Cuarta Práctica (P4)

Diseño e implementación usando herencia

Competencias específicas de la cuarta práctica

- Implementación de mecanismos de reutilización incluidos en un diseño.
- Comprensión del concepto de polimorfismo y su tratamiento en los distintos lenguajes de programación.

Objetivos específicos de la cuarta práctica

- Aprender a interpretar diagramas de clases que incluyen mecanismos de reutilización.
- Ser capaz de realizar las modificaciones en el código previo de las prácticas para incorporar relaciones de herencia.
- Saber cómo redefinir correctamente un método, entendiendo si amplía o modifica el funcionamiento de su superclase.
- Ser capaz de implementar una clase abstracta en Java.
- Llegar a entender el concepto de polimorfismo y su tratamiento en los diferentes lenguajes de programación.

Desarrollo de esta práctica

En esta práctica se modificará el diseño del modelo utilizando herencia. Se rediseñarán las casillas y las sorpresas y se añadirá un nuevo tipo de jugador.

Casillas

En el diseño actual del juego la clase casilla incluye responsabilidades de distintos tipos de casillas. Se desea disponer de una clase para cada tipo de *Casilla*: calle, impuesto, sorpresa, juez y para las casillas de descanso que no producen ningún efecto cuando llega un jugador: salida, parking y cárcel.

Se propone que la clase *Casilla* pase a representar a las casillas de tipo descanso, con poca funcionalidad, y que de esta clase se deriven mediante herencia subclases para el resto de subtipos de casillas, extendiendo funcionalidad y especializando cada una de ellas. Posiblemente tendrás que cambiar la visibilidad de algún método en *Casilla* para que pueda ser usado o redefinido por sus subclases.

Tendrás que eliminar la clase *TipoCasilla*, que ya no tiene sentido, y hacer cambios en *CivitasJuego* y en Tablero, que usan los constructores de *Casilla*, para que se adapten a este nuevo modelo. También es posible que tengas que hacer algún “casting” en Java en el uso de algún

método de *CasillaCalle*.

Sorpresas

Al igual que con las casillas, se va a crear una clase por cada tipo de sorpresa. También se dispondrá de una clase Sorpresa de la que se derivarán por herencia el resto, aunque en este caso no existirán instancias de esa clase. Por ello, se aconseja que en Java se implemente la clase Sorpresa como una clase abstracta.

Al igual que antes, tendrás que cambiar alguna visibilidad de métodos y atributos, así como eliminar la clase *TipoSorpresa y* hacer algún "casting".

Además, deberás usar los constructores de las nuevas clases en el método *inicializaMazoSorpresas* de *CivitasJuego*.

Jugador Especulador

Se desea añadir al juego un nuevo tipo de jugador que se llamará JugadorEspeculador. Los jugadores especuladores se comportan como los jugadores de los que ya dispones, salvo en las siguientes cuestiones:

- Pueden construir el doble de casas y hoteles respecto a un jugador normal (atributo de clase FactorEspeculador=2).
- En caso de que se les requiera ser encarcelados (método *encarcelar*), si no disponen del salvo conducto o carta de libertad, pueden intentar pagar una fianza de su saldo (atributo *fianza*) y así evitar la cárcel. El pago de la fianza siempre implica que se pueda hacer frente al pago con el saldo sin caer en bancarrota.
- Los especuladores solo pagan la mitad de impuestos cuando se les solicita.

Los jugadores especuladores solo se generan por conversión de un jugador normal. En Java, se recomienda que el constructor de JugadorEspeculador utilice el constructor de copia de Jugador para hacer la conversión, añadiendo un nuevo parámetro que será la fianza. En Ruby, el constructor solo inicializará la fianza, y se hará uso de un método de clase (nuevoEspeculador(jugador,fianza)) que creará un nuevo jugadorEspeculador, llamará al método copia de la clase Jugador, para crear el resto de atributos del JugadorEspeculador mediante copia del Jugador, actualizará las propiedades por conversión (ver párrafo siguiente) y devolverá el jugadorEspeculador creado. Recordar que en Ruby el método *copia(otroJugador)* de la clase jugador es el constructor por copia.

Es importante tener en cuenta que durante la creación de un especulador hay que indicar a sus propiedades que tienen un nuevo propietario (*actualizaPropietarioPorConversion*), ya que a nivel de identidad el jugador especulador es un objeto distinto al jugador que se usa como base en el proceso de conversión.

Se añade un nuevo tipo de sorpresa que indica que el jugador debe convertirse. Al aplicar esta sorpresa se sustituirá al jugador por un nuevo *JugadorEspeculador* en la lista de jugadores, invocando para ello al constructor, que recibirá como parámetros el jugador de partida y la fianza especificada en el valor de la carta sorpresa.

También es conveniente que el método que convierte la información de un jugador a una cadena de caracteres se redefina en el jugador especulador para añadir información relativa a que se trata de un tipo de jugador.

Por último, debes prestar también atención a si es necesario volver a declarar consultores de *CasasMax* y *HotelesMax* en JugadorEspeculador y cambiar la visibilidad de esos métodos en la clase Jugador.

Para probar si se hace bien la conversión te aconsejamos que crees una clase de prueba llamada TestP4, con un método main que cree un Jugador, le asocie una propiedad, y luego lo convierta a JugadorEspeculador y lo muestre. Comprueba que la propiedad que se le asocia cambia de propietario al hacer la conversión.