PRACTICO N° 10
Alternativas de Diseño

En todas las soluciones considere la importancia de diseñar algoritmos correctos, eficientes y legibles seleccionando estructuras de control adecuadas. En todas las implementaciones puede agregar métodos para favorecer la modularización.

EJERCICIO 1. La clase ConsumoEnergia permite almacenar el consumo de energía eléctrica de una fábrica en cada uno de los días de un período de tiempo.

ConsumoEnergia
<<atributos de instancia>> CE : entero[]
<<constructores>> ConsumoEnergia (n : entero) <<comandos>> establecerConsumoDia (c:entero,d:entero) <<consultas>> cantDias (): entero obtenerConsumoDia (d:entero): entero mayorConsumo():entero diaMayorConsumo():entero totalConsumo():entero hayMayorConsumo(t: entero): boolean todasMayores(t:entero): boolean cuantasMayores (t:entero) : entero cuantasCercanas (t: entero) : entero primerDiaMayor(t: entero):entero ultimoDiaMayor(t: entero): entero mMayores(t,m:entero):boolean

obtenerConsumoDia (d)
Se asume que se controló que el período tiene al menos d días

- boolean hayMayorConsumo(int t): Determina si hubo consumos mayores a t
- boolean todasMayores(int t): Determina si todos los consumos son mayores a t
- int cuantasMayores (int t) : Calcula cuántos consumos superaron el valor t
- int cuantasCercanas (int t) : Calcula cuántos consumos difieren de t en menos de 3.
- int primerDiaMayor(int t): Retorna el primer día que registra un valor mayor a t.
- int ultimoDiaMayor(int t): Retorna el último día que registra un valor mayor a t.
- boolean mMayores(int t, int m): Determina si hay m valores consecutivos mayores a t.

a. Implemente la clase ConsumoEnergía modelada en el diagrama, considerando la funcionalidad de los servicios descripta.
b. Verifique los servicios de la clase ConsumoEnergia ejecutando y luego extendiendo la siguiente clase tester[1]:

```
class testerCE {
public static void main (String args[]){
  ConsumoEnergia ceFabrica = new ConsumoEnergia(30);
    //Inicializa el primer y el ultimo elemento
  ceFabrica.establecerConsumoDia(20,0);
  ceFabrica.establecerConsumoDia(22,29);
    //inicializa los demás elementos
```

1 La clase testerCE se encuentra disponible en la página de la materia, en el apartado “Material Adicional”.

```
  for (int dia=2; dia<ceFabrica.cantDias()-1; dia=dia+2){
    ceFabrica.establecerConsumoDia(dia*3,dia);
    }
  for (int dia=ceFabrica.cantDias()-3;dia > 0; dia=dia-2){
    ceFabrica.establecerConsumoDia(dia*4,dia);
  }
  mostrar(ceFabrica);
  verificar(ceFabrica);
  //Casos de prueba: El menor es el primero, el mayor el ultimo

  ceFabrica.establecerConsumoDia(ceFabrica.mayorConsumo()+2,29);
  mostrar(ceFabrica);
  verificar(ceFabrica);
  System.out.println();
  //Casos de prueba: El menor es el anteultimo, el mayor el segundo
  ceFabrica.establecerConsumoDia(ceFabrica.mayorConsumo()+3,1);
  mostrar(ceFabrica);
  verificar(ceFabrica);
}
  public static void mostrar(ConsumoEnergia ceFabrica){
   for (int i=0;i< ceFabrica.cantDias();i++)
      System.out.println("Dia "+i+" consumo "+ceFabrica.obtenerConsumoDia(i));
      System.out.println();
    }
  public static void verificar(ConsumoEnergia ceFabrica){
     int tc = ceFabrica.totalConsumo();
    System.out.println("Total Consumo "+tc);
    int promedio = tc/ceFabrica.cantDias();
    System.out.println("Promedio Consumo "+promedio);
    int mayor = ceFabrica.mayorConsumo();
    System.out.println ("Dia mayor consumo "+ceFabrica.diaMayorConsumo()
                        +" Mayor Consumo "+mayor);

   if (ceFabrica.hayMayorConsumo(promedio))
           System.out.println ("Hay consumos mayores a   "+promedio);
   else    System.out.println ("No hay consumos mayores a "+promedio);

   if (ceFabrica.hayMayorConsumo(mayor))
          System.out.println ("Hay consumos mayores a "+mayor);
   else   System.out.println ("No hay consumos mayores a "+mayor);
}
}
```

EJERCICIO 2. La clase ConsumoEnergiaP permite almacenar el consumo de energía eléctrica de una fábrica en cada uno de los días de un período de tiempo tal como se modela en el ejercicio 1, pero **desde las clases cliente de ConsumoEnergíaP los días se referencian de 1 en adelante.**

- **a)** Implemente la clase ConsumoEnergiaP
- **b)** Implemente una clase tester que lea una secuencia de valores de un archivo representando los consumos de un período.

EJERCICIO 3. La clase ConsumoDosPlantas permite almacenar el consumo de energía eléctrica de dos plantas de una misma fábrica en cada uno de los días de un período. Las plantas se identifican con números 1 y 2.

a. Implemente la clase ConsumoDosPlantas modelada en el diagrama.
b. Implemente la clase tester para verificar los servicios.

ConsumoDosPlantas
<<atributos de instancia>> CE1 : entero[] CE2 : entero[]
<<constructores>> ConsumoDosPlantas (n : entero) <<comandos>> establecerConsumoDia (d,p,c:entero) <<consultas>> obtenerConsumoDia (d,p:entero): entero obtenerConsumoDia (d:entero): entero totalConsumo():entero totalConsumoPlanta(p:entero):entero hayMayorConsumo(t: entero): boolean todasMayores(t:entero): boolean cuantasMayores (t:entero) : entero cuantasCercanas(t:entero):entero diaMayorConsumo():entero diaMayorConsumo(p:entero):entero plantaMayorConsumo():entero plantaMayorConsumo(d:entero):entero mMayores(t,m:entero): boolean consumoFabrica():ConsumoEnergia

obtenerConsumoDia (d,p)
Se asume que se controló que el período tiene al menos *d* días y *p* es 1 o 2

- int obtenerConsumoDia(int d,int p) retorna el consumo de una planta en un día.
- int obtenerConsumoDia(int d) retorna el consumo total de las dos plantas en un día.
- int totalConsumo() computa el consumo total de las dos plantas en el periodo completo.
- int totalConsumoPlanta(int p) retorna el consumo de la planta p en todo el período.
- boolean hayMayorConsumo(int t) Determina si hubo consumos mayores a t considerando el consumo total de la fábrica en cada día.
- boolean todasMayores(int t) Determina si todos los consumos son mayores a t considerando el consumo total de la fábrica en cada día.
- int cuantasMayores (int t) Calcula cuántos consumos superaron el valor t considerando el consumo total de la fábrica en cada día.
- int cuantasCercanas (int t) Calcula cuántos consumos difieren de t en menos de 3 considerando el consumo total de la fábrica en cada día.
- int diaMayorConsumo() retorna el número de día en el que se produjo el mayor consumo total.
- int diaMayorConsumo(p:entero) retorna el número de día con mayor consumo en la planta p.
- int plantaMayorConsumo() retorna 1 o 2 según la planta que ha tenido mayor consumo en el período.
- int plantaMayorConsumo(d:entero) retorna 1 o 2 según la planta que ha tenido mayor consumo en el día d
- boolean mMayores(int t,int m) Determina si hubo m días consecutivos con consumo total mayor a t.
- ConsumoEnergia consumoFabrica() Genera un objeto de clase ConsumoEnergia en el cual la componente de cada día es la suma del consumo de las dos plantas en ese día.

Siempre que sea posible utilice los métodos de la interfaz pública para abstraerse de la forma en la que se implementa el estado interno.

EJERCICIO 4. Una alternativa para modelar el consumo de dos plantas en un período es definir un arreglo de dos dimensiones.

a. Implemente la clase ConsumoDosPlantas considerando la misma funcionalidad, pero modificando la representación del atributo de instancia CP.
b. Defina casos de prueba adecuados para cada uno de los servicios e implemente una clase tester .

ConsumoDosPlantas
<<atributos de instancia>> CP : entero[] []

EJERCICIO 5. Una alternativa para modelar el consumo de energía de dos plantas es definir en la clase Cliente dos objetos de la clase ConsumoEnergia definida en el ejercicio 1.

Escriba los métodos leerDatos, mostrar, consumoTotal, diaMaximoConsumoTotal consideando la siguiente definición de la clase Fábrica:

```
class Fabrica {
public static void main (String args[]){
  ConsumoEnergia planta1 = new ConsumoEnergia(30);
  ConsumoEnergia planta2 = new ConsumoEnergia(30);
  //Inicializa el primer y el ultimo elemento
  leerDatos (planta1);
  leerDatos (planta2);
  t = consumoTotal (planta1,planta2);
  System.out.println ("El consumo total de la fábrica es "+t);
  d= diaMaximoConsumoTotal(planta1,planta2);
  System.out.println ("El dia de mayor consumo total fue "+d);
}…
}
```

EJERCICIO 6. La clase ConsumoServicios permite almacenar el consumo de gas, agua y energía eléctrica de una fábrica en cada uno de los días de un período.

ConsumoServicios
<<atributos de instancia>> CE : Servicio []
<<constructores>> ConsumoServicios (n : entero) <<comandos>> establecerConsumoDia (s:Servicio, d:entero) anularConsumoDia (d:entero) <<consultas>> cantDias (): entero obtenerConsumoDia (d:entero): Servicio huboMayores(s:Servicio): boolean todasMayores(s:Servicio): boolean algunoMayor(s:Servicio) :boolean cuantasMayores (s:Servicio) : entero

Servicio
gas:entero agua:entero eElectrica:entero
<<constructores>> Servicio(g,a,ee :entero)

establecerConsumoDia(s,d), *anularConsumoDia(d)* y *obtenerConsumoDia(d)*
Se asume que se controló que el período tiene al menos d días.

- anularConsumoDia (int d) asigna el valor nulo a la referencia que corresponde al día d
- boolean huboMayores(Servicio s): Determinar si hubo días con consumos superiores a s en los tres servicios.
- boolean todasMayores(Servicio s): Determinar si todos los días tuvieron consumos superiores a s en los tres servicios.
- boolean algunoMayor (Servicio s): Determinar si todos los días tuvieron consumos superiores a s en al menos uno de los servicios.
- int cuantasMayores (Servicio s): Calcula cuántos días superaron el consumo s en los tres servicios.

a. Implemente la clase ConsumoServicios modelada en el diagrama, considerando la funcionalidad de los servicios descripta.
b. Establezca un conjunto de casos de prueba y verifique los servicios de la clase ConsumoServicios definiendo una clase tester para esos valores.

Objetivos del práctico

Definir una clase que encapsula:

- un arreglo de una dimensión de componentes elementales
- dos arreglos de una dimensión de componentes elementales
- un arreglo de dos dimensiones de componentes elementales

Alternativas de representación.
Enfatizar recorridos exhaustivos y no exhaustivos.
Verificar utilizando casos de prueba.
Eficiencia y legibilidad.