PROGRAMACION -PRACTICA 4

Objetivos

- definición de interfaces y clases que las implementan (tema 7)

Recursos

Proyecto eclipse que contiene, entre otros ficheros y carpetas:

- Paquete `es.upm.dit.prog.practica4`
- `PruebaPractica4.java` programa que prueba si funcionan bien las clases
- `PruebaInteractiva4.java` programa que recibe órdenes por teclado para probar el funcionamiento.

Descripción

Ya tenemos registrados los Drones y las Misiones en `CentroControl`. En esta práctica construiremos mecanismos para seleccionar o filtrar las Misiones: definiremos una interfaz cuyas clases implementan un método que indica si una `Mision` es interesante o no. Usaremos esta interfaz como selector al llamar a `getMisiones` y de esta manera obtendremos un array de `Misiones` sólo con aquellas que nos interese considerar. Los nuevos elementos necesarios para realizar esta práctica se imparten en el tema 7 -no se pueden usar los elementos que se verán en el tema 8.
Usaremos las clases `Dron`, Posicion y `Mision` tal como están y cambiaremos los contenidos de `CentroControl` de la práctica anterior.

Creación del proyecto

1. Descargue de moodle el fichero zip que encontrará en la entrada `Práctica 4: proyecto para empezar`. En Eclipse debe hacer `File->Import->General->Existing Projects into Workspace` e indicar la localización del fichero zip usando la opción `Select archive file`. El fichero se llama `PROG24practica4` y contiene varios archivos; dentro de la carpeta `src` se encuentra el paquete `es.upm.dit.prog.practica4` que es donde deberán estar las nuevas clases que desarrolle.
2. Copie las clases `Posicion`, `Dron`, `Mision` y `CentroControl` de la práctica 3 en el paquete `es.upm.dit.prog.practica4`.

Programación

1. Cree la interfaz `SelectorMision`.
Esta interfaz define la cabecera del método:
`public boolean seleccionar(Mision m)`
Devuelve `true` si la `Mision m` cumple la condición que se defina en las clases que la implementan.

2. Cree la clase `SelectorMisionTrue` que implementa la interfaz `SelectorMision`. Esta clase devuelve como válida cualquier `Mision` distinta de `null` que se pase como parámetro a `seleccionar` (devuelve `true`).

3. Cree la clase `SelectorMisionActiva` que implementa la interfaz `SelectorMision`.
Esta clase devuelve como válida cualquier `Mision` distinta de `null` y cuya situación sea activa que se pase como parámetro a `seleccionar` (devuelve `true`).

4. Cree la clase `SelectorMisionDronAlejado` que implementa la interfaz `SelectorMision`.
Esta clase tiene un constructor que toma como parámetros `Posicion p` y `double d` y los guarda como atributos. No es preciso añadir accesores, modificadores, `equals`, `toString` ni el resto de métodos.

Esta clase devuelve como válida cualquier `Mision` distinta de `null` que se pase como parámetro a seleccionar cuyo dron esté a una distancia de la posición `this.p` mayor que `this.d`.

5. Cree la clase `SelectorMisionDronDespegue` que implementa la interfaz `SelectorMision`.
Esta clase no tiene constructor con parámetros. No es preciso añadir accesores, modificadores, `equals`, `toString` ni el resto de métodos.

Esta clase devuelve como válida cualquier `Mision` distinta de `null` que se pase como parámetro a seleccionar cuyo dron tenga velocidad menor que 0.2 en los ejes x e y, y mayor que 0 en el eje z.

6. Cree la clase `SelectorMisionDronAterrizando` que implementa la interfaz `SelectorMision`.
Esta clase no tiene constructor con parámetros. No es preciso añadir accesores, modificadores, `equals`, `toString` ni el resto de métodos.

Esta clase devuelve como válida cualquier `Mision` distinta de `null` que se pase como parámetro a seleccionar cuyo dron tenga velocidad menor que 0.2 en los ejes x e y, y menor que 0 en el eje z.

7. Cree la clase `SelectorMisionDronEnPeligro` que implementa la interfaz `SelectorMision`.
Esta clase tiene un constructor que toma como parámetro un array de `Drones` y lo guarda como atributo. No es preciso añadir accesores, modificadores, `equals`, `toString` ni el resto de métodos.

Esta clase devuelve como válida cualquier `Mision` distinta de `null` que se pase como parámetro a seleccionar cuyo dron esté en peligro por cercanía a alguno de los drones del array que se ha pasado como parámetro.

8. En `CentroControl`, hay que mantener los métodos `public void addDron(Dron d)`, `public void addMision(Mision m)`, `public Dron[] getDrones()` y `public void update(long t)`.
Hay que eliminar el método `Mision[] getMisiones()` y en su lugar hay que escribir un método `public Mision[] getMisiones(SelectorMision sm)`. El objetivo del nuevo método es devolver un array `Mision[]` sólo con aquellas

diferentes de `null` y que cumplan una determinada condición (la del `SelectorMision` que se pase como parámetro). Si el `SelectorMision` que se pasa como parámetro es `null`, debe devolver el array obtenido usando un `SelectorMisionTrue`. En el resto del cuerpo del método, cuando se accede a cada casilla de `this.misiones` y se comprueba que no es `null`, hay que añadir la comprobación que se realiza al llamar al método `sm.seleccionar(Mision m)`. Puede hacer el código de este método como en los métodos eliminados: un recorrido contando cuántas `Misiones` cumplen la condición, creación del array de resultado, y rellenado del array copiando las `Misiones` que cumplen la condición.

Pruebas

1. Cuando haya terminado de programar las clases, puede pasar a probar su funcionamiento.

2. Si no tiene errores de compilación, puede ejecutar las pruebas empaquetadas que hay en `PruebaPractica4` (marque este fichero y ejecute el programa: `Run->Run Java application`.). Se crean objetos de las clases que estamos probando, se llama a sus métodos y se comprueba que los efectos y resultados son los correctos. No entregue la práctica hasta haber superado con éxito todas estas pruebas. Deberá leer los mensajes para comprobar que todo funciona como se espera. Este programa proporciona una nota provisional aproximada.

3. Otra forma de probar que funcionan bien es mediante el programa `PruebaInteractiva4`; para ejecutarlo marque el fichero y haga: `Run->Run Java application`. En la consola de eclipse aparecerá un mensaje de saludo, y un cursor para que pueda escribir alguna de estas órdenes; el resultado se mostrará en la siguiente línea, con el cursor preparado para la siguiente orden:

ORDEN	FUNCIONAMIENTO
`hello`	saluda
`status`	muestra los valores de las variables
`help`	muestra la lista de órdenes
`exit`	acaba el programa
`clear`	inicia las variables
`pos x y z`	crea this.pos que es una posicion (objecto Posicion) con x:double y:double z:double
`vel x y z`	crea this.vel que es una velocidad (objeto Posicion) con x:double y:double z:double
`dron id t control`	crea this.dron1 nuevo con id:String this.pos t:long this.vel control:boolean, guarda el anterior en this.dron2
`addvel`	añade vel a this.dron1
`mover1 t`	mueve el this.dron1 con t:long
`mover2 t`	mueve el this.dron2 con t:long

`peligro`	comprueba si this.dron1 está en peligro por this.dron2
`mision id n`	crea una misión con id:String, this.dron1 y n posiciones
`addpos t`	añade this.pos y y t:long a this.mision
`update t`	actualiza this.mision con t:long
`show`	muestra un diagrama con los drones de la estacion filtrados con el último selector
`adddron`	añade this.dron1 a this.cc
`getdrones`	obtiene drones no nulos de this.cc
`addmision`	añade this.mision1 a this.cc
`getmisiones`	obtiene misiones no nulas de this.cc
`updatecc t`	actualiza todos los elementos de this.cc con t:long
`sim tini tfin`	muestra una simulación del movimiento de los drones de la estacion con: tini:long tfin:long use los botones para parar, avanzar t+1 o continuar la simulación: || > >>
`seltrue`	devuelve todas las misiones
`selactivas`	devuelve las misiones activas
`selalejadas d`	devuelve las misiones con drones a mayor distancia de d:double de this.pos
`selaterrizando`	muestra las misiones con drones aterrizando
`seldespegando`	muestra las misiones con drones despegando
`selenpeligro`	muestra las misiones con drones en peligro

Puede copiar las órdenes del fichero `comandos4.txt` y comprobar los resultados.

Entrega y evaluación

No entregue la práctica hasta haber superado con éxito todas estas pruebas. Aunque el programa `PruebaPractica4` proporciona una nota aproximada, si aparece el siguiente mensaje o falta alguna clase o interfaz, su código no es correcto por lo que cuando se ejecute en el servidor para la corrección final aparecerán errores en la compilación y la nota será 0.0.

```
Error en las cabeceras de métodos de CentroControl, Posicion, Dron o Mision
```

No se considerará entregada la práctica hasta que no la suba a moodle -dentro del plazo indicado-; para hacer la entrega de la práctica a través de moodle debe:

1. Con el ratón sobre el proyecto `PROG24practica4`, en Eclipse seleccione el menú: `File->Export...->General->Archive File` (también puede aparecer en español como `Fichero zip`)
2. Seleccione la carpeta `src` de su proyecto, y allí los ficheros Java; en “To Archive file” indique la carpeta de su sistema de ficheros donde quiere colocar el fichero zip y el nombre de este fichero que debe ser

`PROG24practica4.zip`. Asegúrese de que se guarda en formato zip con las carpetas necesarias.
3. Fuera de Eclipse, localice el fichero `PROG24practica4.zip` y súbalo a la tarea Práctica 4 del Moodle de Programación.

Cuando termine el plazo de entrega de la práctica, se recogerán y comprobarán todas las entregas. Se asignará la nota 0.0 a las entregas que no se hayan recibido a tiempo, que no contengan los ficheros indicados o que no compilen correctamente. Para cada una se pasarán pruebas parecidas a las de la nota provisional. Las notas definitivas aparecerán en la entrada de moodle un tiempo después del cierre de la entrega, cuando los profesores corrijan todas las prácticas y revisen que no hay copias de código.

Se recuerda que la copia de código supone el suspenso automático de la práctica y de la asignatura, tanto para quien copia el código como para quien lo cede sin distinción. Además, se emprenderán medidas disciplinarias contra dichos alumnos ante la Escuela y la Universidad.