Resumen Temas 7 y 8

Tema 7 – Clases

POO: Programación Orientada a Objetos.

Clase: Es una plantilla que define la forma de un objeto. Especifica los datos y el código que operará en esos datos.

Atributos: Son los datos que definen una clase.

Objeto: Es una abstracción del mundo real en el que se representan un conjunto de atributos y métodos que comparten.

Operador "new": Se utiliza para crear objetos y asignar una referencia a la variable del objeto (p = new Persona();).

Referencia "null": Es una referencia nula, una referencia a ningún bloque de memoria. Cuando declaramos una variable referencia se inicializa por defecto a null.

Recolector de basura: Es un mecanismo de Java que se ejecuta de vez en cuando de forma transparente al usuario y se encarga de comprobar todos los objetos de la memoria. Si alguno no estuviera referenciado por ninguna variable, lo destruye liberando memoria.

Métodos: Son los comportamientos u operaciones que pueden realizar los objetos de una clase. Son funciones que se implementan dentro de una clase.

Ámbito de variables/atributos: Define en qué lugar puede usarse y coincide con el bloque en el que se declara la variable, que puede ser:

- El bloque de una estructura de control: *if*, *if-else*, *switch*, *while*, *do-while* o *for*. Las variables declaradas en este ámbito se denominan **variables de bloque**.
- Una *función* o *método*. Las variables declaradas aquí se conocen como **variables locales**.
- La clase. Cualquier *atributo/método* definido en una clase podrá ser utilizado en cualquier lugar de ella. Los **atributos** son **variables de la clase**.

Un ámbito puede contener otros ámbitos, formando una estructura jerárquica (una clase puede contener don métodos, y estos, distintos bloques de variables de bloque [if-else, while…]).

Ocultación de atributos: Dos variables declaradas en ámbitos anidados no pueden tener el mismo identificador. Existe una excepción cuando una variable local en un método tiene el mismo identificador que un atributo de la clase. Dentro del método, la variable local tiene prioridad sobre el atributo, al utilizar el identificador se accede a la variable local y no al atributo.

Objeto this: Palabra reservada. Permite utilizar un atributo incluso cuando ha sido ocultado por una variable local. Al escribir **this** en el ámbito de una clase se interpreta como la propia clase, y permite acceder a los atributos, aunque se encuentren ocultos.

Atributo estático: También llamado *atributo de la clase*, es aquel del que no existe una copia en cada objeto. Todos los objetos de una misma clase comparten su valor (por ejemplo, un atributo que nos indique que día es hoy, si es lunes, será lunes para todo el mundo).

- Un atributo estático se declara mediante la palabra reservada **static**.

Constructores: Son métodos especial que debe tener el mismo nombre que la clase, se define sin tipo devuelto y se ejecuta inmediatamente después de crear el objeto. Sirven para asignar valores a los atributos, aunque también se pueden utilizar para otros fines como crear tablas, mostrar cualquier tipo de información, etc.

- Los atributos a los que no se les asigna un valor en su declaración se inicializan por defecto dependiendo de su tipo (0 para numéricos, null para referencias y false para booleanos).
- Cada vez que creamos un objeto si no queremos trabajar con los valores por defecto. El operador **new** facilita esta tarea mediante los constructores.
- Se le puede pasar parámetros y se puede sobrecargar.
- La llamada al constructor con los valores de los parámetros de entrada se hace por medio del operador **new**.
- Los atributos declarados como **final** también se pueden inicializar pasando sus valores como parámetros al constructor; no es necesario hacerlo en el sitio donde se declaran.

this(): Es un constructos genérico que sirve para que un constructor pueda llamar a otro y así reutilice su funcionalidad. Tenemos que tener presente que, en el caso de utilizar **this()**, tiene que ser siempre la primera instrucción de un constructor; en otro caso se producirá un error.

```
//Constructor sobrecargado que solo asigna el nombre

Persona (String nombre){

        this(nombre, 0, 1.0); //Invoca el primer constructor

        //la edad se pone a 0 y la estatura a 1.0

}
```

Paquetes: Son contenedores que permiten guardar clases en compartimentos separados, de modo que podamos decidir, por medio de la importación, qué clases son accesibles y cómo se accede a ellas desde una clase que estemos implementando.

- Todas las clases están dentro de algún paquete que, a su vez, pueden estar anidados unos dentro de otros. Se considera que una clase que pertenece a un paquete, que a su vez está dentro de otro, solo pertenece al primero, pero no al segundo.
- Los archivos de fuente Java tendrán una sentencia donde se especifica el paquete al que pertenece, que empieza con la palabra clave **package** seguida del nombre del paquete.
- Una serie opcional de sentencias de importación, con la palabra reservada **import**, que permite importar clases definidas en otros paquetes.
- La definición de una o más clases, de las cuales solo una puede ser declarada pública – por medio del modificador de acceso **public**-.

Modificadores de acceso: Modifican la visibilidad de las clases entre ellas y los paquetes.

Debido a la estructura de clases, organizadas en paquetes

- **Clases vecinas:** Cuando ambas pertenecen al mismo paquete.
- **Clases externas:** Cuando se han definido en paquetes distintos.
- **Visibilidad por defecto:** Cuando definimos una clase sin utilizar ningún modificador de acceso, se dice que usa visibilidad por defecto, que hace que solo sea visible por sus clases vecinas.
- **Visibilidad total:** Mediante el modificador de acceso **public**, hacemos que una clase además de ser visible para sus clases vecinas sea visible también para las clases externas.

Modificadores de acceso para miembros: Igual que podemos regular la visibilidad de una clase, podemos regular la visibilidad de sus miembros. Que un atributo sea visible significa que podemos acceder a él, tanto para leer como para modificarlo.

- **Visibilidad por defecto:** Hace que un miembro sea visible desde las clases vecinas, pero invisible desde clases externas.
- **Modificador de acceso private y public:** Con el modificador **private** obtenemos una visibilidad más restrictiva que por defecto, ya que impide el acceso incluso a las clases vecinas. En cambio, **public**, hace que un miembro sea visible incluso desde clases externas previas a importación. Otorga visibilidad total.

Métodos get/set: Un atributo público puede ser modificado desde cualquier clase, ya que es imposible controlar los valores asignados, que pueden no tener sentido. Por este motivo, se utilizan métodos para modificar los atributos, aunque estén protegidos. Estos se identifican con **set/get** seguido del nombre del atributo.

- Las ventajas de utilizar **set/get** son que la implementación de la clase se encapsula, ocultando los detalles y, por otro lado, permite controlar qué atributos son accesibles para la lectura y cuáles para la escritura, así como los valores asignados.

Enumerados: Los tipos enumerados sirven para definir grupos de constantes como posibles valores de una variable. En la definición usamos la palabra clave **enum** y no **class**. En un programa se accede a sus valores de la forma **nombreClaseEnum.valor**. Un tipo enumerado se puede implementar en un archivo aparte, como si fuera una clase o bien dentro de la definición de la clase donde se va a usar.

- Cuando tengamos que introducir por teclado un valor de tipo enumerado, escribiremos una cadena como “VALOR” y no “nombreClaseEnum.valor” (**Stirng día = sc.nextLine(); //introducimos VALOR**). Para asignarlo a una variable de tipo nombreClaseEnum, tendremos que convertirla en el valor enumerado correspondiente. Para eso se sua el método **valueOf()**, que convierte la cadena “VALOR” en el valor nombreClaseEnum.valor (**DiaDeLaSemana ingles = DiaDeeLaSemana.valueOf(dia)**).

Tema 8 – Herencia

La **herencia** es una de las grandes aportaciones de la POO y permite, igual que en la vida real, que las características pasen de padres a hijos. Cuando una clase hereda de otra, adquiere sus atributos y métodos visibles, permitiendo reutilizar el código y las funcionalidades, que se pueden ampliar o extender.

La clase de la que se hereda se denomina **clase padre o superclase**, y la clase que hereda es conocida como **clase hija o subclase**.

Una **subclase** dispone de los miembros heredados de la superclase y, habitualmente, se amplía añadiéndole nuevos atributos y métodos. La forma de expresar cuál es la superclase de la que heredamos es mediante la palabra reservada **extends**.

Java NO permite herencia múltiple, solo permite herencia simple, donde cada clase tiene como padre una única superclase, cosa que no impide que, a su vez, tenga varias clases hijas.

Modificador de acceso para herencia: Los miembros/atributos de una clase se heredan mientras no tengan el modificador de acceso **private**, no obstante, se puede acceder a ellos indirectamente con un método no privado.

- Junto con los métodos de visibilidad citados, para que un miembro sea accesible desde una subclase, con el fin de obtener una mayor flexibilidad, podemos hacer uso de un nuevo modificador de acceso, **protected**, pensado para facilitar la herencia.
- Funciona de forma muy similar a la visibilidad por defecto, con la diferencia de que los miembros protegidos serán siempre visibles para las clases que hereden, independientemente de si la superclase y la subclase son vecinas o externas, aunque en ese último caso, habrá que importar la superclase.
- Un miembro **protected** es visible en las clases vecinas, no es visible para las clases externas, pero siempre es visible, independientemente del paquete al que pertenezca, desde una clase hija.

Redefinición de miembros heredados: Cuando una clase hereda de otra, en alguna ocasión puede ocurrir que interese modificar el tipo de algún atributo o redefinir un método. Este mecanismo se conoce como **ocultación** para los atributos y **sustitución** para los métodos. Consiste en declarar un miembro con igual nombre que uno heredado, lo que hace que ese se oculte o se sustituya por el nuevo.

Super y super(): Del mismo modo que la palabra reservada **this** se utiliza para indicar la propia clase, disponemos de **super** para hacer referencia a la superclase de aquella donde se usa.

- Una restricción de **super()** es que, si lo utilizamos, tiene que ser forzosamente la primera instrucción que aparezca en la implementación de un constructor.

Selección dinámica de métodos: Cuando definimos una clase como subclase de otra, los objetos de la subclase son también objetos de la superclase. Podemos crear variables de las clases (Persona, Caballo, Fecha…). Solo serán visibles los miembros definidos en la clase creada. Sin embargo, cuando hay ocultación de atributos o sustitución de métodos en la subclase. Los atributos accesibles son los definidos en la clase variable. Si se usa en un método, se ejecuta la versión del objeto referenciado.

La clase Object: Es una clase especial de la que heredan, directa o indirectamente, todas las clases de Java. Es la **superclase** por excelencia, ya que se sitúa en la cúspide de la estructura de herencias entre clases.

- Todas las clases que componen la API descienden de la clase **Object**. Incluso cualquier clase que implementemos nosotros hereda de **Object**. Esta herencia se realiza por defecto, sin necesidad de especificar nada.
- Al heredar de **Object** todas las clases implementan un conjunto de métodos que son de uso universal en Java, como realizar comparaciones entre objetos, clonarlos o representar un objeto como una cadena. La función de esos métodos es ser reimplementados a la medida de cada clase.
- Podemos referenciar cualquier objeto, de cualquier tipo, mediante una variable de tipo **Object**.

Método toString(): Este método está pensado para que devuelva una cadena que represente al objeto que lo invoca con toda la información que interese mostrar. Debe declararse **public**, igual que en la clase **Object**, ya que todo método que sustituye a otro tiene que tener, al menos, el mismo nivel de acceso (**public String toString())**.

Método equals(): Compara dos objetos y decide si son iguales, devolviendo **true** en caso afirmativo y **false** en caso contrario (**public boolean equals(Object otro)**).

Método getClass(): Es común usar una variable **Object** para referenciar un objeto de cualquier clase que, como sabemos, siempre será una subclase de **Object**. A veces necesitamos saber cuál es la clase. Para eso está el método **getClass()**, definico en **Object** y heredado por todas las clases. Invocado por un objeto cualquiera, devuelve su clase que, a su vez, es un objeto de la clase **Class**. Todas las clases de Java, incluidas **Object** y la popia **Class**, son objetos de la clase **Class**.

Clases abstractas: Para declara un método cuya implementación se delega en las subclases (abstracto) se le antepone el modificador **abstract** y se declara el prototipo, sin escribir el cuerpo de la función.

Wrappers: Es la clase utilizada para envolver o encapsular tipos primitivos. Estas clases proporcionan métodos y funcionalidades adicionales para trabajar con primitivos como objetos. Al declararlos se pondrá la primera letra del identificador con mayúscula.

Tipos de relaciones entre clases:

- **Composición:** Un atributo de una clase es de tipo otra clase.

- **Herencia**: Una clase hereda de otra (Superclase) sus métodos y atributos que no son privados.

- **Clientela:** Se puede implementar mediante composición y agregación. Consiste en que la clase cliente depende de la clase proveedora, lo que significa que la clase cliente necesita acceder a los atributos y métodos de la clase proveedora.

- **Anidamiento:** Es posible declarar una clase dentro de otra clase. Esta funcionalidad ofrece al desarrollador otra manera más de organizar su código. Las clases principales se guardan en los paquetes y, por lo tanto, es posible realizar agrupaciones dentro de ellas.

Preguntas de examen

Preguntas finales tipo test (Lo implementa Ja1ro)

1.Sobre una subclase es correcto afirmar que:

a)Tiene menos atributos que su superclase

b)Tiene menos miembros que su superclase

c)Hereda los miembros no privados de su superclase

d)Hereda los miembros de su superclase

Explicación:

Una subclase hereda los miembros no privados (atributos y métodos públicos y protegidos) de su superclase, pero no necesariamente tiene menos atributos o miembros que su superclase. El número de atributos y miembros puede variar en función del diseño de la jerarquía de clases y de las necesidades específicas de la subclase.

2. En relación con las clases abstractas es correcto señalar que:

a) Implementan todos sus métodos

b) No implementan ningún método

c) No tienen atributos

d) Tienen algún método abstracto

Explicación:

Las clases abstractas en la programación orientada a objetos son aquellas que no se pueden instanciar directamente y que suelen contener al menos un método abstracto. Los métodos abstractos son aquellos que no tienen una implementación concreta en la clase abstracta, sino que se definen solo como una firma o prototipo, dejando la implementación real a las subclases que heredan de la clase abstracta.

Las clases abstractas pueden tener atributos y métodos concretos, es decir, métodos que tienen una implementación definida en la clase abstracta. Sin embargo, es común que al menos uno de sus métodos sea abstracto.

3 ¿En qué consiste la sustitución o overriding?:

a) En sustituir un método heredado por otro implementado en la propia clase

b) En sustituir un atributo por otro del mismo nombre

c) En sustituir una clase por un subclase

d) En sustituir un valor de una variable por otro

Explicación:

La respuesta correcta es la opción A: "En sustituir un método heredado por otro implementado en la propia clase". La sustitución o "overriding" es un concepto de programación orientada a objetos que permite a una clase hija proporcionar su propia implementación de un método heredado de su clase padre. Al hacerlo, la clase hija puede personalizar o modificar el comportamiento del método para adaptarse a sus propias necesidades sin cambiar la funcionalidad del método original en la clase padre.

4 Sobre la clase object es cierto indicar

a) Es abstracta

b) Hereda de todas las demás

c) Tiene todos sus métodos abstractos

d) Es superclase de todas las demás

Explicación:

La respuesta correcta es la opción D: "Es la superclase de todas las demás". La clase Object es una clase especial en Java que se considera la raíz de la jerarquía de clases de Java. Todas las clases de Java, directa o indirectamente, heredan de la clase Object. Esta clase proporciona algunos métodos comunes a todas las clases de Java, como toString(), equals() y hashCode().

5 Cuál de las siguientes afirmaciones sobre el método equals() es correcta?

a) Hay que implementarlo, ya que es abstracto

b) Sirve para comparar solo objetos de la clase object

c) Se hereda de object pero debemos implementarlo al definirlo en una clase

d) No hay que implementarlo, ya que se hereda de object

Explicación:

La respuesta correcta es la opción C: "Se hereda de object pero debemos implementarlo al definirlo en una clase". El método equals() se hereda de la clase Object, pero a menudo es necesario que lo implementemos en nuestras propias clases para proporcionar una comparación personalizada entre objetos de la misma clase. La implementación predeterminada de equals() en la clase Object compara simplemente si dos objetos son iguales mediante referencia, es decir, si ambos objetos apuntan a la misma ubicación en la memoria. Por lo tanto, si queremos comparar el contenido o los valores de dos objetos de nuestra propia clase, debemos proporcionar una implementación personalizada de equals() que haga esa comparación

6 Cuál de las siguientes afirmaciones sobre el método toString() es correcta?

a) Sirve para mostrar la información que nos interesa de un objeto

b) Convierte automáticamente un objeto en una cadena

c) Encadena varios objetos

d) Es un método abstracto de object que tenemos que implementar

Explicación:

La respuesta correcta es la opción A: "Sirve para mostrar la información que nos interesa de un objeto". El método toString() en Java se utiliza para devolver una representación de cadena del objeto. En otras palabras, convierte un objeto en una cadena de caracteres. Este método está definido en la clase Object y es heredado por todas las demás clases de Java. La implementación predeterminada de toString() en la clase Object devuelve una cadena que contiene el nombre de la clase del objeto, seguido de su dirección de memoria en formato hexadecimal. Por lo tanto, al redefinir el método toString() en una clase personalizada, podemos proporcionar una representación en cadena más significativa y útil del objeto, que muestre la información relevante para nuestro programa.

7 Cuál de las siguientes afirmaciones sobre el método getClass() es correcta?

a) Convierte objetos en clases

b) Obtiene la clase a la que pertenece un objeto

c) Obtiene la superclase de una clase

d) Obtiene una clase a partir de su nombre

Explicación:

La respuesta correcta es la opción B: "Obtiene la clase a la que pertenece un objeto". El método getClass() es un método de la clase Object que se utiliza para obtener la clase a la que pertenece un objeto. Devuelve un objeto de tipo Class, que proporciona información sobre la clase, como su nombre, los métodos y campos que contiene, y otras propiedades. Por lo tanto, podemos utilizar el método getClass() para obtener información sobre un objeto en tiempo de ejecución, lo que puede ser útil en situaciones en las que necesitamos realizar operaciones dinámicas o reflexivas en los objetos.

8 Una clase puede heredar

a) De una clase

b) De dos clases

c) De todas las clases que queramos

d) Solo de la clase object

Explicación:

La respuesta correcta es la opción A: "De una clase". En Java, una clase puede heredar de una única clase padre directa. Este mecanismo se conoce como herencia de una sola clase y es una característica fundamental de la programación orientada a objetos. La clase padre proporciona los campos y métodos que se pueden heredar por la clase hija, lo que permite a la clase hija reutilizar y extender el código de la clase padre. La herencia múltiple, en la que una clase hija hereda de varias clases padres, no es compatible en Java, aunque se pueden simular algunos de sus efectos utilizando interfaces.

9.La selección dinámica de métodos:

a) Se produce cuando una variable cambia de valor durante la ejecución de un programa.

b) Es el cambio de tipo de una variable en tiempo de ejecución.

c) Es la asignación de un mismo objeto a más de una variable en tiempo de ejecución.

d) Es la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución.

Explicación:

La respuesta correcta es la opción D: Es la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución". La selección dinámica de métodos es un concepto de la programación orientada a objetos que permite la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución. Esto permite que el comportamiento de un programa pueda ser modificado durante su ejecución, sin necesidad de modificar su código fuente.

10.¿Cuál de las siguientes afirmaciones sobre el método super ( ) es correcta?

a) Sirve para llamar al constructor de la superclase.

b) Sirve para invocar un método escrito más arriba en el código.

c) Sirve para llamar a cualquier método de la superclase.

d) Sirve para hacer referencia a un atributo de la superclase.

Explicación:

La respuesta correcta es la opción a)

"Sirve para llamar al constructor de la superclase". El método super() se utiliza en programación orientada a objetos para acceder a los métodos y atributos de la superclase. Cuando se llama al método super(), se puede acceder al constructor de la superclase y se pueden pasar argumentos a ese constructor si es necesario. Esto es útil cuando se quiere extender una clase existente y agregar funcionalidad adicional a los métodos existentes.

1. Explica todos los tipos de relaciones que existen.Ejemplificalo:

- **Composición:** Un atributo de una clase es de tipo otra clase.

- **Herencia**: Una clase hereda de otra (Superclase) sus métodos y atributos que no son privados.

- **Clientela:** Se puede implementar mediante composición y agregación. Consiste en que la clase cliente depende de la clase proveedora, lo que significa que la clase cliente necesita acceder a los atributos y métodos de la clase proveedora.

- **Anidamiento:** Es posible declarar una clase dentro de otra clase. Esta funcionalidad ofrece al desarrollador otra manera más de organizar su código. Las clases principales se guardan en los paquetes y, por lo tanto, es posible realizar agrupaciones dentro de ellas.

2. Definición de “Clase abstracta” (De otra forma) y la instanciación concreta.

Una clase **abstracta** es una clase que no se puede instanciar directamente, lo que significa que no puedes crear objetos de esa clase. En su lugar, se utiliza como una clase base para otras clases que extienden la clase abstracta. Una clase abstracta proporciona una estructura común y define métodos que pueden ser implementados por las clases derivadas.

La **instanciación concreta** se refiere a la creación de objetos de una clase concreta que extiende una clase abstracta. Para ello, debes crear una clase que herede de la clase abstracta y proporcionar una implementación para todos los métodos abstractos definidos en la clase abstracta.

3. Diferencias entre **array** y **ArrayList**.

- **Tamaño:** Los **arrays** tienen un tamaño fijo que se especifica al momento de su creación y no puede ser modificado posteriormente. En cambio, los **ArrayList** tienen un tamaño dinámico que puede crecer o disminuir según sea necesario.

- **Tipo de elementos**: Los **arrays** pueden contener elementos de cualquier tipo, incluyendo tipos primitivos y objetos. Los **ArrayList**, por otro lado, solo pueden contener objetos.

- **Flexibilidad:** Los **arrays** proporcionan acceso directo a sus elementos mediante índices, lo que permite un acceso rápido a los elementos individuales. Los **ArrayList** ofrecen una serie de métodos y operaciones adicionales que facilitan la manipulación de la colección, como agregar, eliminar y buscar elementos, sin necesidad de preocuparse por el manejo del tamaño.

- **Funcionalidad:** Los **ArrayList** proporcionan una serie de métodos útiles, como add(), remove(), get(), size(), entre otros, que facilitan la manipulación de la colección. Los **arrays**, por otro lado, tienen menos métodos incorporados y requieren escribir código adicional para realizar operaciones comunes, como agregar o eliminar elementos.

- **Capacidad de crecimiento:** Los **arrays** tienen un tamaño fijo y, si se necesita aumentar su capacidad, se debe crear un nuevo array y copiar los elementos existentes al nuevo array. Los **ArrayList**, en cambio, pueden crecer automáticamente según se agreguen elementos, sin requerir una reasignación manual.

2. Define que es la instanciación.

La instanciación se refiere al proceso de crear un objeto específico de una clase. Una clase es solo una plantilla o un molde que define las características y comportamientos de un objeto, mientras que una instancia es una ocurrencia real de esa clase en la memoria durante la ejecución del programa.

Para crear una instancia de una clase en Java, se utiliza el operador new, seguido del nombre de la clase y paréntesis vacíos (si la clase no tiene un constructor con argumentos).

Objeto objeto = new Objeto();

3. Define final, abstract e implement.

- **final:** La palabra clave **final** se utiliza para declarar una constante, una clase o un método que no puede ser modificado o extendido.

- **abstract**: La palabra clave **abstract** se utiliza para declarar una clase o un método abstracto. Una clase abstracta no se puede instanciar directamente y se utiliza como una plantilla base para clases derivadas.

- **implement**: La palabra clave **implement** se utiliza para declarar que una clase está implementando una o varias interfaces en Java.

4. Que es la especialización.

La especialización se refiere al proceso de crear una clase derivada o subclase a partir de una clase base o superclase existente. La subclase hereda los atributos y métodos de la superclase y puede agregar nuevos atributos y métodos, o modificar los existentes para adaptarlos a sus necesidades específicas.