

1.00Clase 8

Datos y métodos estáticos

Datos y métodos estáticos de las clases

- **Campos de datos estáticos:**
 - Sólo una instancia de elemento de datos para toda una clase.
 - Ninguna por objeto.
 - “Estático” es un término clave histórico utilizado en CyC++.
 - “Campos de datos de una clase” es un término más apropiado.
 - A diferencia de los “campos de datos de la instancia” (por objeto).
- **Métodos estáticos:**
 - No funcionan con objetos y no utilizan ningún objeto específico.
 - Pueden acceder únicamente a los campos de datos estáticos de la clase.
 - No pueden acceder a los campos de instancia de los objetos.
 - “Métodos de la clase” es un término más adecuado.
 - A diferencia de los “métodos de instancia” (que operan con un objeto).

Ejemplos de campos de datos estáticos

- Variables de las que existen verdaderamente sólo una por clase.
 - Por ejemplo, el siguiente número ID (identificador), disponible para todos los estudiantes del MIT (si suponemos que éstos se emiten secuencialmente). Añadir a una clase *Student* (estudiante):

```
private static int nextID;           // 1 valor de clase
private int ID;                       // Cada instancia
public static void getID() { return nextID++;}
```
- Constantes utilizadas por una clase (palabra clave `final`).
 - Hay una por clase; no hace falta una en cada objeto.

```
public static final double TEMP_CONVERT= 1.8;
```
 - Si está en la clase *Temperature* (temperatura), se invoca mediante:

```
Double t= Temperature.TEMP_CONVERT * temp + ...;
```
 - Las constantes tradicionalmente van en mayúsculas (C, C++).
 - (Las variables estáticas son distintas en C, C++).

Ejemplos de métodos estáticos

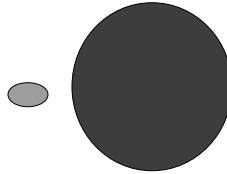
- Para métodos que utilizan sólo los argumentos y , por tanto, no necesitan un objeto.

```
public static double pow(double b, double p)
// Librería Math, eleva b a la potencia p
```
- Para métodos que sólo requieren campos de datos estáticos.

```
public static void getID() { return nextID++;}
// nextID es una variable estática (véase página anterior)
```
- Método *Main* dentro de la clase donde se inicia el programa:
 - No existen aún objetos con los que éste pueda operar.
- (Todos los métodos de C son como los métodos estáticos de Java, dado que C no posee clases / objetos; C++ posee métodos como los de Java y como los de C).

Ejemplo:Flujo de calor

- Dos cuerpos próximos , a distinta temperatura, intercambiarán energía por radiación:
 - La ley de Boltzmann se utiliza para calcular la transferencia de calor.
- En nuestro ejemplo, un objeto es pequeño y templado y el otro es muy grande y frío.



Ejemplo,2ª parte

- Los objetos poseen temperatura, área de superficie y emisividad(cuerpo gris).
- Existe una constante en la ecuación de Boltzmann.
- Asignaremos un ID a cada cuerpo gris para identificarlo a la hora de informar de los resultados:
 - Esto es un poco absurdo,pero...
- Escribiremos tanto un método miembro como un método de la clase para calcular el flujo de calor:
 - El método miembro se emplea en programas que utilizan la clase *GrayBody* que vamos a escribir.
 - El método de clase se utiliza en programas que no emplean nuestra clase.

Código de radiación, 1ª parte

```
class GrayBody {
    public static final double STEFAN_BOLTZMANN= 5.669E-8;
    private double temp;                // Kelvin
    private double surfaceArea;         // m^2
    private double surfaceEmissivity;   // 0-1
    static private int nextID= 8;       // Campo de la clase
    private int ID;                     // Campo miembro
    GrayBody(double t, double sa, double se) {
        temp= t;
        surfaceArea= sa;
        surfaceEmissivity= se;
        ID= nextID++;
    }
    public double getTemp() { return temp;}
    public double getSurfaceArea() {return surfaceArea; }
    public double getSurfaceEmissivity() {
        return surfaceEmissivity;}
    public int getID() {return ID; }
```

Código de radiación, 2ª parte

```
// Método miembro(en este ejemplo simple no se han utilizado todos los campos)
public double heatFlow(GrayBody other) {
    double ot= other.getTemp();
    return STEFAN_BOLTZMANN /
        (1+(1-surfaceEmissivity)/surfaceEmissivity)*
        surfaceArea*(temp*temp*temp*temp - ot*ot*ot*ot);
}

// Método de la clase (en este ejemplo simple no se han utilizado todos los campos)
static public double heatFlow(double t1, double sa1,
    double se1, double t2, double sa2, double se2) {
    return STEFAN_BOLTZMANN / (1+(1-se1)/se1)*
        sa1*(t1*t1*t1*t1 - t2*t2*t2*t2);
}
}

// Esta no es la ecuación completa de la ley de Boltzmann
```

Código de radiación, 3ª parte

```
public class StaticExample {
public static void main(String[] args) {

    GrayBody one= new GrayBody(315.0, 0.03, 0.9);
    GrayBody two= new GrayBody(165.0, 100000.0, 1.0);

    double heat12= one.heatFlow(two);        // Método miembro
    int id= one.getID();
    System.out.println("Object 1 ID: " + id +
        "\nFlujo de calor (miembro): " + heat12);

    double heat12x= GrayBody.heatFlow(315.0, 0.03, 0.9,
        165.0, 100000.0, 1.0);                // Método de la clase
    System.out.println("Flujo de calor (estático): " + heat12);
    System.out.println("ID del objeto 2: " + two.getID());
    System.exit(0);
}
} // ¿Sabe alguien lo que este ejemplo reproduce?
```

Descarga de la clase *Street* (calle)

A.Descargue *Street.java* y *StreetTest.java* :

- Le hace falta la ruta del directorio montado: en la vista *Explorer* (explorador), haga clic en la pestaña *FileSystems* (sistema de archivos). La ruta del directorio montado se muestra como enlace en el árbol *Filesystems* (si ha montado varios directorios, elija el superior).
- En su navegador Web, vaya a:
web.mit.edu/1.00/www/Lectures/Lecture8/Street.java
Guarde el archivo descargado en el directorio montado (haga clic con el botón derecho del ratón y seleccione "Guardar enlace como").
- Haga lo mismo con:
web.mit.edu/1.00/www/Lectures/Lecture8/StreetTest.java
- Vuelva a la vista *Explorer* y a *FileSystems*. Haga clic con el botón derecho sobre directorio montado y seleccione actualizar. Debería aparecerle el archivo recién guardado.

Descarga de la clase *Street*(2)

B. Cree un proyecto llamado `street` y añádale los archivos `street.java` y `streetTest.java` (para añadir un archivo a su proyecto, haga clic con el botón derecho del ratón sobre el archivo y seleccione *Tools-> Add to Project*; en español, Herramientas-> añadir al proyecto).

C. A continuación le mostramos un resumen de la clase *Street* que usted ha descargado:

- *laneWidth*: ancho de l carril (en pies). (En nuestro caso, todos los carriles poseen el mismo ancho= 12 pies)(1 pie = 0'305 metros).
- *medianStripeColor*: color de la mediana. (En nuestro caso, todas las líneas de la mediana son amarillas).
- *numLanes*: número de carriles.
- *speedLimit*: límite de velocidad (en mph).
- *trafficPerLane*: tráfico por carril (en vehículos/hora).

Cómo mejorar la clase *Street*

- A. Compile y ejecute su programa. La salida tendría que mostrar la información de las tres calles que se han instanciado (creado) en el método `main()` de la clase `StreetTest`.
- B. Hemos dicho anteriormente que todas las calles poseen el mismo:
- color de línea en la mediana (amarillo).
 - ancho del carril (12 pies).
- C. En la implementación actual de la clase *Street*, cada objeto *street* posee una instancia separada de los campos `medianStripeColor` y `laneWidth`. Dado que ambos campos son idénticos en todos los objetos, modifique la clase *Street* para que todos los objetos compartan los mismos campos de datos `medianStripeColor` y `laneWidth`. Asegúrese de que únicamente inicializa esos campos una vez (en vez de inicializarlos cada vez que cree un objeto).
- D. Compile y ejecute su programa. Asegúrese de que aún obtiene la misma salida que antes.

Número total de objetos *Street*

- A.** Es necesario que mantengamos un control del número total de objetos *Street* que se han creado. Almacenaremos este número dentro de un entero privado llamado `numberStreets`.
Añada este campo de datos a la clase *Street* y realice los cambios apropiados para que se incremente cada vez que se cree un nuevo objeto *street*.
- B.** En la clase *Street*, cree un método `getNumberStreets()` que devuelva el número total de calles (objetos *Street*). Debería ser capaz de llamar a este método sin haber creado ningún objeto *street*.
- C.** En la clase `streetTest`, utilice el método `getNumberStreets()` para mostrar el número total de calles (objetos *Street*) creadas.
- D.** Compile y ejecute su programa.

Cómo crear métodos nuevos

- A.** Es necesario que agotemos el comportamiento de nuestra clase *Street*. Cree tres métodos nuevos:
- `getTrafficDensity()`: devuelve la densidad del tráfico de un carril. La densidad del tráfico (vehículos/milla) se define como:
 - Tráfico por carril (veh/hora) / Límite de velocidad (mph).
 - `getLaneWidthMeters()`: devuelve el ancho del carril en mts. (1 pie = 0.3048 m.)
 - `getStreetWidth()`: devuelve el ancho total de la calle en pies. El ancho de una calle se define por:
 - Número de carriles x ancho del carril (pies) x 2 (suponemos que son calles de dos carriles).

Nota: es necesario que decida si va a declarar estos métodos como estáticos. Recuerde que un método de clase se aplica a la clase, a diferencia de cualquier instancia en concreto. El comportamiento de un método de clase no depende del estado (campos) de un objeto concreto.

Cómo crear métodos (2)

- B.** En la clase `StreetTest`, llame a estos métodos para mostrar el ancho de un carril en metros, y el ancho de la calle y la densidad del tráfico en *Mass . Ave. (Massachusetts Avenue)* .
- C.** Compile y ejecute su programa.

ID de los objetos *Street*(Opcional)

- Queremos que cada calle se identifique con un sólo ID. La clase `Street` debe facilitar un mecanismo que asigne automáticamente un único ID a cada objeto *Street* que se cree.
 - A.** Es necesario que haga lo siguiente:
 - Cree un campo llamado `nextID` que sea el siguiente número ID disponible para todas las calles u objetos *Street* (suponga que los números se emiten secuencialmente).
 - Cree un campo llamado `ID` que almacene el único ID de una calle concreta.
 - Modifique el constructor de forma que asigne un ID a la recién creada instancia *street*.

ID de los objetos *Street*(Opcional)(2)

- B.** Modifique `displayStreetInfo()` para imprimir el ID de los objetos *Street*.
- C.** Compile ejecute su proyecto. Suponiendo que comenzamos a emitir IDs (identificadores) a partir de "0", debería tener lo siguiente:
- ID de *Mass Ave*:0; ID de la calle *Vassar*:1;ID de la calle *Albany*:2
- D.** Es necesaria una opción que comience emitiendo IDsa partir de un número arbitrario determinado (por ejemplo"100").
- En la clase *Street*,añada un método llamado `setNextID(int i)` que haga que el próximo id que se emita sea igual a "i".
 - Mediante este método, modifique la clase `StreetTest` para que:
 - *Mass Ave* posea un IDde 100.
 - La calle *Vassar*posea un IDde101.
 - La calle *Albany*posea un IDde102.