USO DE PAQUETES

Es recomendable el uso de paquetes, aunque sea una aplicación muy sencilla.

La estructura de un paquete debería ser similar a una dirección web pero al revés, por ejemplo com.empresa.aplicación

Netbeans: Refactor>Rename o hacer otro y arrastrar dentro.

STATIC

Los métodos y propiedades estáticos son "de clase" es decir, no es necesario crear un objeto para usarlos, sino que se hace a través del nombre de la clase.

Además, los métodos estáticos solo pueden acceder a las propiedades estáticas y NUNCA a las no estáticas.

Entonces, como norma general:

- Usaremos static para constantes.
- Usaremos static para propiedades que no sean distintas en cada objeto. Un ejemplo típico es contar el número de instancias de una clase (objetos creados) si queremos limitar su número por ser tipos que consumen muchos recursos o por ser objetos únicos (patrón singleton).
- Usaremos métodos static si no debemos usar las propiedades de la clase, por ejemplo, validar datos para crear un objeto con los datos correctos ¡antes de crearlo! (en la tarea de la cuenta bancaria el validarCCC). Ejemplo típico es la clase Math que incorpora operaciones matemáticas, todos sus métodos son estáticos.
- En la clase que contiene la función main, si queremos tener propiedades u otros métodos que se usen en main deberán ser static. Esto tampoco es mucho problema porque quien crea un objeto de dicha clase es la JVM y solo crea uno. Otra opción es crear un objeto de la clase principal dentro de main ¡Sí, se puede! En la tarea 5 de la cuenta bancaria se podrían poner métodos estáticos para mostrar el menú y validar las entradas.
- Explicación muy clara:
    - https://www.adictosaltrabajo.com/2015/09/17/la-directiva-static-en-java/
    - Es antiguo, pero esto sigue vigente
- Avanzado:
    - En clases anidadas (nested clases)
    - Bloques static se ejecutan una sola vez al cargar la clase, sirve para inicializar variables static.

EXCEPCIONES

La cláusula THROWS en la declaración de un método sirve para decir que ese método puede lanzar esta excepción. Esto se hace para dos cosas:

- Lanzar manualmente ese tipo de excepción cuando ocurre algo excepcional.
- Evitar tratar esta excepción dentro del código de nuestro método cuando incluimos llamadas a otros métodos que la pueden lanzar y que el programa no acabe abruptamente. Retrasamos el tratamiento de la excepción delegándolo en el código que llame a nuestro método.

Por tanto, poner una cláusula THROWS en el método main no tiene mucho sentido, ya que la excepción no puede ser enviada “más afuera”.

Obviamente, si colocamos la cláusula THROWS en un método deberemos poner un try-catch en todo el código que llame a este método.

Se puede lanzar una excepción en el constructor y no se creará el objeto.

BUCLES INFINITOS PARA LECTURA

Hay que tener un poco de cuidado cuando leemos datos de proveer una forma razonable de abortar la cuestión.

Leer hasta que nos pongan los datos correctos puede ser un incordio más que una ventaja. Y dar instrucciones claras, por ejemplo “Introduzca el nombre o pulse escape para salir” y comprobar si se ha pulsado el escape, claro.

Return

- Cuando en un if ponemos return ya no es necesario poner else sino solo las instrucciones ya que si se entra en el if de todos modos no se ejecutarán.
- ¿Se puede hacer una instrucción como esta?

```
return unidades = unidades + uds;
```

Sí se puede porque en java los operadores devuelven un valor, esto incluye los operadores de asignación. Un operador de asignación devuelve el valor asignado. ¡Ojo! no true o false porque no es una comparación, la comparación se hace con ==.

Así que esa instrucción es como si hiciéramos estas dos:

```
unidades=unidades+uds;
return unidades;
```

Debido a esto podemos hacer algo como esto `a=b=c=3;` lo que da como resultado que las tres variables tomen el valor 3 ya que se ejecuta de derecha a izquierda así:

```
c=3;
b=c;
a=b;
```

Public y private y métodos accesores (getter y setter)

En teoría las propiedades deberían ser todas private y, si queremos que se acceda a su valor implementaremos como public los métodos get (para leer el valor) y/o set (para cambiar el valor). Si omitimos el set será una propiedad de solo lectura. Si omitimos el get será una propiedad “oculta”. Normalmente no se omite el get, o por lo menos no solo el get sino también el set haciendo que la propiedad sea de uso interno de la clase.

El hecho de que las propiedades sean private forma parte de una característica de la programación orientada a objetos (POO u OOP en inglés) que se llama **encapsulación**, pero en la práctica, si vamos a crear get y set sin restricciones no merece realmente la pena y se pueden dejar como public.

Sin restricciones quiere decir que el set se encarga de asignar el valor, pero sin comprobaciones adicionales y el get de entregar el valor sin comprobaciones también.

Cuestiones sobre clases

- Los nombres de los métodos y propiedades van en minúsculas. (convención)
- Los nombres de las clases empiezan por mayúscula. (convención)
- No se deben dar valores iniciales al declarar sino en el constructor.
- No es necesario poner el constructor por defecto si no vamos a poner otro constructor. Lo hace la JVM.
- Una clausula extends sirve para hacer herencia (Se ve más adelante) para lanzar una excepción es throws y se pone en el método no en la declaración de la clase.
- Los paquetes se crean para meter clases dentro, no para dejarlos en el default.

Otras cuestiones

- Uso de variables booleanas.
- No solicitar datos por teclado en los métodos en general, salvo casos en los que el método sirva precisamente para eso.
- Organizar los proyectos en una clase que tendrá el método main y, posiblemente algún método para gestionar la interfaz (gráfica o textual) y las demás clases. No poner main en una clase para crear objetos.