

1.00Clase11

Ámbito y acceso

Ciclos de vida de las variables

- **Variables de instancia(uobjeto) :**
 - Se creancuando se crea el objeto que las contiene.
 - Se inicializan por defectosi no se hace de modo explícito :
 - 0para números, "false"para booleano,"null"para objetos.
 - Se destruyencuando el recolector de basura de Java no encuentra referencias activas para el objeto.
- **Variables estáticas(ode clase) :**
 - Se crean cuando la clase se usa por primera vez.
 - Se inicializan por defecto si no se hace de modo explícito:
 - 0 para números, "false" para booleano, "null" para objetos
 - Suelen existir para el resto del programa(salvo que no esté cargado) .
- **Variables locales (ode bloque) :**
 - Creadasenla sentencia en la que están definidas .
 - Nose inicializan por defecto.Contienen datos imprevisibles .
 - Se destruyen al salir del bloque(en la llave final).

Ámbito de la variable

- **Ámbito:** limita las partes del programa donde se define y es visible una variable o un método:
 - Evita conflictos entre variables y métodos en diferentes partes de un programa :
 - Las variables son la preocupación principal, más que los métodos .
 - Limita los efectos de los cambios al módulo o al ámbito más pequeño posible.
 - Permite que varias personas trabajen a la vez en programas grandes.
 - Permite la realización de pruebas, la corrección de errores y el mantenimiento necesario, al tiempo que limita la introducción de nuevos errores.

Ámbito: variables y métodos

- **Variables locales (en un método o bloque)**
 - Existen desde el punto de definición hasta el final del bloque
 - Los bloques se definen mediante llaves {}
 - Los bloques se suelen utilizar para definir:
 - El cuerpo del método
 - Sentencias múltiples en operaciones de tipo if-else o en forma de bucles
 - Las variables locales sólo existen dentro del propio método
 - No se da acceso a ningún otro método
 - Si la variable local y de instancia tienen el mismo nombre, la variable local oculta a la de instancia:
 - Acceda a la variable de instancia mediante:
`this.nombreVariable;`

Acceso: variables y métodos

- Las variables y métodos de instancia y estáticos (en una clase) tienen 4 modificadores de acceso:
 - Private: acceso sólo a los métodos de su propia clase
 - Los campos de datos casi siempre tienen que ser privados
 - Public: acceso a todos los métodos y clases
 - Los métodos que van a utilizar otras clases son públicos
 - Los métodos sólo para uso interno son privados
 - Package: acceso a los métodos de las clases del mismo paquete (un paquete es un grupo de clases)
 - Este es el valor por defecto. Especifique siempre el ámbito
 - No hay palabra clave 'package', es el valor por defecto sin clave
 - Protected: utilizado con herencia (lo veremos más adelante)
 - Es como una variable privada, salvo porque es visible para las clases derivadas o subclasses (y, en Java, para las otras clases del mismo paquete)

Acceso: paquetes

- Si añade, al inicio del programa:

```
package nombrePaquete;
```

 - Esto colocará las clases del archivo fuente en un paquete, junto con cualquier otro archivo fuente con la misma sentencia al principio:
 - Los paquetes se almacenan en carpetas del sistema de archivos de Forte, dentro de su PC o terminal Athena.
 - Así, estas clases tendrán acceso a los métodos y campos de datos a los que puedan acceder las otras clases del paquete.
 - Utilice la opción "New Package" (nuevo paquete) de Forte a conveniencia.
- Para utilizar el paquete en otra clase, añada al principio de la clase:

```
import nombrePaquete.*;
```

Ejemplo de paquete

```
package Lecture11PkgClass;

public class PkgClass {
    public int publicInt;
    private int privateInt;
    int packageInt;           // Acceso al paquete
    public PkgClass(int pu, int pr, int pa) {
        publicInt= pu;
        privateInt= pr;
        packageInt= pa;
    }
    public void publicPrint() {
        System.out.println("Public");
    }
    private void privatePrint() {
        System.out.println("Private");
    }
    void packagePrint() {           // Acceso al paquete
        System.out.println("Package");
    }
}
```

Clase1 de paquete de prueba

```
package Lecture11PkgClass;           // En el mismo paquete

public class PkgTest {
    public static void main(String[] args) {
        PkgClass object1= new PkgClass(1, 2, 3);
        int pu= object1.publicInt;
        int pr= object1.privateInt;
        int pa= object1.packageInt;
        object1.publicPrint();
        object1.privatePrint();
        object1.packagePrint();
        System.exit(0);
    }
}

// ¿Qué sentencias no compilan?
```

Clase 1 de paquete de prueba

```
package Lecture11PkgClass;           // En el mismo paquete

public class PkgTest {
    public static void main(String[] args) {
        PkgClass object1= new PkgClass(1, 2, 3);
        int pu= object1.publicInt;
        // int pr= object1.privateInt;           ¡Sin acceso!
        int pa= object1.packageInt;
        object1.publicPrint();
        // object1.privatePrint();             ¡Sin acceso!
        object1.packagePrint();
        System.exit(0);
    }
}
```

Clase 2 de paquete de prueba

```
package Lecture11TestPkg; // Diferente paquete (o ninguno)

import Lecture11PkgClass.*; // Importa el paquete deseado

public class PkgTest {
    public static void main(String[] args) {
        PkgClass object1= new PkgClass(1, 2, 3);
        int pu= object1.publicInt;
        int pr= object1.privateInt;
        int pa= object1.packageInt;
        object1.publicPrint();
        object1.privatePrint();
        object1.packagePrint();
        System.exit(0);
    }
}
// ¿Qué sentencias no compilan?
```

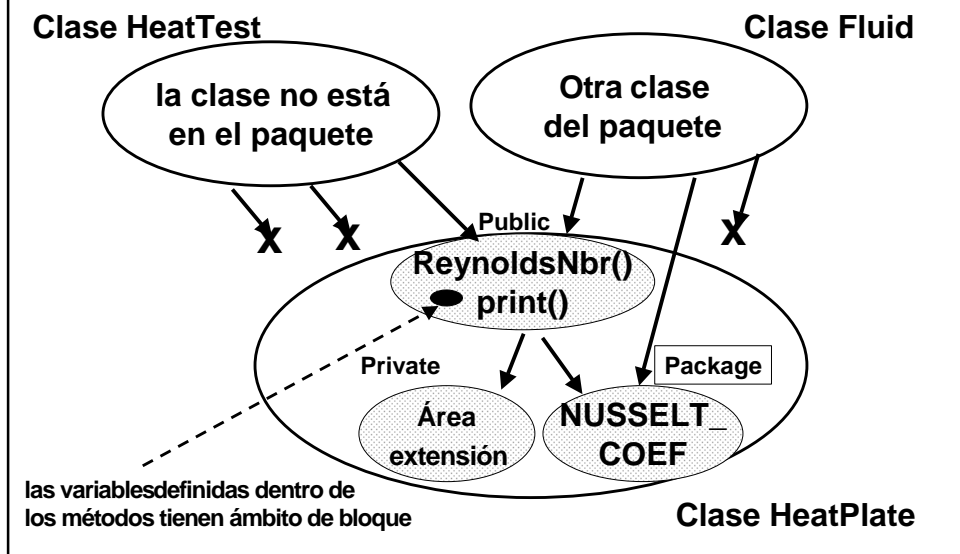
Clase 2 de paquete de prueba

```
package Lecture11TestPkg; // Diferente paquete (o ninguno)

import Lecture11PkgClass.*; // Importa el paquete deseado

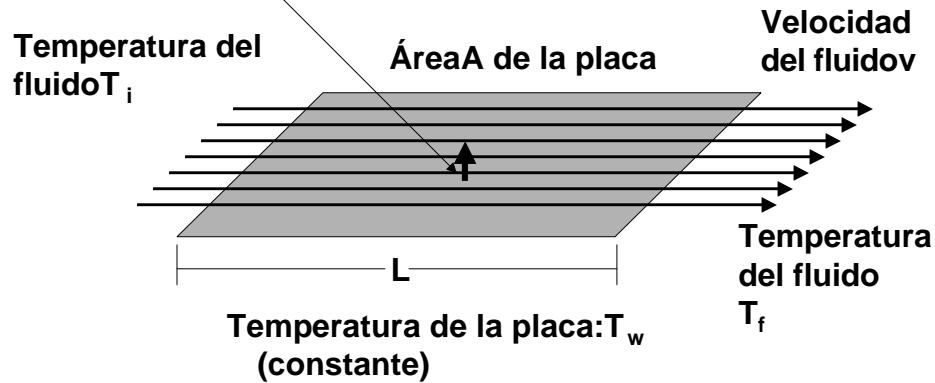
public class PkgTest {
    public static void main(String[] args) {
        PkgClass object1= new PkgClass(1, 2, 3);
        int pu= object1.publicInt;
        // int pr= object1.privateInt; Sin acceso
        // int pa= object1.packageInt; Sin acceso
        object1.publicPrint();
        // object1.privatePrint(); Sin acceso
        // object1.packagePrint(); Sin acceso
        System.exit(0);
    }
}
```

Ejemplo de acceso



Flujo de calor sobre placa

Transferencia de calor de la placa al fluido



Flujo de calor sobre la placa

- Calcule el número de Prandtl :
 - Amplitud de los límites:
 - De la temperatura y velocidad del fluido.
- Calcule el número de Reynolds:
 - Flujo laminar frente a flujo turbulento (viscosidad, velocidad).
 - El análisis estándar supone un flujo laminar.
- Calcule el número de Nusselt :
 - Transferencia de calor desde la superficie; transmitida por el fluido
- Calcule h_{Plate}
 - Tasa (velocidad), área y grados de la transferencia de calor
- Calcule la transferencia de calor:
 - $h_{\text{Plate}} * A * (\text{dif. temp.})$
- Calcule T_f : temperatura de la capa del fluido

Flujo de calor: clase Fluid

```
class Fluid { // Ámbito del paquete
    private double u; // Viscosidad del fluido (kg/m-sec)
    private double p; // Densidad del fluido (kg/m^3)
    private double Cp; // Calor específico del fluido (kJ/kg-K)
    private double k; // Conductividad del fluido (W/m-K)
    private String name; // Nombre del fluido
    public Fluid(double u, double p, double Cp, double k,
        String name) {
        this.u= u;
        this.p= p;
        this.Cp= Cp;
        this.k= k;
        this.name= name;    }
    public double getU() {return u;}
    public double getP() {return p;}
    public double getCp() {return Cp;}
    public double getK() {return k;}
    public String getName() {return name;}
    public void print() { /* Véase el código de ejemplo */ }
    public double PrandtlNumber() {
        return 1000.0*Cp*u/k; } // 1000 is unit convert
```

Flujo de calor: clase HeatPlate

```
class HeatPlate { // Ámbito del paquete
    private double L; // Longitud de la placa (m)
    private double A; // Área de la placa (m^2)
    private double Tw; // Temperatura de la placa/wall (C)
    private double v; // Fluid free stream velocity(m/sec)
    private double Tf; // Fluid free stream temperature (C)
    private String name; // Nombre de la placa
    private Fluid f;
    public HeatPlate(double L, double A, double Tw, double v,
        double Tf, Fluid f, String name) {
        this.L= L;
        this.A= A;
        this.Tw= Tw;
        this.v= v;
        this.Tf= Tf;
        this.f= f;
        this.name= name;    }

    public void print() { /* Véase el código de ejemplo */ }
    public void printResult() { /* Véase el código de ejemplo */ }
```

Flujo de calor: clase HeatPlate 2

```
public String getName() {return name; }
public void setL(double L) { this.L= L; }
public void setA(double A) { this.A= A; }
public double ReynoldsNumber() {          // Dif. que en tuberías
    double retval= f.getP()*v*L/f.getU();
    return retval; }                    // ¡Nunca haga esto!
public double NusseltNumber() {
    double Re= ReynoldsNumber();
    double Pr= f.PrandtlNumber();
    double retval= 0.664*Math.pow(Pr, 1.0/3.0)*Math.sqrt(Re);
    return retval; }
private double hPlate() {                // Observe que el método es privado
    double retval= f.getK()*NusseltNumber()/L;
    return retval; }
public double HeatFlow() {
    double retval= hPlate()*A*(Tw-Tf);
    return retval; }
public double TFilm() {
    double retval= 0.5*(Tw+Tf);
    return retval; } }                // Retval demos scope!
```

```
public class ScopeTest {
    public static void main(String[] args) {
        Fluid air= new Fluid(0.000017, 1.29, 1.005, 0.025,"aire");
        Fluid water= new Fluid(0.001, 1000., 4.2, 0.607,"agua");
        if (air.getName().equals("aire")) {
            HeatPlate platel= new HeatPlate(0.2, 0.04, 70.0, 2.0,
                25.0, air, "placa1");
            for (int i=1; i < 3; i++) {
                platel.setL(0.2*i);
                platel.setA(0.04*i*i);
                double hf= platel.HeatFlow();
                // double hp= platel.hPlate(); // Método privado
                platel.print();
                platel.printResult(); }
            // System.out.println(" Num de placas: " + i); ¡Ámbito!
            // System.out.println(" Flujo de calor: " + hf); ¡Ámbito!
            air.print();
            water.print();
            platel.print(); }
        if (water.getName().equals("agua")) {
            air.print();
            water.print();
            // platel.print(); ¡Ámbito! } }
    }
```

Ejercicio: acceso a paquetes

- Descargue `HeatTransfer.java` de la web del curso 1.00; lo encontrará en la Clase 11
- Coloque las 3 clases (`HeatTransfer`, `Fluid`, `HeatPlate`) en archivos separados
- Coloque `FluidyHeatPlate` en un paquete
 - Defina como públicas las clases `FluidyHeatPlate`
 - Utilice el acceso a paquetes donde sea adecuado (campos, métodos); experimente con esto
- Coloque `HeatTransfer`, con `main()`, en un paquete diferente (o en ninguno, como prefiera)
 - Importe el paquete que contiene las clases `HeatPlate` y `Fluid`
- Compile y pruebe/lea con el depurador

Ejercicio: ámbito y acceso

- En `HeatTransfer`, en `main()`:
 - Imprima el número de placas creadas después de salir del bucle que las creó
 - Imprima el flujo de calor de cada placa después de salir del bucle que las creó
- En `HeatPlate`:
 - Escriba un método privado para comprobar si el flujo es laminar ($Re < 10^6$).
 - Si no es así, devuelva un error/aviso indicando que no se cumplen los supuestos del análisis. Tal vez le parezca complicado; esto es por lo que Java tiene excepciones, que trataremos más adelante
 - Haga que 0,664 sea una constante (`NUSSELT_COEF`) del paquete
- Compile y pruebe/lea con el depurador

Ejercicio:diseño de clases

- **Diseño de nueva clase HeatPlate:**
 - Debata con su compañero las ventajas e inconvenientes de diversos cambios, como:
 - ¿Debería almacenar Re y Nu? Si lo hace, ¿cómo puede estar seguro de que se calculan y se actualizan?
¿Depende esto de la existencia de métodos setXXX() en la clase?
 - ¿Debería almacenar hPlate?
 - El hecho de que métodos llamen a métodos puede resultar confuso.
¿Se debería cambiar?
 - Elementos que pueden ayudar a mantener el control:
 - Evite la devolución de valores sin sentido.
 - Vuelva a nombrarlos argumentos y evite usar 'this'
 - ¿Alguno de estos métodos debería ser estático?
- **Compile y pruebe/lea con el depurador**