

Agenda

- Recapitulación del diseño orientado a objetos
- Atributo *Static* y método *Static*
- Diferencia entre *array* y *vector*
- Preparación del boletín de problemas nº 3
 - Analizar el problema 1
 - Explicar el problema 2

Guía en 5 pasos para el diseño orientado a objetos

- Paso 1: leer la sentencia del problema e identificar las clases
 - Las clases son nombres
 - Cada clase contiene un conjunto de atributos y métodos
- Paso 2: identificar los atributos de cada clase
 - Los atributos también son nombres
 - ¿Un atributo es privado o público?
 - ¿Debería ser estático?
 - Si un atributo es privado, ¿deberíamos otorgarle acceso a los métodos (getX() y setX())?

Guía en 5 pasos para el diseño orientado a objetos

- **Paso 3: escribir el constructor**
 - La finalidad de un constructor es crear una instancia de la clase utilizando los valores facilitados por el usuario (a través de los argumentos del constructor)
 - Debe ser público y NO tener ningún tipo de devolución
 - Una clase puede contener más de un constructor
 - Una clase decide a qué constructor llamar según el número de argumentos pasados al constructor
- **Paso 4: definir métodos especiales**
 - Alcance
 - Tipo de devolución
 - Nombre
 - Argumentos y sus tipos de datos

Guía en 5 pasos para el diseño orientado a objetos

- **Paso 5: crear la clase del usuario**
 - La clase del usuario es la única que contiene la función main()
 - La finalidad principal de la clase del usuario es servir de interfaz con el usuario (esto es, aceptar datos de entrada) y administrar las interacciones entre las clases

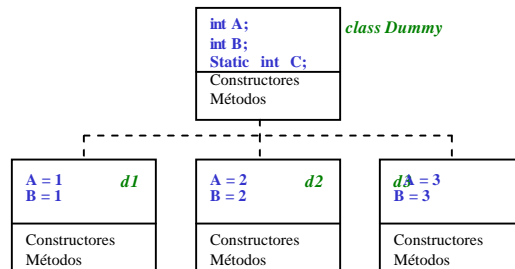
```
public static void main(String[] args) {
    String text= JOptionPane.showInputDialog("Escriba el piso más alto: ");
    int maxFloor= Integer.parseInt(text);
    text= JOptionPane.showInputDialog("Escriba el piso más bajo: ");
    int minFloor= Integer.parseInt(text);

    Elevator one= new Elevator(maxFloor, minFloor, "One");
    Elevator two= new Elevator(maxFloor, minFloor, "Two");
    Elevator three= new Elevator(maxFloor, minFloor, "Three");

    Controller master= new Controller(one, two, three);
    .....
}
```

Atributo *Static* y método *Static*

- Cada vez que se crea la instancia de una clase, el ordenador crea una copia de todas las variables
- No obstante, independientemente del número de objetos que haya creado, sólo hay 1 copia de las variables estáticas en la memoria



Arrays y vectores

	<i>Array</i>	<i>Vector</i>
Longitud	Fija	Flexible
Tipo de datos de elementos	Deben ser iguales	Pueden ser distintos
Crea	<code>int[] Arr = new int[4]</code>	<code>Vector Vr = new Vector();</code>
Asigna valor	<code>Arr[0] = 4;</code>	<code>Vr.addElement(p0);</code>
Acc. a elemento	<code>Arr[2]</code>	<code>Vr.elementAt(2)</code>
Longitud	<code>Arr.length</code>	<code>Vr.size()</code>

Atributo *Static* y método *Static* (cont.)

- Como cada clase tiene sólo 1 copia de las variables estáticas, se debe hacer referencia a ellas mediante el **nombre de la clase, NO el nombre del objeto**

```
System.out.println(Dummy.C);
```

- Lo mismo se aplica al método *Static*

```
class Pipe {
    public static void setRoughness (double r)
    { .....}
}

class Test {
    public static void main (string[] args) {
        Pipe.setRoughness (9.8);
    }
}
```

Boletín de problemas 3, problema 1

- Dos clases: *Fluid* (fluido) y *Pipe* (conducto)

Fluid

Atributos:
String name;
double density;

Constructors
Dos argumentos
Methods
printData ()

Pipe

Atributos:
String name;
double length;
double diameter;
static double roughness;
final static double gravity = 9.8;

Constructors
Tres argumentos
Methods
printData ()
static setRoughness(...)
static printStatistics()

- Ya podemos crearlas

Boletín de problemas 3, problema 1

- Funcionalidad de la función main() de la clase del usuario

- Crea un array de 3 Fluid

```
Fluid[] substance = new Fluid[3];  
substance[0] = new Fluid("water", 1.0E3);
```

- Crea un vector de 3 Pipe

```
Vector network = new Vector ();  
Pipe p0 = new Pipe("AB", 50.0, 0.5);  
network.addElement(p0);
```

¿Hay alguna forma mejor de hacerlo?

- Define la aspereza de la clase Pipe

```
Pipe.setRoughness(0.0001);
```

- Muestra los datos

Boletín de problemas 3, problema 2

- Clase Pipe

Atributo	Regular	Estático	Final	Calculado
D	x			
L	x			
z	x			
r	x			
v		x(1.0)		
q	x			
R				x
f				x
g = 9,8		x	x	
h		x(.00001)		
Pb				x

Boletín de problemas 3, problema 2

- Salida (“water”, 100000, 1, 0.00001)

Conducto AB

Número de Reynolds: 500000.0

Fricción: 0.013030745525640063

Presión en la entrada: 100000.0

Presión a la salida: 100328.462723718

Caída de presión: 328.4627237179957

Tasa de flujo de volumen: 0.19634954084936207

Conducto BC

.....

Bol. problemas 3, prob. 2: una solución

- Reto de diseño: calcular la presión
- *Pipes* y *Fluids* son clases independientes; sin embargo, el cálculo de la presión depende de ambas y la presión de entrada depende del conducto de subida. Esto significa que las presiones de los conductos se deben calcular en un orden concreto.
- Sugerencia: haga que los conductos incluyan referencias a sus conductos de salida (con un vector, por ejemplo)
- Piense en los conductos como si estuviesen “vacíos” (sin flujo ni fluido) o “lentos” (con flujo de fluido)

Bol. problemas 3, prob. 2: una solución

```
class Pipe {
  private Vector outflows; /* def. mediante addOutflow() */
  public void startFlow(Fluid f, double v, double
    inputPressure) {
    /* la presión se calcula aquí */
    ...
    /* calcula la presión de todas las salidas */
    for (int i=0; i<outflows.size(); i++) {
      Pipe outPipe = (Pipe)outflows.elementAt(i);
      outPipe.startFlow(f, v, outputPressure);
    }
  }
  ...
  .
  .
}
```

Llama a startFlow() en el primero conducto de la red sólo para calcular las presiones de todos los conductos.

Bol. problemas 3, prob. 2: una solución

- Éste es sólo un diseño concreto pero hay muchos otros posibles
- Esta solución delega todo el cálculo de presión en el método startFlow(), que sólo necesita ser llamado una vez