36019475. CSIFC03. MP0485. Programación

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E ORDENACIÓN UNIVERSITARIA

NAVEGACIÓN

Indicacións e Explicacións Á solución

A tarefa pide que desenvolvamos as clases necesarias para controlar un enchufe (con operacións de apagado e encendido), unha bombilla normal (con operacions de apagado e encendido) e unha bombilla de cores (con operacións de apagado, encendido, cambio de brillo e cambio de cor).

As clases que xa temos programadas nos ofrecen a seguinte funcionalidade:

Domotica.java

Esta clase nos ofrece o API básico para controlar un obxecto domótico, a operación principal é o **control** do dispositivo, que está sen implementar (l*anza unha excepción de operación non soportada*) xa que cada dispositivo ofrecerá un menú de control distinto. Os demáis métodos están xa implementados. Os métodos son:

- **void control(void) throws DomoticaException;** // Se supon que debe visualizar o menu de control do dispositivo
- **boolean getStatus() throws DomoticaException;** // Devolve o estado do dispositivo (true = On, false = Off)
- **int getColor() throws DomoticaException;** // Devolve o color do dispositivo
- **int getValue() throws DomoticaException;** // Devolve o valor interno do dispositivo (como a intensidade)
- **void up() throws DomoticaException;** // Aumenta en 1 o valor interno do dispositivo
- **void dn() throws DomoticaException;** // Disminúe en 1 o valor interno do dispositivo
- **void color(int col) throws DomoticaException;** // Pon o color indicado ao dispositivo

Como vemos todos estes métodos lanzan unha excepción en caso de erro...

DomoticaException.java

Esta clase extende (hereda) Exception, e se utilizará para lanzar as excepcións correspondentes aos posibles erros dos aparellos domóticos. Si observamos a clase, vemos que as excepcións de este tipo se constrúen a partir dun códgio de erro en lugar de unha mensaxe.

Connection.java

É o 'driver' do dispositivo, o encargado de conectar co dispositivo e realizar a operación que lle mande o obxecto da clase Domotica correspondente. O API ofrecido (métodos utilizables dende o exterior) e similar ao da clase Domotica, pero únicamente ten implementados os métodos on() e off(), xa que é un "driver xenérico" cunha capacidade limitada.

Neste punto, convén fixarse que os distintos métodos de Connection NON SON PÚBLICOS, se non que únicamente se poden utilizar dende clases do mesmo paquete. Polo tanto non se poden chamar dende a aplicación principal, que debería estar fora do paquete. A aplicación principal únicamente poderá facer uso dos métodos da clase Domotica.

Vamos agora a centrarnos no problema:

1.- Crear un obxecto "Enchufe".

Un obxecto "Enchufe" únicamente se podrá conectar e desconectar, co que basta coa funcionalidade ofrecida pola clase **Connection**. Heredaremos da clase Domotica para que ofreza un menú capaz de controlar un enchufe, e para que utilice un obxecto da clase Connection para o seu control:

```
package com.iesrodeira.domotica;

import java.util.Scanner;

public class DomoticaEnchufe extends Domotica {

  public DomoticaEnchufe(String nome,String ip) {
      super(nome,ip);
  }

  protected Connection getConnection() throws DomoticaException {
     return new Connection(ip);
   }
```

```
    public void control() throws DomoticaException {
        try {
            Scanner scn=new Scanner(System.in);
            System.out.println("Operacións para "+nome+": ");
            System.out.println("1.- Encender");
            System.out.println("2.- Apagar");
            String opc=scn.nextLine();
            int op=Integer.parseInt(opc);
            switch(op) {
              case 1: on(); break;
              case 2: off(); break;
              default: throw new DomoticaException(DomError.NOTSUPPORTED);
            }
    } catch (NumberFormatException e) {
        throw new DomoticaException(DomError.NOTSUPPORTED);
    }
  }
}
```

2.- Crear unha "Bombilla Normal"

Unha "bombilla normal" únicamente poden encenderse e apagarse, igual que un enchufe (na solución publicada poden tamén subir e baixar de intensidade, fallo meu). Supoñendo que tanto o Enchufe como a Bombilla utilizan o mesmo 'protocolo' o "driver" Connection xenérico sirve, e a clase Domotica anterior tamén podría valer xa que a funcionalidade é a mesma. O ideal neste caso é simplemente chamarlle á clase anterior dun modo neutro e utilizar a mesma clase para o punto 1 e 2.

Digamos que podemos renomear a clase anterior a **DomoticaOnOff** e utilizala para controlar tanto a bombilla como o enchufe.

Esto non coincide coa versión publicada, xa que na versión publicada a "Bombilla Normal" tamén é capaz de cambiar de intensidade....

3.- Crear unha "Bombilla de Cores"

Unha "Bombilla de Cores" pode encenderse, apagarse, variar a súa intensidade ou cambiar de cor. Polo tanto a clase DomoticaOnOff non ofrece a funcionalidade suficiente e o driver xenérico Connection tampouco nos ofrece o necesario. Necesitamos crear unha nova clase Domotica capaz de soportar as bombillas de cores co seu Connection correspondente:

```
package com.iesrodeira.domotica;

import java.util.Scanner;

public class DomoticaLampColor extends Domotica  {
   public DomoticaLampColor(String nome,String ip) {
      super(nome,ip);
   }

   protected Connection getConnection() throws DomoticaException {

      return new ConnectionLamp(ip);
   }

   public void control() throws DomoticaException {

      try {
         Scanner scn=new Scanner(System.in);
         System.out.println("Operacións para "+nome+": ");
         System.out.println("1.- Encender");
         System.out.println("2.- Apagar");
         System.out.println("3.- Subir Intensidade");
         System.out.println("4.- Baixar Intensidade");
         System.out.println("5.- Cambiar Color");
         String opc=scn.nextLine();
         int op=Integer.parseInt(opc);
         switch(op) {
           case 1:  on();  break;
           case 2:  off();  break;
```

```
            case 3:  up();  break;
            case 4:  dn(); break;
            case 5:
               try {
                  System.out.println("Cor Desexada? : ");
                  int value=Integer.parseInt(scn.nextLine());
                  color(value);
               } catch (NumberFormatException e) {
                  System.out.println("Debes introducir un número enteiro");
               }
               break;

            default: throw new DomoticaException(DomError.NOTSUPPORTED);
         }
      } catch (NumberFormatException e) {
           throw new DomoticaException(DomError.NOTSUPPORTED);
      }
   }
}
```

O driver sería:

```
package com.iesrodeira.domotica;

class ConnectionLamp extends Connection {

   private final int MAXVALUE=10;
   private final int MINVALUE=0;
   private int value;
   private int color;

   /**

    Nun caso real, este constructor conectaría co dispositivo ip e obtería o seu          estado actual
   */
   ConnectionLamp(String ip) {
      super(ip);
      this.name="Lamp Rodeira Domotics Device";
      this.value=0; // Nun caso real obtería o valor do dispositivo....
      this.color=0; // Nun caso real obtería o valor do dispositivo...
      System.out.println("Conectado a "+this.name);
   }

   void up() throws DomoticaException {

     if (this.value < MAXVALUE) this.value++;
    else throw new DomoticaException(DomError.ERRVALUE);
    System.out.println("Subida a intensidade a "+this.value);
   }

   void dn() throws DomoticaException {

     if (this.value > MINVALUE) this.value--;
    else throw new DomoticaException(DomError.ERRVALUE);
    System.out.println("Baixada a intensidade a "+this.value);
   }

   void setColor(int value) throws DomoticaException {

     this.color=value; // Nun caso real se conectaría co dispositivo e configuraría        a súa cor
     System.out.println("Color posto a "+value);
   }

   int getColor() throws DomoticaException {

     return this.color;
   }

   int getValue() throws DomoticaException {
```

```
        return this.value;
    }
}
```

Unha vez desenvoltas as clases, poderei escribir o programa:

3.- Programa Principal

O programa principal debe permitir controlar os dispositivos anteriores. Esta versión é algo máis elaborada que a subida como solución, espero que non teñades dificultade en comprendela:

```
import com.iesrodeira.domotica.*;

import java.util.Scanner;

class ControlDomotico {
    public static void main(String[] args) {
        Scanner scn=new Scanner(System.in);
        int op=0;

        Domotica[] devices= {
              new DomoticaOnOff("Enchufe Salon","172.20.12.17"),
              new DomoticaOnOff("Lampara Habitación","172.20.12.18"),
               new DomoticaLampColor("Lámpara Salón","172.20.12.20")
         };

        do {
            System.out.println("1.- Control Enchufe Salón");
            System.out.println("2.- Lámpara Habitación");
            System.out.println("3.- Lámpara Salón");
            System.out.println("4.- Saír");
            System.out.println("Elixe opcion: ");
            String opcstr=scn.nextLine();
            try {
               op=Integer.parseInt(opcstr);
               switch (op) {
                  case 1: case 2: case 3: devices[op-1].control(); break;
                }
            } catch (NumberFormatException e) {
                op=0;
            } catch (DomoticaException e) {
                System.out.println(e.getMessage());
            }

        } while (op!=4);

    }
}
```

Si probades a aplicación veredes que si subides varias veces a intensidade da lámpara, sempre a pon a 1. Esto nun caso "real" non pasaría, xa que o valor actual o collería do dispositivo. Neste exemplo ocurre, porque cada vez que nos eleximos controlar un dispositivo creamos un **novo** obxecto Connection, que ten os valores iniciais. Unha solución para unha mellor "simulación" neste caso concreto sería facer os atributos value e color ***static.*** Facede a proba....

Espero que agora se entenda ben, se non, xa sabedes. Usade o foro.

Última modificación: Mércores, 14 de Decembro do 2016, 12:42