Visibilidad de Métodos y Atributos. Encapsulación (get y set)

Programación Orientada a Objetos

Visibilidad

- Utilidad de métodos y atributos
  - Externa: hay métodos y atributos que son útiles para quien utiliza una clase en sus programas.
  - Interna: hay métodos y atributos auxiliares que sólo son útiles para funcionalidades y cálculos internos de la clase.
- Buena práctica: sólo hacer "visibles" al programador aquellos métodos y atributos con utilidad para él.

Modificadores de acceso

- Se usan para especificar qué partes de una clase son "visibles" para otras clases.
- Se especifican escribiendo `public`, `private`, `protected` o "nada" delante de cualquier **método**, **atributo** o **constructor** de una clase, o de la definición de la propia clase.

Modificador de acceso	Acceso permitido desde...
`private`	Métodos de la misma clase
Ninguno	Métodos de la misma clase Métodos de clases del mismo paquete (*package*)
`protected`	Métodos de la misma clase Métodos de clases del mismo paquete (*package*) Métodos de subclases *(ya veremos más adelante qué es esto)*
`public`	Cualquier punto del programa

Ejemplo: ¿Qué accesos están permitidos?

```
package poo.ejemplo;

public class Manzana {
  public float acidez;
  private String tipo;
  protected boolean madura;

  protected Manzana() {
    acidez = 5.5;
    tipo = "Royal Gala";
  }
}
```

./src/poo/ejemplo/Manzana.java

```
package poo.ejemplo;

public class Arbol {

  public Manzana coger() {
    Manzana m = new Manzana();
    m.madura = true;
    m.tipo = "Starking";
    return m;
  }
}
```

./src/poo/ejemplo/Arbol.java

```
package poo.otropaquete;

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

    Manzana m = new Manzana();
    Arbol arbol = new Arbol();
    m = arbol.coger();

    if(m.acidez > 5.5) {
      System.out.println("Ácida!");
    }

    if(!m.madura) {
      System.out.println("🍏!");
    }
  }
}
```

./src/poo/otropaquete/Ejemplo.java

Encapsulación (I)

- Las propiedades de un objeto no siempre están almacenadas como atributos
  - Algunas están calculadas
  - Otras se tienen que ir a buscar a alguna parte (base de datos, disco, Internet...)
- Otras veces, queremos que un atributo se pueda leer desde cualquier punto del programa, pero no queremos que su valor se pueda modificar más que desde la propia clase.

Encapsulación (II)

- Solución: **encapsulación** de datos.
  - Se trata de "esconder" los atributos de una clase a todas las demás clases, haciéndolos sólo accesibles a través de métodos.

```
public class Complejo {
      private float real, img;
      public float getReal() {
         return real;
      }
      public void setReal(float r) {
         real = r;
      }
      public float getImg() {
         return img;
      }
      public void setImg(float i) {
         img = i;
      }
  }
```

Encapsulación (III)

Por convenio, se encapsula de la siguiente manera.

- Sea un atributo privado:

```
private <tipo> <nombreAtributo>;
```

- Obtener un dato (*getter*)

```
public <tipo> get<NombreAtributo>() {
   return <nombreAtributo>;
}
public boolean is<NombreAtributo> >() {
   return <nombreAtributo>;   // solo para boolean
}
```

- Modificar un atributo (*setter*)

```
public void set<NombreAtributo>(<tipo> valor){
  this.<nombreAtributo> = valor;
}
```

Las guías de estilo de Java recomiendan que todos los atributos deben ser encapsulados.

Ejemplo de utilidad de la encapsulación (I)

- Código asociado a la lectura o escritura de un atributo.

```
public class Circulo {
  private double radio;
  public double getRadio() {
    return radio;
  }
  public void setRadio(double radio) {
    this.radio = radio;
  }
  public double getPerimetro() {
    return 2 * 3.1416 * radio;
  }
  public void setPerimetro(double perimetro) {
    this.radio = perimetro / (2 * 3.1416);
  }
}
```

Accedemos a todas las propiedades de un círculo (radio, área, diámetro, perímetro...) de la misma manera. Nos da igual que estén guardadas como atributos o se obtengan mediante cálculos.

Ejemplo de utilidad de la encapsulación (II)

- Atributos sólo de lectura.

```
public class Coche {
   private String marca;
   public Coche(String marca) {
      this.marca = marca;
   }
   public String getMarca() {
      return marca;
   }
}
```

- **Oculta** detalles de implementación: Nos da igual si un dato está guardado como un atributo o se obtiene mediante complicados procesos.

Clase Ventana: https://github.com/mariomac/ventanas/

*Ejercicio: **diseñar** las clases de la siguiente "situación"*

- Sistema de calificaciones "online"
  - Un profesor (identificado por Nombre y DNI) puede **crear** la cualificación de un alumno en una determinada materia (cuyas propiedades son el nombre de la materia, la nota, y el alumno al que pertenecen).
  - El alumno (identificado por Nombre, DNI y una lista de asignaturas que cursa) puede **leer** sus notas.
  - El alumno puede **solicitar una revisión** al profesor, sobre una determinada materia.
  - El profesor **puede modificar** (o no) la nota del alumno.
- Diseñar las clases:
  - Definir atributos y visibilidad
  - Definir constructores (sólo cabeceras) y visibilidad
  - Definir métodos (sólo cabeceras) y visibilidad.
  - Agrupar clases por *packages* (paquetes).

Apéndice: Reglas de estilo Java

- Nombres de clase: **siempre** en mayúscula la primera letra de cada palabra.

```
public class ComplejoEncapsulado {...}
```

- Nombres de variables, atributos, métodos y parámetros: la primera letra de cada palabra en mayúscula, excepto la primera de todas, que siempre es minúscula.

```
float var;
private float atributoCualquiera;
public void setAtributoCualquiera(float valorCualquiera)
```

- *Setters*: siempre empiezan por "set" seguido del nombre del dato que modifican.

```
public void setReal(float valor)
```

- *Getters*: casi todos empiezan por "get" seguido del nombre del dato que leen.

```
public float getReal()
```

- *Getters* de booleanos: empiezan por "is"

```
public boolean isMadura()
```