Oriol Estabanell, Albert Gubau u172749, u172750

Lab 2
Programación Orientada a Objetos

1. Descripción del problema a solucionar

En este laboratorio deberemos implementar la solución del anterior laboratorio para crear un programa que nos permita dibujar por pantalla una "**turtle**" y que ésta dibuje según las instrucciones que le vengan dadas por el programa que ejecute.

-Solución escogida para la resolución del problema

Deberemos crear un entorno gráfico en el que se mostrará la tortuga y sus dibujos llamado **LogoWindow**, deberemos crear una clase para gestionar este entorno llamada Logo y deberemos crear una clase para gestionar la tortuga llamada **Turtle**.

Como hemos dicho, utilizaremos la solución de la práctica anterior de manera que deberemos copiar las clases **Instruction** y **Program** de ésta para poder gestionar las instrucciones que seguirá la tortuga y poderlas realizar en el entorno gráfico.

-Implementación de la clase Turtle

Esta clase será la que nos permitirá gestionar la tortuga que dibujará en el entorno gráfico **Logo**. En cuanto a sus atributos podemos ver que tendremos la posición actual (**x, y**) y las direcciones que toma la tortuga (**dirX, dirY**) además de un booleano llamado pen que será el que nos dirá si la tortuga estará pintando o no:

```
private int x;
private int y;
private double dirX;
private double dirY;
private Boolean pen;
```

- Turtle (Constructor)

El método **Turtle** será el método constructor de la clase, éste nos permitirá crear nuevas instancias de nuestra clase de manera correcta:

```
Turtle(int x, int y, double dirX, double dirY){
    this.x = x;
    this.y = y;
    this.dirX = dirX;
    this.dirY = dirY;
    pen = false;
}
```

En este método asignamos los valores correspondientes a la posición y dirección inicial de la tortuga así como el estado del **pen** en falso puesto que en un inicio no pintará.

- getX, getY, dirX, dirY (getters)

Éstos serán los métodos que nos permitirán saber el valor de la posición y dirección de nuestra tortuga en cualquier momento:

```
public int getX(){
    return x;
}

public int getY(){
    return y;
}

public double dirX(){
    return dirX;
}

public double dirY(){
    return dirY;
}
```

Al ser aplicados devuelven los valores de la posición y la dirección de la tortuga.

- setX, setY, setCoord, setDirX, setDirY (setters)

Éstos serán los métodos que nos permitirán cambiar el valor de la posición y la dirección de la tortuga en cualquier momento:

```
public void setX(int x){
    this.x = x;
}

public void setY(int y){
    this.y = y;
}

public void setCoord(int x, int y){
    this.x = x;
    this.y = y;
}

public void setDirX(double dirX){
    this.dirX = dirX;
}

public void setDirY(double dirY){
    this.dirY = dirY;
}

public void setDir(int dx, int dy){
    this.dirX = dx;
    this.dirY = dy;
}
```

Como podemos observar, los **setters** asignarán a la variable correspondiente el valor que les entra por parámetro.

- forward y turn

Éstos dos métodos nos permitirán avanzar la tortuga y girarla, su implementación está basada en una gestión de la distancia que queramos avanzar la tortuga en el caso de **forward** y del angulo que queramos rotar la tortuga en el caso de **turn**.

```
public void forward(double distance){

    x += distance * dirX;
    y += distance * dirY;
}

public void turn(double a){

    double ar = a * Math.PI/180;

    double dirXanterior = dirX;
    double dirYanterior = dirY;

    dirX = Math.cos(ar)*dirXanterior - Math.sin(ar)*dirYanterior;
    dirY = Math.sin(ar)*dirXanterior + Math.cos(ar)*dirYanterior;

}
```

Como podemos ver, para implementar el método **forward**, nos bastará con asignar a la posición en la que se encuentra la tortuga un valor correspondiente a la multiplicación de la distancia por la dirección en la que apunta la tortuga.

En el caso de **turn**, deberemos pasar el angulo que nos entra como parámetro de la función a radianes de manera que podremos calcular la nueva dirección de la tortuga con el cálculo:

$$dx' = cos(\alpha)dx - sin(\alpha)dy$$
$$dy' = sin(\alpha)dx + cos(\alpha)dy$$

Para poder realizar este cálculo debemos antes realizar un **import** de la librería **Math** para poder usar el **cosinus** y el **sinus** del angulo que hemos pasado a radianes.

- setPen y isPenOn

Éstos métodos nos permitirán gestionar el lápiz de la tortuga y su estado:

```
public void setPen(Boolean on){
    pen = on;
}

public Boolean isPenOn(){
    return pen;
}
```

setPen será un **setter** del atributo **pen** que según el valor del booleano que le entre por parámetro cambiará el valor del pen a **true** o **false**. Si el pen está en true significará que la tortuga pintará, sino no.

isPenOn nos devuelve true si el **pen** está activo, es decir, si el atributo **pen = true**, y **false** si no lo está.

- draw

El método **draw** nos dibujará la tortuga, en el caso de éste laboratorio, se ha decidido que la tortuga sea un triangulo, por lo tanto, la implementación del método draw será la siguiente:

```
public void draw(Graphics g){

    int nPoints = 3;

    int[] xc = new int[3];
    int[] yc = new int[3];

    xc[0] = (int)(x+8*dirY);
    yc[0] = (int)(y-8*dirX);

    xc[1] = (int)(x-8*dirY);
    yc[1] = (int)(y+8*dirX);

    xc[2] = (int)(x+16*dirX);
    yc[2] = (int)(y+16*dirY);

    g.drawPolygon(xc, yc, nPoints);
}
```

Como podemos ver, se inicializan **2 vectores** de 3 enteros correspondientes a las coordenadas de los puntos del triangulo y una variable **nPoints** que se refiere al número de puntos a tener en cuenta, en nuestro caso serán 3 puesto que escogemos la forma de un **triangulo**.

Realizamos los cálculos correspondientes para dibujar un triangulo mediante los vectores de coordenadas y llamamos al método **drawPolygon** de la clase **Graphics** de java. Para poder llamar a éste método deberemos realizar un **import** de la clase **Graphics** y pasar por parámetro el entorno gráfico.

-Implementación de la clase Logo

Esta clase será la que nos permitirá gestionar como se leerán las instrucciones del programa principal y que hará la tortuga cuando se lean. Gestionará el tamaño de la ventana de la interfaz gráfica y sus componentes.

Como atributos de la clase podremos encontrar el ancho y el alto de la ventana en la que se mostrará la tortuga y sus dibujos (**width y height**), y la instancia de la tortuga que estará en la ventana y realizará acciones.

```
private int width;
private int height;
private Turtle t;
```

Para implementarla deberemos aplicar los métodos siguientes:

- Logo (constructor)

Este método será el método que nos permitirá crear nuevas instancias de la clase Logo de manera correcta:

```
Logo(int w, int h){
    width = w;
    height = h;
    t = new Turtle(300, 350, 1,0);
}
```

Como podemos ver se asignará a las variables **width i height** el valor que entre por parámetro del método y se creará una instancia de la tortuga en la posición **x= 300**, **y = 350** mirando hacia la derecha (de esta manera se ve bien posicionada la tortuga).

- getWidth y getHeight (getters)

Estos métodos serán los **getters** de la **classe** Logo y nos permitirán tomar el valor de las variables **width** y **height** en cualquier momento de la siguiente manera:

```
public int getWidth(){
    return width;
}

public int getHeight(){
    return height;
}
```

Como podemos observar, se devuelve el valor de la variable **width** y de la variable **height** de la clase **Logo**.

- resetTurtle

Este será el método que nos permitirá colocar la tortuga en su posición inicial de nuevo, como habíamos dicho que esta era **x = 300 y = 350** la colocaremos allí de nuevo y la pondremos con el lápiz apagado para que no pinte en un inicio.

```
public void resetTurtle(){
    t.setCoord(300,350);
    t.setPen(false);
}
```

Como podemos ver, utilizamos el método **setCoord** de la clase **Turtle** para asignar la coordenada en la que se situará la tortuga al resetearse y **setPen** para apagar el lápiz de la tortuga para que una vez hecho el **reset** su estado inicial sea no pintar.

- execute

Este será uno de los métodos más importantes de nuestra implementación. En este método leeremos las instrucciones del programa y las interpretaremos según su código y parámetro. Cuando el programa acabe se reseteará la tortuga y si ha habido un error de instrucciones se imprimirán los errores correspondientes.

```
public void execute(Program p, Graphics g){

    t.draw(g);
    if(p.isCorrect()){
        p.restart();
        while(!p.hasFinished()){
            Instruction instr = p.getNextInstruction();

            if(!instr.isRepInstruction()){

                if("FWD".equals(instr.getCode())){
                    int x = t.getX();
                    int y = t.getY();
                    t.forward(instr.getParam());

                    if(t.isPenOn()){
                        g.drawLine(x, y,t.getX(),t.getY());
                    }
                    t.draw(g);

                } else if("ROT".equals(instr.getCode())){
                        t.turn(instr.getParam());
                        t.draw(g);

                }else if("PEN".equals(instr.getCode())){
                    double on = instr.getParam();
                    if(on == 0){
                        t.setPen(false);
                    } else if(on == 1){
                        t.setPen(true);
                    }
                    t.draw(g);
                }
                System.out.println(instr.info());
            }
        }
        resetTurtle();
    }else{
       p.printErrors();
    }
```

Para empezar, dibujaremos la tortuga, seguidamente, comprobaremos si la secuencia de instrucciones son correctas, si lo son iteraremos a lo largo del programa hasta que este haya finalizado. Al iterar tomamos la instrucción con **getNextInstruction** que ya explicamos en el anterior informe, después realizamos las comprobaciones pertinentes.
Primero comprobamos que no sea una instrucción **REP** o **END**, si da el caso de que no lo es entonces será una instrucción que modificará la tortuga.
Si se trata de una instrucción con código **FWD** significará que debemos avanzar la tortuga y si el lápiz está activado aplicaremos **drawLine** para dibujar una línea mientras se desplaza y luego imprimiremos la tortuga en su nueva posición.

En el caso de que se trate de una instrucción con código **ROT** lo que deberemos hacer es rotar la tortuga, por lo tanto, aplicaremos el método que hemos comentado en la clase **Turtle** llamado **turn**, que rotará la tortuga el angulo que le pidamos.

En el caso de que se trate una instrucción con código **PEN** se tendrá que poner el estado del pen según el parámetro de la instrucción. Si el parámetro es un **0** lo pondremos en **false**, es decir, que no pinte, en el caso de que sea **1** lo pondremos en **true**, esto es, que pinte.

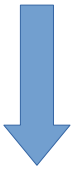
Por último, deberemos imprimir por pantalla la serie de instrucciones que se han llevado a cabo por la tortuga y al finalizar el programa reseteamos la tortuga en su posición inicial correspondiente y con el **pen** en false (método **resetTurtle**).

Cambio importante en **getNextInstruction**:

Hemos realizado un cambio importante en la implementación del método getNextInstruction, más específicamente en el método **goToStartLoop** que se llama en este:

```
private void goToStartLoop(){

    int counter = currentLine; //Inicializamos un contador d

    for(int i = currentLine; i < instructions.size(); i--){

        Instruction instruction = instructions.get(i);

        if(instruction.getCode() == "REP"){ //Si encontramos

            currentLine = counter + 1; //Asignaremos a curre
            break; //Aplicamos un break porque no tendremos
        }

        counter--; //Si no la encontramos decrementamos el c
    }
}
```

El cambio que hemos realizado ha sido el de iterar des de la instrucción **END** hasta el primer **REP** que se encuentre, de manera que iteramos hacia atrás desde la posición de la instrucción **END** hasta el primer **REP** para que funcione correctamente.



-Implementación de la clase LogoWindow

Esta clase será la que gestione el entorno gráfico mediante un panel **JFrame** en el que añadimos un componente **JPanel** para que se pueda dibujar en él.

Por lo tanto, esta clase heredará de la clase **java.swing.JFrame**. En esta clase tendremos dos atributos, un logo de tipo **Logo** y un programa de tipo **Program**.

```
Logo logo;
Program prog;
```

- LogoWindow (constructor)

Este será el método constructor de la clase, en este inicializaremos los componentes del Jframe con la función **initComponents()**, crearemos las instancias de los atributos **logo** y **prog** (programa), añadiremos las instrucciones que tendrá nuestro programa y asignaremos el tamaño de la ventana **Logo**.

```
public LogoWindow() {

    initComponents();
    logo = new Logo(800, 600);
    prog = new Program("Square");

    prog.addInstruction("PEN", 1);
    prog.addInstruction("REP", 4);
    prog.addInstruction("FWD", 100);
    prog.addInstruction("ROT", 90);
    prog.addInstruction("END", 1);
    prog.addInstruction("PEN", 0);
    prog.addInstruction("FWD", 200);
    prog.addInstruction("PEN", 1);
    prog.addInstruction("REP", 4);
    prog.addInstruction("FWD", 100);
    prog.addInstruction("ROT", 90);
    prog.addInstruction("END", 1);

    setSize(logo.getWidth(), logo.getHeight());
}
```

Como podemos ver, inicializamos la ventana logo con un tamaño de **800x600** y el programa con el nombre de **Square** seguido de las instrucciones correspondientes (las instrucciones que se muestran son las que dibujarán 2 cuadrados separados).
También vemos que aplicamos un **setSize** para aplicar el tamaño de la ventana que queremos.

- paint

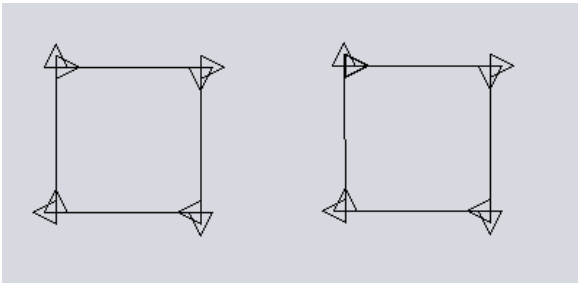
Este método será una re-definición del método **paint** de la clase **Graphics** de java. En este podemos ver que lo que hacemos es llamar al método constructor de la clase padre (**Graphics**) con el parámetro de gráficos (**g**) y seguidamente aplicamos el método **execute** de la clase **Logo** para que se ejecute el programa completo y la tortuga dibuje los dos cuadrados por pantalla.

```
@Override
public void paint(Graphics g){

    super.paint(g);
    logo.execute(prog, g);
}
```

-Conclusiones y resultado obtenido

Llegado al resultado final, podemos concluir que el programa funciona correctamente. Tal como indica el objetivo, hemos hecho una implementación en forma de código del diseño del seminario 2 de una aplicación de dibujo mediante un entorno gráfico. Este es el resultado que obtendríamos aplicando esta serie de instrucciones (dibujo de dos cuadrados):

```
PEN 1
REP 4
FWD 100
ROT 90
END 1
PEN 0
FWD 200
PEN 1
REP 4
FWD 100
ROT 90
END 1
```



En conclusión, nuestro programa implementa una solución en la que siguiendo las instrucciones anteriores muestra por pantalla la tortuga y los dos cuadrados a dibujar, por lo tanto, podemos decir que ha sido una implementación correcta en cuanto a este ejercicio. Por lo que hace a la prueba del programa, no hemos realizado muchas con la finalidad de encontrar errores así que no podemos comentarlos. Nuestro programa no podrá realizar bucles anidados, es decir, secuencias de 2 REP seguidos, por lo tanto, si queremos implementar lo siguiente:

```
REP 2
REP 4
FWD 100
ROT 90
END 1
END 1
```

Deberemos declararlo de esta manera:

```
REP 8
FWD 100
ROT 90
END 1
```