Universidad Nacional Autónoma de México

Facultad de Ciencias

Introducción a Ciencias de la Computación

Practica 19

Profesora: Amparo López Gaona
Ayudante: Diana Isabel Ramírez García
Ayudante: Francisco Manuel Monreal Gamboa

2021-1

1. Objetivo

El objetivo de esta práctica es que el alumno adquiera experiencia en la creación de jerarquıas de clases que contengan al menos una clase cuyo comportamiento no puede definirse completamente al momento de su creación y por lo tanto obligan a que sus subclases implementen los métodos que no se pueden definir en ella. Estas clases se conocen como clases abstractas. Las clases abstractas sirven para especificar el comportamiento que deben tener las clases derivadas. También se presenta el concepto de polimorfismo y se ejercita su uso a lo largo de las prácticas.

2. Introducción

Una *clase abstracta* es aquella que tiene al menos un método sin implementación porque no es posible definirlo, este método se denomina abstracto y su implementación se deja a las clases descendientes. Las clases abstractas sirven para especificar el comportamiento que deben tener las clases derivadas sin incluir la implementación de algunos métodos.
La habilidad de decidir cuál método aplicar a un objeto en una jerarquıa de herencia se denomina *cpolimorfismo*. La palabra polimorfismo significa varias formas, esto es que se permite usar el mismo nombre para referirse a distintos métodos. Por lo tanto, la misma llamada puede, en distintos momentos, referirse a diferentes métodos dependiendo de la clase del objeto que la hace. Las clases abstractas facilitan el polimorfismo, pues se tiene al menos un método con la misma firma pero cuyo comportamiento depende de cada subclase.
Una clase abstracta se programa igual que cualquier otra clase, excepto que la palabra `abstract` precede la palabra `class` y se permite que al menos un método no tenga cuerpo, en ese caso la firma del método empieza con la palabra `abstract` y el cuerpo se sustituye por un punto y coma. No es posible crear objetos de clases abstractas debido a que éstas pueden no tener completamente definido el comportamiento de sus objetos.

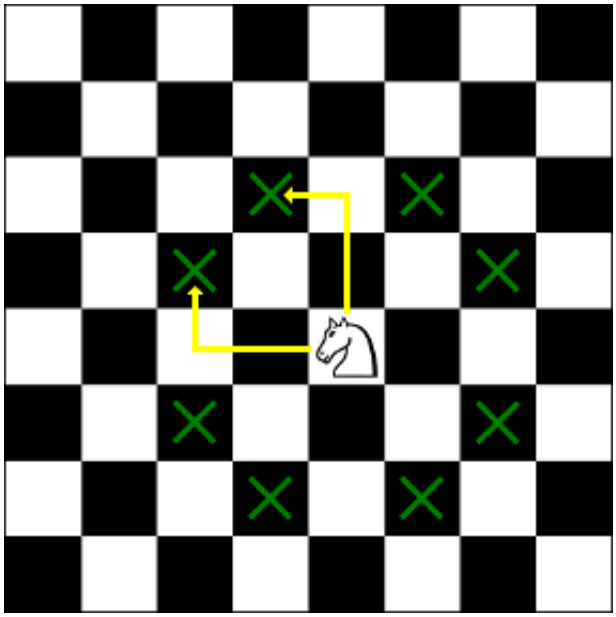
```
[modificador] abstract tipoDevuelto nombreMetodo(parametros);
```

3. Desarrollo

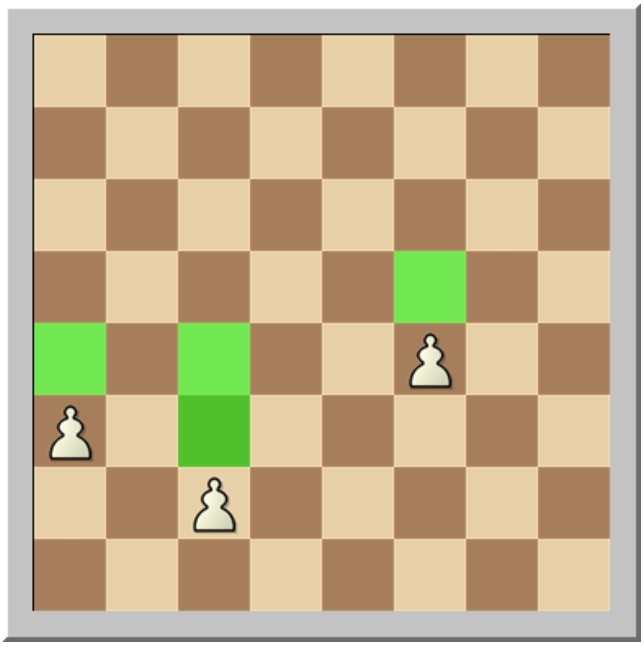
La práctica consiste en escribir un programa para que indique los posibles movimientos de una pieza de ajedrez dada una posición. No se trata de un programa que juegue ajedrez, por lo tanto basta con considerar que el tablero tiene una sola pieza en cada ejecución del programa.
Durante la práctica se debe crear una clase abstracta que sea la pieza de ajedrez y clases particulares para los peones, rey, caballos y torres. Cada pieza debe tener su posición y color.

1. Implementar una clase abstracta para las piezas de ajedrez. Todas las piezas deben tener posición, color, un método abstracto `posiblesMovimientos` que devuelva una cadena con las coordenadas de los posibles movimientos de una pieza a partir de su posición inicial.

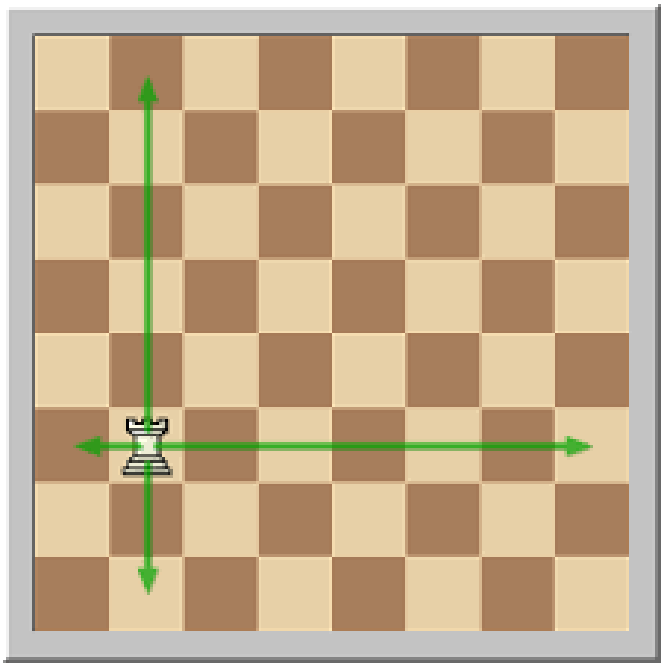
2. Programar la clase Caballo como una pieza de ajedrez cuyo movimiento no es lineal, como el de las otras piezas, sino que describe una trayectoria en forma de L; es decir, se desplaza dos casillas en dirección horizontal o vertical y una en dirección perpendicular a la anterior.



3. Programar la clase Peon como una pieza que se mueve verticalmente por la columna en la que se encuentra, sin poder retroceder. En el primer movimiento, desde el punto inicial, pueden avanzar dos casilla y, a partir de allí, de uno en uno



4. Programar la clase Torre como una pieza que se mueve en una línea recta horizontal o vertical a lo largo de cualquier número de casillas



Hacer un programa llamado **Main.java** que pruebe las clases implementadas. Este programa debe crear un arreglo de objetos de la clase abstracta Pieza, pedir al usuario el tipo y ubicación de varias piezas. Crear estos objetos y almacenarlos en el arreglo. Luego en un ciclo obtener las posibles posiciones de las piezas del arreglo.

Ejemplo:

```
bienvenido, las piezas que puedes poner son: peon, torre y caballo

Nombre de la pieza: peon
ubicacion inicial: 2,2
Otra pieza (s/n): s

Nombre de la pieza: torre
ubicacion inicial: 3,7
Otra pieza (s/n): s

Nombre de la pieza: caballo
ubicacion inicial: 4,4
Otra pieza (s/n): s

Los movimientos posibles son:
El peon puede moverse de la posicion (2,2) a las casillas:
(2,3), (2,4)

La torre puede moverse de la posicion (3,7) a las casillas:
(3,1), (3,2), (3,3), (3,4), (3,5), (3,6), (3,8),
(1,7), (2,7), (4,7), (5,7), (6,7), (7,7), (8,7)

El caballo puede moverse de la posicion (4,4) a las casillas:
(6,5) (5,6) (3,6) (2,5) (2,3) (3,2) (5,2) (6,3)
```

Opcional
Programar los posibles movimientos para un alfil

4. Forma de entrega

1. Las prácticas serán entregadas de forma individual.
2. Cada práctica (sus archivos y directorios) deberá estar contenida en un directorio llamado <apellidoPaterno_nombre_pX>, donde X es el número de la práctica.

   Por ejemplo: monreal_francisco_p1
3. **NO** incluir los archivos .class dentro de la carpeta
4. Comprimir el directorio <apellidoPaterno_nombre_pX>, en un archivo <apellidoPaterno_nombre_pX>.zip

   Por ejemplo: monreal_francisco_p1.zip
5. Las prácticas deberán ser entregadas durante la clase de laboratorio, la hora límite será a las 12:00 del día en que se publiquen.
6. Los archivos de código fuente deben estar documentados.
7. Se pueden discutir y resolver dudas entre los integrantes del grupo. Pero cualquier práctica plagiada total o parcialmente será penalizada con cero para los involucrados.
8. La práctica sera enviada al classroom de la clase en la sección que le corresponda.