poder)`

Este método devuelve true si el ítem en cuestión dispone del objeto PoderItem pasado como argumento poder; devuelve false en caso contrario.

```
public boolean tienePoder(PoderItem poder) {
  return poderes.contains(poder);
}
```

`public boolean puedePasar(Jugador jugador)`

Este método devuelve true si el jugador no posee ningún ítem (cuando un jugador cae en una casilla que tiene un Item, si el jugador no tiene ninguno en su poder puede recogerlo; por el contrario, si tiene algún ítem, no puede recoger el de la celda y por tanto no puede pasar por esa celda al impedírselo el ítem).

```
public boolean puedePasar(Jugador jugador) {
  return jugador.getItem() == null;
}
```

`public boolean pasaATraves(Jugador j, Celda c)`

Realiza las acciones pertinentes cuando el jugador pasado por parámetro "pasa a través del ítem" contenido en la celda pasada como segundo argumento. Por "pasar a través del ítem" se entiende que el método se encargue de que este ítem de la celda sea recogido por el jugador y se elimine dicho ítem de la celda (que queda libre a partir de ese momento). Este método **siempre devuelve false**.

```
public boolean pasaATraves(Jugador jugador, Celda c) {
  j.setItem(this);
  c.setObjeto(null);
  return false;
}
```

PREGUNTA 7 (1,25 puntos) Implementad los métodos que se indican a continuación:

En clase **Fuego**

`public boolean puedePasar(Jugador jugador)`

Este método devuelve true si el jugador pasado por argumento dispone de un ítem con el poder ApagaFuegos. Devuelve false en caso contrario.

```
public boolean puedePasar(Jugador jugador) {
  return jugador.getItem() != null && jugador.getItem().tienePoder(ApagaFuegos.INSTANCIA);
}
```

En clase **Obstaculo**

`public boolean pasaATraves(Jugador j, Celda c)`

Este método realiza las acciones pertinentes cuando el jugador pasado por el primer argumento "pasa a través del obstáculo" contenido en la celda pasada por el segundo argumento. Este método se invoca solo cuando previamente se ha comprobado que el jugador posee un ítem con un poder que elimina el obstáculo. En consecuencia, este método debe solamente eliminar el ítem del jugador, eliminar el obstáculo de la celda y devolver false.

```
public boolean pasaATraves(Jugador jugador, Celda c) {
  j.setItem(null);
  c.setObjeto(null);
  return false;
```

```
}
```

En clase **Puerta**

`public boolean pasaATraves(Jugador j, Celda c)`

Este método realiza las acciones pertinentes cuando el jugador pasado por el primer argumento "pasa a través de la puerta" contenida en la celda pasada como segundo argumento. Como para cualquier otro obstáculo, este método se invoca solo cuando previamente se ha comprobado que el jugador posee un ítem (la llave en este caso) con un poder (AbrePuertas en este caso) que permite abrir la puerta. Este método debe hacer lo que hacía el método definido en Obstáculo y **devolver siempre true**.

```
public boolean pasaATraves(Jugador j, Celda c) {
  super.pasaATraves(j, c);
  return true;
}
```

PREGUNTA 8 (1,75 puntos) Implementad los métodos siguientes:

En la clase **Celda**

`public boolean puedePasar(Jugador j)`

Este método devuelve true en caso de que el jugador puede pasar a través de la celda. Eso sucederá si en la celda no hay ningún objeto o si hay un objeto y el jugador "puede pasar a través de dicho objeto". NOTA: para generar el código de este método revisad el diagrama de clases y los métodos que habéis implementado en preguntas anteriores.

```
public boolean puedePasar(Jugador j) {
  return objeto == null || objeto.puedePasar(j);
}
```

En la clase **Celda**

`public boolean pasaATraves (Jugador j)`

Este método realiza la acción que debe llevarse a cabo cuando el jugador pasado como argumento, "pasa a través de esta celda". Este método se invoca solo cuando previamente se ha comprobado que el jugador puede pasar por esta celda. Si no hay objeto en la celda debe devolver true. Si lo hay debe "pasar a través del objeto (ítem u obstáculo)" que haya en la celda (ya habéis implementado métodos para ello en las preguntas anteriores) y retornando el valor devuelto por esa acción.

```
public boolean pasaATraves(Jugador j) {
  if (objeto == null) {
    return true;
  }
  return objeto.pasaATraves(j, this);
}
```

En la clase **Nivel**:

`public boolean intentaMoverJugador(int fila, int col)`

Este método intenta mover el jugador a la celda que está en la fila y columna indicadas en los argumentos fila y col. Si el jugador puede "pasar a través de esa celda", se hace que el jugador pase a estar en esa celda y se hace "pasar al jugador a través de dicha celda", retornando el valor correspondiente. En caso contrario, el método devuelve false.

```
public boolean intentaMoverJugador(int fila, int col) {
  if (celdas[fila][col].puedePasar(jugador)) {
    jugador.setCelda(celdas[fila][col]);
    return celdas[fila][col].pasaATraves(jugador);
  }
  return false;
}
```

NOTACIÓN PARA EL DIAGRAMA: Las clases se identifican con una **C** encerrada en un círculo. Las clases abstractas se identifican con una A encerrada en un círculo. Las interfaces java se identifican con una **I** encerrada en un círculo. Los modificadores de acceso para atributos y métodos se denotan de la siguiente manera: un cuadrado denota “private”. Un círculo denota “public”