**ENCAPSULACIÓN**

* Consiste en ocultar la información que no es pertinente o necesaria para realizar una determinada tarea
* Las clases pueden ser declaradas como públicas (**public**) y como **package** (accesibles sólo para otras clases del package).
* Las variables miembro y los métodos pueden ser **public**, **private**, **protected** y **package**. De esta forma se puede controlar el acceso y evitar un uso inadecuado

**HERENCIA**

* Una clase puede derivar de otra (**extends**), y en ese caso hereda todas sus variables y métodos.
* Una clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados

**POLIMORFISMO**

* Los objetos de distintas clases pertenecientes a una misma jerarquía o que implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo
* La idea básica es que una referencia a un objeto de una determinada clase es capaz de servir de referencia o de nombre a objetos de cualquiera de sus clases derivadas
* El poder utilizar nombres de una súper-clase o de una interface permite tratar de un modo unificado objetos distintos, aunque pertenecientes a distintas sub-clases o bien a clases que implementan dicha interface

**CLASES Y MÉTODOS ABSTRACTOS**

* Una clase abstracta (**abstract**) es una clase de la que no se pueden crear objetos.
* Su utilidad es permitir que otras clases deriven de ella.
* Proporcionándoles un marco o modelo que deben seguir y algunos métodos de utilidad general
* Las clases abstractas se declaran anteponiéndoles la palabra abstract
* Una clase abstract puede tener métodos declarados como abstract, en cuyo caso no se da la definición del método
* Si una clase tiene algún método abstract es obligatorio que la clase sea abstract
* En cualquier sub-clase este método deberá bien ser redefinido o bien volver a declararse como abstract (el método y la sub-clase)
* Una clase abstract puede tener métodos que no son abstract. Aunque no se puedan crear objetos de esta clase, sus sub-clases heredarán el método completamente a punto para ser utilizado
* Ejemplo: public abstract class Geometría { ... }
* diferencia de las clases, las interfaces de Java sí tienen herencia múltiple

**CLASES Y MÉTODOS FINALES**

* las variables declaradas como final no pueden cambiar su valor una vez que han sido inicializadas
* Una clase declarada final no puede tener clases derivadas
* Un método declarado como final no puede ser redefinido por una clase que derive de su propia clase

**CONTROL DE LA SERIALIZACIÓN**

* La palabra clave **transient** permite indicar que un objeto o variable miembro no sea serializado con el resto del objeto. Al recuperarlo, lo que esté marcado como transient será 0, null o false (en esta operación no se llama a ningún constructor) hasta que se le dé un nuevo valor. Podría ser el caso de un password que no se quiere guardar por seguridad
* Las variables y objetos **static** no son serializados. Si se quieren incluir hay que escribir el código que lo haga. Por ejemplo, habrá que programar un método que serialice los objetos estáticos al que se llamará después de serializar el resto de los elementos. También habría que recuperarlos explícitamente después de recuperar el resto de los objetos
* Las clases que implementan **Serializable** pueden definir dos métodos con los que controlar la serialización. No están obligadas a hacerlo porque una clase sin estos métodos obtiene directamente el comportamiento por defecto. Si los define serán los que se utilicen al serializar

**INTERFACE**

* Una interface es un conjunto de declaraciones de funciones.
* Si una clase implementa (implements) una interface, debe definir todas las funciones especificadas por la interface.
* Las interfaces pueden definir también variables finales (constantes). Una clase puede implementar más de una interface, representando una alternativa a la herencia múltiple
* Una interface puede derivar de otra o incluso de varias interfaces, en cuyo caso incorpora las declaraciones de todos los métodos de las interfaces de las que deriva

**STATIC**

* Indica que es un método de clase, es decir, un método que puede ser utilizado aunque no se haya creado ningún objeto de la clase

**VARIABLES FINALES**

* Una variable de un tipo primitivo declarada como **final** no puede cambiar su valor a lo largo de la ejecución del programa. Puede ser considerada como una constante

**PERMISOS DE ACCESO EN JAVA**

* Accesibilidad de los Package:
* Para agrupar clases relacionadas.
* Para evitar conflictos de nombres (se recuerda que el dominio de nombres de Java es la Internet). En caso de conflicto de nombres entre clases importadas, el compilador obliga a cualificar en el código los nombres de dichas clases con el nombre del package.
* Para ayudar en el control de la accesibilidad de clases y miembros
* Accesibilidad de clases o interfaces
* Cualquier clase o interface de un package es accesible para todas las demás clases del package, tanto si es public como si no lo es
* Una clase public es accesible para cualquier otra clase siempre que su package sea accesible
* Recuérdese que las clases e interfaces sólo pueden ser public o package (la opción por defecto cuando no se pone ningún modificador)
* Accesibilidad de las variables y métodos miembros de una clase
* Desde dentro de la propia clase:

1. Todos los miembros de una clase son directamente accesibles (sin cualificar con ningún nombre o cualificando con la referencia this) desde dentro de la propia clase. Los métodos no necesitan que las variables miembro sean pasadas como argumento.
2. Los miembros private de una clase sólo son accesibles para la propia clase.
3. Si el constructor de una clase es private, sólo un método static de la propia clase puede crear objetos.

* Desde una sub-clase:

1. Las sub-clases heredan los miembros private de su super-clase, pero sólo pueden acceder a ellos a través de métodos public, protected o package de la super-clase.

* Desde otras clases del package:

1. Desde una clase de un package se tiene acceso a todos los miembros que no sean private de las demás clases del package

* Desde otras clases fuera del package

1. Los métodos y variables son accesibles si la clase es public y el miembro es public.
2. También son accesibles si la clase que accede es una sub-clase y el miembro es protected.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resumen de permisos de acceso en java** | | | | |
| Visibilidad | Public | Protected | Private | Default |
| Desde la propia clase | Si | Si | Si | Si |
| Desde otra clase en el propio package | Si | Si | No | Si |
| Desde otra clase fuera del package | Si | No | No | No |
| Desde una sub-clase en el propio package | Si | Si | No | Si |
| Desde una sub-clase fuera del propio package | Si | Si | No | No |