Universidad Carlos III de Madrid

Programación de Sistemas
Grado en Ingeniería Telemática

Leganés, 13 de marzo de 2013 — Examen parcial 1 (problemas)
Duración de la prueba: 45 min — Puntuación: 5 puntos sobre 10 del examen

Problema 1 (3,5 puntos)

Nos han encargado implementar un programa que juegue al ajedrez. En concreto, queremos diseñar las clases que representan cada pieza aprovechando las ventajas del paradigma de la orientación a objetos. En concreto, haremos una clase genérica llamada *Pieza* y otra clase por cada tipo de pieza del ajedrez.

Ya nos dan programada, y por tanto no es necesario que programes tú, una clase *Casilla* con estos métodos que nos resultarán útiles:

```
// Devuelve cierto si otraCasilla es adyacente a esta
public boolean esAdyacente(Casilla otraCasilla) {...}
// Devuelve la pieza de esta casilla, o null si está vacía
public Pieza getPieza() {...}
// Establece la pieza en esta casilla; si es null se marca como vacía
public void setPieza(Pieza pieza) {...}
```

Apartado 1 (2,25 puntos)

Programa la clase *Pieza* siguiendo indicaciones de este apartado. Pondremos los siguientes atributos privados en la clase *Pieza*: *casilla* en el tablero en que está situada (un objeto de la clase *Casilla*) y *color* de la pieza (*booleano* que toma valor *cierto* si la pieza es blanca).

La clase debe tener métodos "get" que permitan leer el valor de estos atributos. Además, queremos que tenga un constructor que reciba los valores necesarios para inicializar ambos atributos.

Queremos también declarar un método llamado *esAlcanzable* que reciba una casilla destino. El método devolvería un *booleano* con valor *cierto* si la pieza puede llegar en un movimiento a la casilla destino desde su casilla actual, o *falso* en caso contrario. Obviamente, no tiene sentido proporcionar código para este método en *Pieza*, y por tanto sólo necesitamos declararlo de forma que se obligue a las piezas concretas a proporcionar su código.

Por último, queremos un método llamado *mover* que reciba una casilla destino. Si dicha casilla es alcanzable por la pieza, actualiza el atributo *casilla* de esta pieza. Además, establece esta pieza en la casilla destino mediante su método *setPieza* y marca como vacía la casilla origen. Si la casilla destino no es alcanzable, el método no debe hacer nada. Este método sí se puede programar en *Pieza*, gracias a que hemos declarado el método *esAlcanzable*, y por tanto lo podemos invocar desde *mover*.

Apartado 2 (1,25 puntos)

Programa la clase *Rey* de tal forma que sea instanciable. Un rey es una de las piezas del ajedrez. Para no complicar innecesariamente el apartado, asume que el rey puede alcanzar casillas contiguas a la suya actual, siempre que dicha casilla esté vacía u ocupada por una pieza del color contrario. Ignoraremos tanto enroques como el hecho de que el rey pueda entrar en jaque al moverse a la casilla destino.

Problema 2 (1,5 puntos)

Programa una ventana que presente dos botones. Utiliza un *FlowLayout* para que los botones aparezcan uno al lado del otro. El texto de los botones será "Añadir" y "Borrar".

Tenemos una clase escuchadora llamada *EscuchadorBotones* que implementa *ActionListener* y tiene un constructor que no recibe parámetros. Esta clase ya está programada, no la tienes que programar. Haz que una única instancia de esta clase escuche los eventos de acción de los dos botones (esto es, la misma instancia escucha los eventos de ambos).

Problema 1

Apartado 1

```
public abstract class Pieza {
        private Casilla casilla;
        private boolean color;

        public Pieza(Casilla casilla, boolean color) {
                this.casilla = casilla;
                this.color = color;
        }

        public Casilla getCasilla() {
                return casilla;
        }

        public boolean getColor() {
                return color;
        }

        public abstract boolean esAlcanzable(Casilla destino);

        public void mover(Casilla destino) {
                if (esAlcanzable(destino)) {
                        casilla.setPieza(null);
                        destino.setPieza(this);
                        casilla = destino;
                }
        }
}
```

Declaramos el método *esAlcanzable* como abstracto porque, tal y como indica el enunciado, no tiene sentido proporcionar su implementación en *Pieza*. Deben ser las clases que hereden de pieza (torre, alfil, dama, peón, etc.) las que proporcionen el código de este método según la pieza concreta de que se trate. Esto obliga también a que la clase *Pieza* sea abstracta.

Pese a ello, nada impide que el método *mover* invoque al método *esAlcanzable*, porque está garantizado que todos los objetos de clases que hereden de *Pieza* van a disponer de una implementación para *esAlcanzable*. Según el objeto sobre el que se invoque a *mover*, la máquina virtual ejecutará el *esAlcanzable* que corresponda.

Apartado 2

```
public class Rey extends Pieza {

        public Rey(Casilla casilla, boolean color) {
                super(casilla, color);
        }

        public boolean esAlcanzable(Casilla destino) {
                return destino.esAdyacente(getCasilla())
                        && (destino.getPieza() == null
                            || destino.getPieza().getColor() == !getColor());
        }
}
```

Como el rey es una pieza, la clase *Rey* debe heredar de *Pieza*. El enunciado no menciona que hagan falta atributos específicos en esta clase, así que basta con los heredados de *Pieza*. El constructor, por tanto, sólo necesita recibir los valores iniciales de estos dos atributos e invocar con ellos al constructor de la superclase.

Por otra parte, para que se pueda crear instancias de *Rey*, esto es, para que *Rey* no sea una clase abstracta, hay que implementar *esAlcanzable*.

Problema 2

```
public class Interfaz extends JFrame {
        public Interfaz() {
                getContentPane().setLayout(new FlowLayout());
                JButton anadir = new JButton("Añadir");
                JButton borrar = new JButton("Borrar");
                getContentPane().add(anadir);
                getContentPane().add(borrar);
                ActionListener escuchador = new EscuchadorBotones();
                anadir.addActionListener(escuchador);
                borrar.addActionListener(escuchador);
                pack();
                setVisible(true);
        }
}
```

No se tendrá en cuenta en la corrección los *import* ni la invocación al método *pack()*.