90 - Clases anidadas en Java

Listado completo de tutoriales

Hemos utilizado las clases anidadas en conceptos anteriores. Decimos que una clase es anidada si está contenida en otra clase.

Veremos que hay varios tipos de clases anidadas en Java:

- Clase anidada interna.
- Clase anidada estática.
- Clase local.
- Clase anónima.

Clase anidada interna.

El anidamiento de una clase tiene por objetivo favorecer el encapsulamiento. Una clase anidada se dice que es interna si se la declara dentro de otra clase pero fuera de cualquier método de la clase contenedora.

Puede declararse con cualquiera de los modificadores: private, protected o public.

Una característica fundamental es que una clase interna tiene acceso a todos los atributos de la clase que la contiene, luego para que exista una clase anidada interna es necesario que exista un objeto de la clase contenedora.

Problema:

Confeccionar una clase llamada Coordenadas que almacene la referencia de puntos en el plano mediante x e y. Declarar una clase interna que represente un punto en el plano.
La clase Coordenada debe almacenar en un ArrayList con elementos de tipo Punto. Además la clase Coordenadas debe poder calcular la cantidad de puntos almacenados en cada cuadrante.

Clase: Coordenadas

```
import java.util.ArrayList;

public class Coordenadas {

    private class Punto {
        private int x, y;
```

```
        public Punto(int x, int y) {
            fijarX(x);
            fijarY(y);
        }

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

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

        public int retornarCuadrante() {
            if (x > 0 && y > 0)
                return 1;
            else if (x < 0 && y > 0)
                return 2;
            else if (x < 0 && y < 0)
                return 3;
            else if (x > 0 && y < 0)
                return 4;
```

Clase: PruebaCoordenadas

```
public class PruebaCoordenadas {
    public static void main(String[] ar) {
        Coordenadas coordenadas = new Coordenadas();
        coordenadas.agregarPunto(30, 30);
        coordenadas.agregarPunto(2, 7);
        coordenadas.agregarPunto(-3, 2);
        coordenadas.agregarPunto(-5, -4);
        coordenadas.agregarPunto(-9, -2);
        System.out.println("Cantidad de puntos en el primer cuadrante:
                + coordenadas.cantidadPuntosCuadrante(1));
        System.out.println("Cantidad de puntos en el segundo cuadrante
                + coordenadas.cantidadPuntosCuadrante(2));
        System.out.println("Cantidad de puntos en el tercer cuadrante:
                + coordenadas.cantidadPuntosCuadrante(3));
        System.out.println("Cantidad de puntos en el cuarto cuadrante:
                + coordenadas.cantidadPuntosCuadrante(4));
    }
}
```

Se dice que la clase privada Punto es una clase interna de la clase Coordenadas:

```
public class Coordenadas {

    private class Punto {
```

La clase Punto define dos atributos privados:

```
        private int x, y;
```

En el constructor de la clase Punto llamamos a los métodos fijarX y fijarY para que se inicialicen los atributos x e y:

```
        public Punto(int x, int y) {
            fijarX(x);
            fijarY(y);
        }

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

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

Finalmente el método retornarCuadrante nos informa en que cuadrante se encuentra el punto o un -1 si el punto se encuentra sobre uno de los ejes x o y:

```
        public int retornarCuadrante() {
            if (x > 0 && y > 0)
                return 1;
            else if (x < 0 && y > 0)
                return 2;
            else if (x < 0 && y < 0)
                return 3;
            else if (x > 0 && y < 0)
                return 4;
            else
                return -1;
        }
    }
```

Por otro lado la clase Coordenadas define un atributo de tipo ArrayList con componentes de tipo Punto:

```
    private ArrayList<Punto> puntos;
```

En el constructor creamos el ArrayList:

```
    public Coordenadas() {
        puntos = new ArrayList<Punto>();
    }
```

El método 'agregarPunto' crea un objeto de la clase Punto y lo añade al ArrayList:

```
    public void agregarPunto(int x, int y) {
        puntos.add(new Punto(x, y));
    }
```

Para conocer la cantidad de puntos que hay en un determinado cuadrante debemos recorrer el ArrayList y consultar a cada Punto si coincide con el cuadrante que estamos buscando:

```
    public int cantidadPuntosCuadrante(int cuadrante) {
        int cant = 0;
        for (Punto pun : puntos)
            if (pun.retornarCuadrante() == cuadrante)
                cant++;
        return cant;
    }
```

Para probar la clase 'Coordenadas' creamos la clase 'PruebaCoordenadas' donde creamos un objeto, agregamos un conjunto de puntos y finalmente llamamos al método 'cantidadPuntosCuadrante':

```
public class PruebaCoordenadas {
    public static void main(String[] ar) {
        Coordenadas coordenadas = new Coordenadas();
        coordenadas.agregarPunto(30, 30);
        coordenadas.agregarPunto(2, 7);
        coordenadas.agregarPunto(-3, 2);
        coordenadas.agregarPunto(-5, -4);
        coordenadas.agregarPunto(-9, -2);
        System.out.println("Cantidad de puntos en el primer cuadrante:
                + coordenadas.cantidadPuntosCuadrante(1));
        System.out.println("Cantidad de puntos en el segundo cuadrante
                + coordenadas.cantidadPuntosCuadrante(2));
        System.out.println("Cantidad de puntos en el tercer cuadrante:
                + coordenadas.cantidadPuntosCuadrante(3));
        System.out.println("Cantidad de puntos en el cuarto cuadrante:
                + coordenadas.cantidadPuntosCuadrante(4));
    }
}
```

La clase interna puede acceder a los atributos de la clase contenedora, veamos un ejercicio donde se muestra su acceso.

Problema:

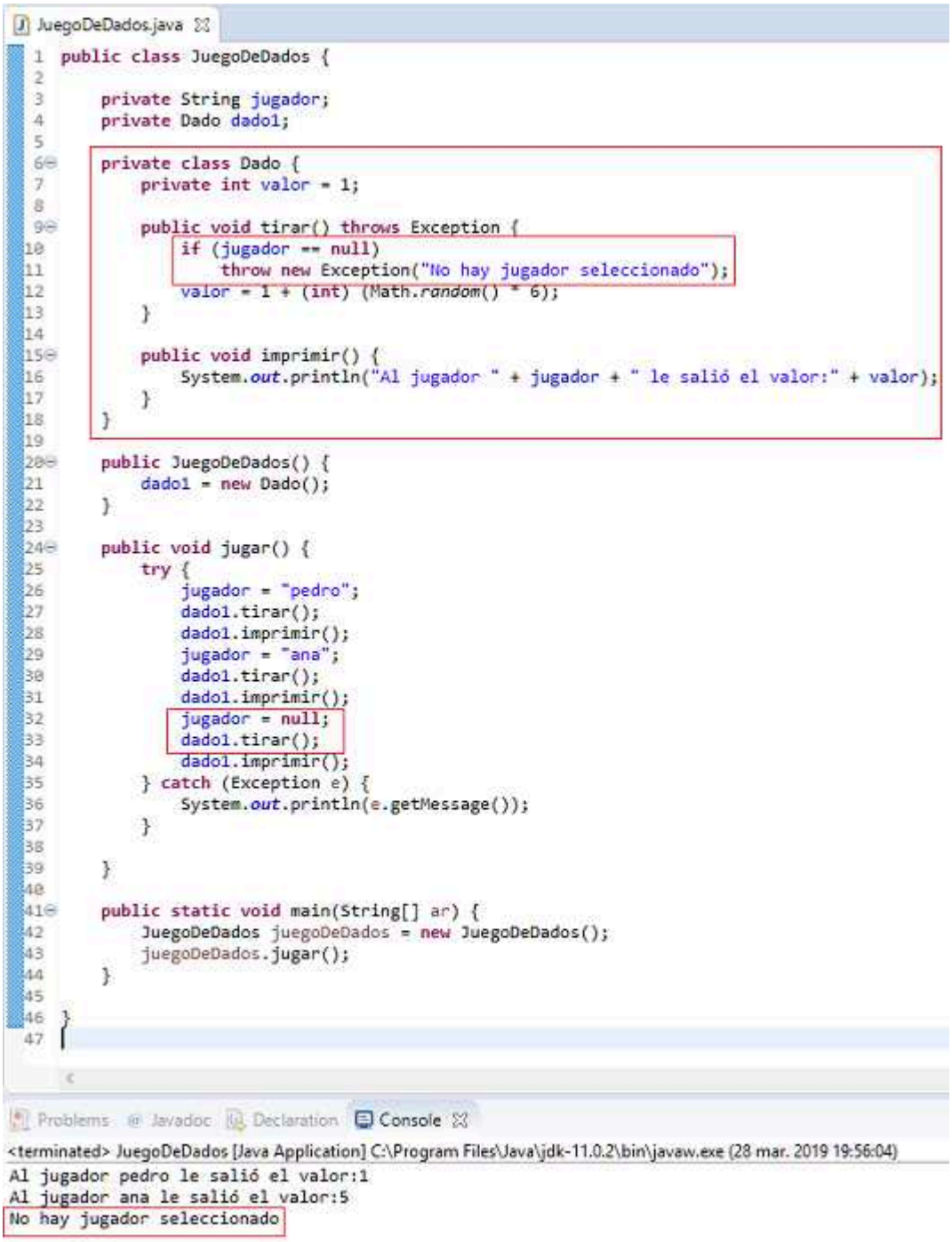
Confeccionar una clase llamada JuegoDeDados que contenga una clase anidada interna llamada Dado.
La clase JuegoDeDados tiene como atributo el nombre del jugador que tirará el dado y un objeto de la clase Dado.

Cada vez que se tire un dado la clase Dado debe verificar que el atributo 'jugador' de la clase externa tenga el nombre de una persona.

Clase: JuegoDeDados

```
public class JuegoDeDados {

    private String jugador;
    private Dado dado1;

    private class Dado {
        private int valor = 1;

        public void tirar() throws Exception {
            if (jugador == null)
                throw new Exception("No hay jugador seleccionado");
            valor = 1 + (int) (Math.random() * 6);
        }

        public void imprimir() {
            System.out.println("Al jugador " + jugador + " le salió
        }
    }

    public JuegoDeDados() {
        dado1 = new Dado();
    }

    public void jugar() {
        try {
            jugador = "pedro";
            dado1.tirar();
```

Si ejecutamos la aplicación podemos observar que se genera la excepción cuando no hay seleccionado un nombre de jugador:



La clase Dado puede consultar el valor del atributo 'jugador' de la clase 'JuegoDeDados':

```
public void tirar() throws Exception {
    if (jugador == null)
        throw new Exception("No hay jugador seleccionado");
```

Es importante tener en cuenta que la clase JuegoDeDados no puede acceder directamente a los atributos de la clase interna, sino a través de un objeto de la misma.

Creación de un objeto de la clase interna desde fuera de la clase externa.

Podemos crear un objeto de la clase interna desde fuera de la clase externa siempre y cuando la clase interna sea visible a nivel de paquete (no se especifica modificador de acceso), public o protected:

Clases: Externa e Interna

```
public class Externa {

    public class Interna {
        public void imprimir() {
            System.out.println("Clase interna");
        }
    }

    public void imprimir() {
        System.out.println("Clase externa");
    }
}
```

Clase: PruebaClaseInterna

```
public class PruebaClaseInterna {

    public static void main(String[] args) {
        Externa externa=new Externa();
        Externa.Interna interna=externa.new Interna();
        interna.imprimir();
        externa.imprimir();
    }

}
```

Para crear un objeto de la clase 'Interna' primero debemos crear un objeto de la clase 'Externa':

```
Externa externa=new Externa();
```

Ahora podemos definir un objeto de la clase 'Interna' mediante la sintaxis:

```
Externa.Interna interna=externa.new Interna();
```

Clase anidada estática.

En Java podemos definir clases internas con el modificador 'static'. Luego la clase interna se comporta como una clase normal de Java con la salvedad que se encuentra dentro de otra.

Para crear un objeto de la clase interna tenemos que utilizar la siguiente sintaxis:

Clases: Externa e Interna

```
public class Externa {

    public static class Interna {
        public void imprimir() {
            System.out.println("Clase interna estática");
        }
    }

    public void imprimir() {
        System.out.println("Clase externa");
    }
}
```

Clase: PruebaClaseInterna

```
public class PruebaClaseInterna {

    public static void main(String[] args) {
        Externa.Interna interna = new Externa.Interna();
        interna.imprimir();
    }

}
```

Cuando declaramos una clase interna 'static' luego podemos crear objetos de la misma en forma independiente a la clase externa:

```
Externa.Interna interna = new Externa.Interna();
interna.imprimir();
```

Es importante tener en cuenta que una clase interna estática solo puede acceder a los atributos y métodos estáticos de la clase que la contiene y no a los atributos de instancia como vimos en las clases no estáticas.

Clase local.

El lenguaje Java permite declarar una clase local a un método o inclusive a un bloque dentro de un método.

Clases: Externa y Local

```
public class Externa {
    public void imprimir() {
        System.out.println("Comienzo del método imprimir de la clase E
```

```
        class Local {
            public void imprimir() {
                System.out.println("Método imprimir de la clase Local.
            }
        }
        Local local1 = new Local();
        local1.imprimir();
        System.out.println("Fin del método imprimir de la clase Exterr
    }

    public static void main(String[] ar) {
        Externa externa1 = new Externa();
        externa1.imprimir();
    }
}
```

La clase 'Local' se encuentra declarada dentro del método 'imprimir' de la clase 'Externa'. Luego solo en dicho método podemos crear objetos de dicha clase local.

La clase local tiene acceso a los métodos y atributos de la clase externa, variables locales y parámetros del método donde se la declara:

Clases: Externa y Local

```
public class Externa {
    int atributo1 = 10;

    public void imprimir(String parametro) {
        System.out.println("Comienzo del método imprimir de la clase E
        int variablelocal = 4;
        class Local {
            public void imprimir() {
                System.out.println("Método imprimir de la clase Local.
                System.out.println(atributo1);
                System.out.println(parametro);
                System.out.println(variablelocal);
            }
        }
        Local local1 = new Local();
        local1.imprimir();
        System.out.println("Fin del método imprimir de la clase Exterr
    }

    public static void main(String[] ar) {
        Externa externa1 = new Externa();
        externa1.imprimir("Prueba");
    }
}
```

Clase anónima.

Las clases anónimas en Java son clases anidadas sin un nombre de clase. Normalmente se declaran como una subclase de una clase existente o como la implementación de una interfaz:

Ejemplo

```
public class PruebaClaseAnonima {

    abstract class A {
        public abstract void imprimir();
    }

    interface B {
        void imprimir();
    }

    public void probar() {
        (new A() {
            public void imprimir() {
                System.out.println("Clase");
            }
        }).imprimir();

        (new B() {
            public void imprimir() {
                System.out.println("Interface");
            }
        }).imprimir();
    }

    public static void main(String[] ar) {
        PruebaClaseAnonima p = new PruebaClaseAnonima();
        p.probar();
    }
```

En el método probar creamos dos clases anónimas. Primero creamos una clase anónima que hereda de la clase A e implementa su método abstracto:

```
        (new A() {
            public void imprimir() {
                System.out.println("Clase");
            }
        }).imprimir();
```

A partir del objeto que se crea de la clase anónima llamamos al método imprimir.

De forma similar podemos crear una clase anónima implementando una interfaz:

```
        (new B() {
            public void imprimir() {
```

```
                System.out.println("Interface");
            }
        }).imprimir();
```

En conceptos anteriores utilizamos clases anónimas para la captura del click de botones. Veamos un ejemplo donde cada vez que se presiona un botón se incrementa en uno la etiqueta del mismo:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;

public class FormularioBoton extends JFrame {
    private JButton boton1;

    public FormularioBoton() {
        setLayout(null);
        boton1 = new JButton("0");
        boton1.setBounds(40, 20, 100, 50);
        add(boton1);
        boton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int valor = Integer.parseInt(boton1.getText());
                valor++;
                boton1.setText(String.valueOf(valor));
            }
        });
    }

    public static void main(String[] args) {
        FormularioBoton fb = new FormularioBoton();
        fb.setBounds(0, 0, 200, 120);
        fb.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        fb.setVisible(true);
```

Al método addActionListener le pasamos la referencia de un objeto de una clase anónima que implementa la interfaz ActionListener:

```
        boton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int valor = Integer.parseInt(boton1.getText());
                valor++;
                boton1.setText(String.valueOf(valor));
            }
        });
```

El código que se requiere con una clase anónima es mucho más compacto que declarar una clase concreta:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;

public class FormularioBoton extends JFrame {
    private JButton boton1;

    public FormularioBoton() {
        setLayout(null);
        boton1 = new JButton("0");
        boton1.setBounds(40, 20, 100, 50);
        add(boton1);
        boton1.addActionListener(new EscuchaPresionBoton());
    }

    class EscuchaPresionBoton implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            int valor = Integer.parseInt(boton1.getText());
            valor++;
            boton1.setText(String.valueOf(valor));
        }

    }

    public static void main(String[] args) {
        FormularioBoton fb = new FormularioBoton();
```

Retornar