**RESUMEN UNIDADES 2.1 Y 2.2**

**2.1. CLASES**

Hasta aquí hemos utilizado un paradigma de programación llamado programación estructurada, que emplea control de estructuras, funciones y datos. Pero ahora usaremos un nuevo paradigma, la programación orientada a objetos.

7.1. Definición de una clase

La POO se inspira en una abstracción del mundo real, en la que los objetos se clasifican en grupos.

Podemos definir estos grupos o clase mediante propiedades y comportamientos que presenta sus miembros.

Class nombreclase {

//Definición de clase

}

7.2. Crear una clase desde NetBeans

La definición de una clase tiene que escribirse en un fichero independiente que tiene que llamarse igual que la clase y tendrá una extensión en .java.

Esto podemos hacerlo desde NetBeans siguiendo unos pasos de configuración.

7.3. Atributos

Los datos que definen una clase se denominan atributos. Tipo atributo

7.3.1. Inicialización

Se puede asignar un valor por defecto a los atributos de una clase, haciendo una propia declaración de la clase.

7.4. Objetos

Los elementos que pertenecen a una clase se denominan objetos. Cada uno tiene su propio valor. Una clase es un formulario donde se solicitan una serie de datos.

7.4.1. Referencias

El comportamiento de los objetos en la memoria del ordenador y sus operaciones elementales, es idénticas al de las tablas, porque ambos utilizan referencias.

7.4.2. Variables referenciadas

Antes de construir un objeto necesitamos declarar una variable cuyo tipo sea clase. La diferencia entre un tipo primitivo y un referencial es lo que almacena.

7.4.3. Operador new

La forma de crear objetos es mediante el operador new. El cual busca un hueco disponible donde construir dicho objeto, y por ultimo darle una referencia. Es importante saber cómo ha ceder a dicho objeto mediante distintas variables.

7.4.4. Referencia null

El valor null es una referencia nula.

7.4.5. Recolector de basura

Existen tres formas de conseguir que un objeto no esté relacionado:

-Crear un objeto vacío.

-Asignarle null.

-Asignar una variable distinta.

7.5. Métodos

También disponemos de comportamientos estos son llamados métodos. La definición de un método es un bloque de instrucciones. Los métodos tienen acceso a variables, parámetros, atributos y clases.

7.5.1. Ámbitos de las variables y atributos

El ámbito de una variable define en qué lugar puede usarse y coincide con el bloque en el que se declara la variable. Cualquier atributo o método dentro de una clase podrá ser utilizado en cualquier lugar de ella.

7.5.2. Ocultación de atributos

Dos variables declaradas en ámbitos adunados no pueden tener el mismo identificador, porque genera un error. Sin embargo existe excesiones cuando una variable local en un método tiene el mismo identificador que un atributo de la clase.

7.5.3. Objeto this

La palabra reservada this permite utilizar un atributo cuando ha sido ocultado por una variable local. Al escribir this. En el ámbito de una clase se interpreta como la propia clase.

7.6. Atributos y métodos estáticos

Un atributo estático, también llamado atributo de clase, es aquel del que no existe una copia. Este se declara con static.

La forma de llamar a un método estático es igual que los atributos estáticos, mediante las palabras claves, de forma directa.

7.7. Constructores

Los atributos a los que no les asignamos un valor, se inicializan por defecto en un valor determinado por su tipo. Un constructor es como cualquier otro método se le puede pasar parámetros y se puede sobrecargar. Cuando en una clase no se implementa un constructor java se encarga de crear uno por defecto.

7.7.1. This ()

Cuando una clase dispone de un conjunto de constructores sobrecargados, es posible construir uno para reutilizar su funcionamiento.

7.8. Paquetes

En java es importante controlar la accesibilidad de una clase desde otra, por razones de seguridad y eficiencia. Esto se hace mediante paquetes, son contenedores que permiten guardar clases en compartimiento separados. Package import public.

7.8.1. Crear un paquete desde NetBeans

En java cada paquete se convierte físicamente en un directorio que contiene clases y otros paquetes. Gracia a NetBeans siguiendo unos pasos podemos crear dichos paquete.

7.9. Modificadores de acceso

Una clase puede ser visible por otro o no, dependiendo de la ubicación y los modificadores.

7.9.1. Modificadores de acceso para clases

Debido a la estructura de clases, organizadas en paquetes, que utiliza java, las clases pueden definirse como vecinas o externas. Existen dos formas de verlas:

Por Defecto

Cuando una de las clases esta definida sin ningún valor.

Total

Mediante el modificador public.

7.9.2. Modificadores de acceso para miembros

De igual manera que es posible modificar la visibilidad de una clase, podemos regularla para los miembros. Para ello es indispensable que ambas clase sean visibles. Esto se puede ver de dos formas:

Por Defecto

Es aquella que se aplica a miembros declarados sin ningún miembro declarados sin ningún modificador de acceso. Esto lo hace visible desde clases vecinas pero invisibles a clases externas.

Modificadores de acceso prívate y public

PRIVATE: permite una visibilidad más restrictiva que por defecto, porque impide el acceso a clase vecinas. Y son invisibles desde fuera de la clase donde están declaradas.

PUBLIC: es todo lo contrario a prívate.

7.9.3. Métodos get/set

Un atributo público puede ser modificado desde dentro de la clase.

Set: permite asignar un valor al atributo, controlando el rango valido de valores.

Get: Devuelve el atributo, conociendo su valor.

La ventaja de usar estos métodos es la implementación de la clase se encapsula.

7.10. Enumerados

Los tipos enumerados sirven para definir grupos de constantes como posibles valores de una variable. Para ello utilizamos la palabra clave: num.

Si vamos a utilizar una clase num., únicamente en una clase esta se puede definir dentro de ella. Los tipos num. Se pueden definir en paquetes distintos a donde se vayan a usar, en ese caso hay que declararlos como public e impórtalos al igual que una clase.

**2.2. HERENCIA**

La herencia es una de las grandes aportaciones de la POO que permite unas características pases de padre a hijos, cuando esto pasa los atributos y métodos visibles, se permiten reutilizar en el código.

8.1. Subclase y superclase

Una subclase dispone de los miembros heredados de la superclase, añadiendo nuevos atributos y métodos. Esto aumenta las posibilidades del funcionamiento del código, evitando repeticiones del mismo y velocidad. Realmente todas las clases de java vienen heredadas de Object.

Class Subclase extends Superclase {

…

}

8.2. Modificador de acceso para herencia

Con la aparición de la herencia podemos plantearnos algunas cuestiones, Todo lo que se hereda es aquello que no es prívate, a los cuales no tenemos acceso, aunque podemos acceder a hechos a través de un método public. Esto se puede solucionar utilizando el método protected, para facilitar la herencia.

8.3. Redefinición de miembros heredados

Cuando una clase hereda de otra, en algunas ocasiones puede ocurrir que se tenga que modificar algún atributo o método. Consiste en declarar un miembro con el mismo nombre pero que oculte el viejo, dándole un nuevo valor o modificándolo.

8.3.1. Súper y súper ()

Del mismo modo que la palabra reservada this se utiliza para indicar la propia clase, para ello disponemos de súper haciendo referencia a una superclase, donde la vallamos a usar.

8.3.2. Selección dinámica de métodos

Cuando definimos una clase como subclase de otra, los objetos son parte de la superclase.

8.4. La clase Object

L clase Object del paquete de java.lang es una clase especial de la que heredan, de forma directa o indirecta, todas las clases de java. Es la superclase por excelencia ya que está en la cúspide de las herencias de clases. Todas las clases API descienden de Object.

8.4.1. Medito toString ()

Este método está pensado para que devuelva una cadena que represente un objeto que lo invoca con toda la información que interesa mostrar. Su implementación en la clase Object consiste en devolver el no nombre cualificado de la clase a la que pertenece el objeto, seguido de un @ junto con la referencia del objeto.

8.4.2. Método equals ()

Compara dos objetos y decide si son iguales, devolviendo un true o un false. Es un prototipo de parámetros de entrada de tipo Object para poder comparar ojetos.Debe mantenerse al hacer overriding en cualquier subclase, porque si no sería una sobrecarga de la clase.

8.4.3. Método getclass ()

Es común usar variables Object para referenciar un objeto de cualquier clase que como sabemos, siempre viene de Object. Para saber de dónde viene esta clase utilizamos getClass, este método lo que hace es invocar un objeto cualquiera y nos devuelve su clase.

8.5. Cases abstractas

En la jerarquía de herencias de clases, cuanto más bajo, más específica y particular es el método. Asimismo cuanto más arriba más general.

Hay métodos que no podemos implementar en una clase determinada por falta de datos pero si en una subclase, para añadir los atributos necesarios. La idea es incrementar vacíos donde solo con el prototipo, la superclase hace un overring para implementar los detalles. Todas las clases tienen un método abstracto que debe ser declarado como abstract.

Las clases abaratas existen para heredar de otra y no para ser instanciadas.