**PRACTICA N°2**

**SEMINARIO DE SISTEMAS**

**1.- Las diferencias entre la Programación Estructurada vs la Programación Orientada a Objetos.**

No existe un estándar internacionalmente aceptado cuando se trata de definir los términos. En pocas palabras, la programación orientada a objetos es un estilo que trata los datos como objetos con atributos y métodos que pueden aplicarse a estos objetos y también ser heredados por otros objetos. Java es un gran ejemplo de un lenguaje que emplea este concepto. Pero Java es un lenguaje multi-paradigma y también utiliza algunos conceptos familiares para la Programación Procedimental.

La programación estructurada, por otro lado, es un tipo de programación imperativa, donde las declaraciones se ponen en procedimientos, que se pueden volver a llamar cuando sea necesario. C usa programación procedimental.

La POO se centra en clases y objetos. Al representar variables como objetos, se le puede pasar una función (método). Un objeto que pertenece a una clase en particular se puede tratar de forma independiente. La POO puede estar basada en clases, que, en este caso, los objetos se basan en clases predefinidas. La POO basada en prototipos también existe, por lo que no hay necesidad de clases y solo se utilizan objetos.

La programación procedimental no necesita objetos. Como su nombre lo indica, tiene procedimientos que podrían ser estructuras de datos, rutinas y subrutinas.

**Terminología**

La terminología utilizada en cada paradigma varía, aunque pueden significar lo mismo. En la programación de procedimental, las funciones se denominan “procedimientos”, mientras que en POO; preferirán ser nombrados como “métodos”.

La nomenclatura de las estructuras de datos también difiere. La programación procedimental los etiqueta como “registros” mientras que POO usa “objetos”. La Programación procedimental usa una llamada de procedimiento para llamar a una función, mientras tanto, POO utiliza una llamada de mensaje para solicitar acciones de objetos.

**Herencia**

La característica más distinguible del paradigma de POO es la herencia. La herencia da un impulso a POO, ya que permite una facilidad general a través de la cual el código se puede reutilizar y extender sin cambiar el código existente. Los objetos nuevos son capaces de “heredar” las propiedades de los objetos más antiguos. Una subclase puede, por lo tanto, anular un método definido en una superclase. En los casos en que un objeto hereda características de más de un objeto principal, se genera el concepto de Herencia Múltiple.

La programación procedimental no admite la herencia. La herencia solo se puede aplicar a los objetos. Debido a que la programación procedimental carece de objetos, carece de esta característica, por lo que la distingue de la POO.

**Subtipificación**

En POO, se puede lograr un polimorfismo de subtipo, mediante el cual una función escrita para los elementos de un tipo de datos (supertipo) se puede hacer funcionar en los elementos de otro tipo de datos relacionado (subtipo). El principio de sustituibilidad entra en juego; los objetos de un tipo pueden ser reemplazados por objetos de otro tipo si existe una relación “is-a-subtty-of” entre los tipos. POO es versátil y, como tal, la sustituibilidad se puede implementar sin cambiar otras propiedades.

La programación estructurada no tiene esta habilidad. Como tal, subtipos y supertipos no pueden ser declarados. Tampoco se puede lograr la subtipificación del comportamiento.

**Encapsulación**

A diferencia de su contraparte, la POO es capaz de datos vinculantes, así como los métodos que manejan los datos. Forma una cápsula imaginaria que envuelve los datos y métodos, protegiéndolos así de la interferencia externa. La encapsulación es un tipo de abstracción que la POO hace bastante bien. El código se puede escribir para restringir el uso de datos fuera de la cápsula en la que se emplea.

Este ha sido un breve resumen de las diferencias entre la programación orientada a objetos y la programación estructurada.

**2.- ¿Qué es Java?**

Java se puede referir a un lenguaje de programación de computadoras o a una plataforma de software para computadoras.

Java es un lenguaje de programación orientado a objetos cuya versión 1.0 fue publicada por la compañía Sun Microsystems en 1995. Actualmente verás que Java es propiedad de la compañía Oracle, esto es debido a que ésta compró a Sun Microsystems en 2009.

**3.- Características**

* **Sencillo**

Este lenguaje de programación es muy factible de aprender y su coordinación es simple, pulcra y fácil de concebir. Según Sun, el idioma Java es un lenguaje de clasificación simple debido a que:

La coordinación de Java se fundamenta en C ++ para que los programadores la instruyan más sencillamente luego de C ++.

Este lenguaje de programación ha excluido numerosas características enredadas y poco usadas, por ejemplo, sobresalientes explícitos, exceso de operador, entre otros.

No es preciso eliminar esencias sin una reseña porque existe una recaudación automática de componentes no usados en este lenguaje de programación.

* **Orientado a objetos**

Este lenguaje de programación es una lengua de programación encaminado a objetos. Por lo tanto todo en Java es un objeto. Situada a objetos significa que creamos nuestro software como una composición de desiguales prototipos de objetos que concentra datos y conducta.

La programación situada a objetos, catalogándose como una metodología que facilita el desarrollo y el entretenimiento del software al suministrar ciertas reglas. Por lo tanto estos objetos en este lenguaje son reconocidos como OOP.

Los conceptos básicos de OOP son:

Clase  
Objeto  
Herencia  
Polimorfismo  
Distracción  
Encapsulación  
Plataforma autónoma

Este lenguaje de programación es independiente de la plataforma

Este lenguaje de programación es autónoma de la plataforma debido a que es diferente de otras expresiones como por ejemplo C, C ++, entre otro. Que se reúnen en máquinas determinadas de la plataforma, mientras que este lenguaje de programación es una vez de manuscrito, se confecciona en cualquier sitio. Una plataforma es el medio ambiente de hardware o software en el que se confecciona un programa. Existen dos prototipos de plataformas asentadas en software y establecidas en hardware. Este lenguaje de programación suministra una plataforma asentada en software.

La plataforma de este lenguaje de programación prorroga la mayoría de las nuevas plataformas en el sentido de que es una plataforma asentada en software que se confecciona en la parte óptima de otras plataformas afirmadas en hardware. Por lo tanto poseen dos componentes: El medio de ejecución y API (Interfaz de programación de aplicaciones).

El símbolo Java se logra confeccionar en múltiples plataformas, como en Windows, Mac / OS, Linux, Sun Solaris, entre otros. El colector reúne el código Java y lo cambia en bytecode. Este bytecode es una cifra autónoma de la plataforma que se logra ejecutar en múltiples escenarios, es decir, Run Anywhere (WORA) y Write Once.

* **Asegurado**

Este lenguaje de programación es mejor acreditado por su excelente seguridad. Con este lenguaje de programación, logramos desplegar sistemas independientes de virus. Este lenguaje de programación está predilecto porque: No posee un puntero explícito y los programas Java se confeccionan dentro de un medio ambiente delimitado de máquina virtual.

Cómo se asegura este lenguaje de programación

Classloader: Esta expresión en este lenguaje de programación forma parte del tradicional Java Runtime Environment (JRE) que se usa para cargar géneros de Java en el artefacto virtual Java de manera dinámica. Añade seguridad al apartar el paquete de los géneros del sistema de registro local de las que se interesan de las fuentes de red.

Bytecode Verifier: por medio de este lenguaje se prueba las fracciones de código en busca del código ilegal que consigue quebrantar el derecho de entrada a los objetos.

Administrador de seguridad: Establece a qué recursos logra acceder una variedad, de cómo leer y escribir dentro del disco local.

El lenguaje Java suministra estos importes por defecto. El desarrollador de una atención igualmente consigue suministrar algo de seguridad evidentemente por medio de SSL, JAAS, abreviatura, entre otros.

* **Robusto**

La expresión de robusto sencillamente figura fuerte. Por lo tanto este lenguaje de programación es robusto debido a que:

Usa un servicio de memoria sólida.

Existe una falta de itinerarios que impidan las dificultades de seguridad.

Existe una recaudación automática de basura en Java que se confecciona en el Aparato Virtual Java para poder eliminar los objetos que ya no están siendo usados por una aplicación Java.

Existe una administración de irregularidades y el dispositivo de comprobación de tipos en Java. Todos estos sitios descubren que este lenguaje de programación sea más robusto.

* **Arquitectura neutral**

Este lenguaje de programación es imparcial en arquitectura porque no existe característica dependiente de la ejecución, por ejemplo, el tamaño de los prototipos primitivos es afirmado.

En la clasificación C, el prototipo de registro int invade 2 bytes de memoria para el diseño de 32 bits y 4 bytes de recordación para el diseño de 64 bits. Aunque invade 4 bytes de memoria para diseños de 32 y 64 bits en este lenguaje de programación.

* **Portátil**

Se ha determinado que Java es portátil debido a que le aprueba llevar el bytecode de Java a cualquier escenario. No pretende ninguna ejecución.

* **Alto rendimiento**

Sabemos que este lenguaje de programación es más rápido que otras lenguas de programación desarrolladas habitualmente debido a que el bytecode de Java está mucho más cerca del código nativo. Aún es un poco más pausado que una lengua acopiada como por ejemplo, C ++. Este lenguaje de programación es una lengua efectuada por lo que es más lento que los expresados compilados, como por ejemplo, C, C ++, entre otros.

* **Repartido**

Este lenguaje de programación se comercializa porque suministra a los beneficiarios establecer aplicaciones comerciadas en Java. Como por ejemplo EJB y RMI se usan para establecer aplicaciones distribuidas. Esta propiedad de Java nos aprueba acceder a los registros citando a los procedimientos desde cualquier aparato en Internet.

* **Multi-hilo**

Se determina que un hilo es como un esquema apartado, que se confecciona al mismo tiempo. Logramos escribir programas con este lenguaje de programación que dominen numerosas acciones a la vez precisando diversos hilos. La primordial ventaja de multi-threading es que no invade la memoria para cada hilo. Participa en un área de memoria usual Los hilos son significativos para multimedia, programas, aplicaciones web, entre otros.

* **Dinámica**

Este lenguaje de programación es una lengua dinámica. Aprueba la carga dinámica de géneros. Simboliza que los géneros se cargan a pedido. Igualmente acepta funciones de sus dialectos nativos, es decir, de C y C ++.

Este lenguaje de programación acepta la recopilación dinámica y la dirección automática de la memoria en la recolección de basura.

* **Arquitectura Java neutral**

En este lenguaje de programación, los caracteres se reúnen y, por lo tanto, se crean bytecodes. Estas cifras de bytes se logran confeccionar en cualquier diseño de computadora, lo que hace que el diseño de este lenguaje de programación sea neutral.

* **Alto rendimiento del lenguaje**

Este lenguaje de programación brinda un alto beneficio debido a sus caracteres de bytes, es más apresurado que cualquier expresión interpretada. La arquitectura de Java está delineada para oprimir los gastos usuales durante el período de ejecución. Este lenguaje de programación logra crear aplicaciones tratadas utilizando aplicaciones especiales.

**4.- Tipos de datos (primitivos y objetos)**

Byte Cadenas de caracteres (string)  
short Vectores o arrays  
int Tipos definidos por el usuario (leer y escribir archivos,   
long enviar correos electrónicos,  
float ejecutar otras aplicaciones o crear cadenas de texto  
double más especializadas)  
Boolean  
char

**5.- Clases e interfaces.**

Cuando se escribe código en Java, se están escribiendo clases e interfaces. Dentro de dichas clases, los métodos y las variables, además de algunas otras pocas cosas. El cómo se declaran las clases, interfaces, métodos y variables afecta dramáticamente el comportamiento del código. Por ejemplo, un método marcado como público (public) puede ser accesado desde cualquier parte de la aplicación, no así cuando es privado (private). Analicemos entonces poco a poco la declaración de clases en Java.

Reglas para declarar archivos fuente::

Antes de infiltrarnos en el extenso mundo de las clases, veamos una serie de reglas importantes asociadas con la declaración de clases, sentencias import y package en un código fuente:

+ Sólo puede haber una clase pública por cada archivo fuente.

+ Los comentarios pueden aparecer en el principio o final de cualquier línea de código, son independientes de las reglas que se mencionan.

+ Si existe una clase pública en un archivo, el nombre del archivo debe de coincidir con el nombre de dicha clase.

+ Si una clase es parte de un paquete, la sentencia package debe de aparecer en la primera línea de código, antes de cualquier sentencia import o declaración de clases y métodos.

+ Si existen sentencias import, debe de aparecer después de la declaración del paquete (en caso de haber alguno) y antes de la declaración de la(s) clase(s). Si no hay una declaración de paquete, la sentencia import debe de ser la primera en aparecer en el código. Si no hay sentencias package o import, la declaración de la clase debe de aparecer primero.

+ Las declaraciones de paquete e import aplican para todas las clases nombradas en un archivo, no es posible tener varias clases en un archivo y que cada clase pertenezca a un paquete diferente o utilice diferentes import's.

+ Un archivo puede tener más de una clase NO pública.

+ Un archivo sin clases públicas puede tener cualquier nombre, aunque no coincida con el nombre de ninguna de sus clases.

Declaraciones de clases y modificadores::

La declaración general de una clase se hace de la siguiente manera:

class NombreDeMiClase { }

Además de lo anterior puedes agregar modificadores antes del nombre de la misma. Los tipos de modificadores (no todos aplicables a una clase) son:

1. Modificadores de acceso: public, protected, private.

2. Modificadores que no son de acceso: strictfp, final y abstract.

Los modificadores de acceso nos ayudan a restringir-permitir el acceso a una clase, métodos, variables, etc. que se ha creado. Existen 4 niveles de acceso pero solo 3 modificadores de acceso, el cuarto nivel de acceso que no está representado con ningún nombre de modificador es aquel que se obtiene cuando no estableces ningún modificador de acceso (llamado el acceso por defecto o default). En otras palabras, todas y cada una de las clases y componentes de las mismas en Java tienen control de acceso, independientemente si lo estableces manualmente o no.

Acceso a las clases::

Qué significa acceder a una clase? Cuando decimos que el código de una clase A tiene acceso a otra clase B, significa que A hizo una de tres cosas:

+ Creó una instancia de la clase B.

+ Extendió o heredó a la clase B (A es una subclase de B).

+ Accedió a ciertos métodos dentro de la clase B, dependiendo del control de acceso de dichos métodos y variables.

Acceso significa 'visibilidad'. Para que A pueda usar el código de B tiene que poder 'verlo', para ello son los modificadores de acceso que describiré a continuación:

Acceso por defecto (modificador default): Una clase con control de acceso por defecto significa una de dos cosas:

+ Se omitió la especificación de un modificador de acceso.

+ Se escribió la palabra default como modificador de acceso.

El control de acceso por defecto brinda un control a nivel de paquete, es decir, solo las clases dentro del mismo paquete podrán acceder a los métodos, variables y demás cuestiones definidas dentro de la clase. P. ej.: Si A está en el paquete Uno y B está en el paquete Dos, entonces no hay manera de que B sepa de la existencia de A cuando A tiene control de acceso por defecto. Si se trata de utilizar una A dentro de B, el compilador arrojará un error.

Acceso público (modificador public): Una declaración de una clase pública proporciona el acceso a todas las clases de todos los paquetes, todas las clases en el Universo Java (JU) tienen acceso a una clase pública. No debemos olvidar que, aunque se trate de una clase pública, aún necesitamos importar el paquete en el que se encuentra (en caso de que tratemos de acceder a ella desde un paquete distinto) para que podamos utilizarla. P. Ej.:

Código de la clase MiClaseA:---

package paqueteUno;public class MiClaseA {

//métodos y comportamiento de la clase

}

Código de la clase MiClaseB:---

package paqueteDos;

import paqueteUno.MiClaseA;

public class MiClaseB{

//métodos y comportamiento de la clase

}

Es importante mencionar que en el código de MiClaseB podemos sustituir import paqueteUno.MiClaseA; por import paqueteUno.\*; para poder así importar todas las clases dentro de paqueteUno.

Otros modificadores::

Existen otros modificadores que no controlan el acceso a la clase pero sí el cómo es que esta se comporta al momento de ser extendida o instanciada. Estos modificadores son: final y abstract. Analicemos un poco la función de cada uno de ellos.

Clases finales (modificador final): Cuando el modificador final es utilizado en la declaración de una clase, significa que dicha clase no puede ser heredada, es decir, que no puede haber subclases de dicha clase, y para utilizarla debemos de crear una instancia de la misma. Esto se utiliza normalmente cuando queremos que los métodos y función de nuestra clase se queden tal como están, sin que haya otra clase que herede el comportamiento de la que acabamos de definir ni pueda sobreescribir los métodos que contiene. Muchas clases en Java son finales, un ejemplo es la clase String que no puede ser heredada, imaginemos un poco la situación en caso de que pudieramos sobreescribir los métodos de la importantísima clase String, si lo hacemos de manera irresponsable estaríamos afectando el comportamiento tal vez no solo de una clase o una aplicación, sino de todas aquellas clases que utilicen los métodos definidos por String.

Clases abstractas (modificador abstract): Contrario al modificador final, abstract indica que una clase no puede ser instanciada sino que su única razón de existir es ser heredada. Un ejemplo de la utilidad de una clase abstracta podría ser la clase Carro, sabemos que todos los carros tienen características generales que los describen, sin embargo, existen ciertas cosas que distinguen a uno de otro, como la marca, modelo, número de vehículo, etc. Por ello, podríamos definir una clase abstracta Carro, que define métodos y variables que cualquier carro puede poseer pero que necesita de alguna otra información que distinga a un carro de otro.

En este punto es importante hablar un poco más a fondo de las interfaces, anteriormente he mencionado información general sobre ellas pero ahora que hemos tocado el modificador abstract es posible que las interfaces se vuelvan un poquito más fáciles de entender.

Interfaces::

Una interfaz es una plantilla que define métodos acerca de lo que puede o no hacer una clase, se puede definir meramente como una clase abstracta al 100%, por ejemplo, si definimos la interfaz Animal, podremos deducir que todos los animales pueden comer, respirar, etc. pero puede ser que cada uno lo haga de manera diferente, por ello, en la interfaz definimos el comportamiento que todos los animales pueden tener pero no el cómo hacen las cosas ya que cada animal lo realiza de manera diferente. En código se traduce así:

interface Animal{

void comer();

int respirar();

}

class Perro implements Animal{

public void comer(){

//definimos cómo come el perro

}

public int respirar(){

//definimos cómo respira el perro

}

public String ladrar(){

//definimos un método exclusivo del perro

}}

Como podemos observar en el código anterior, en la interfaz existen un par de métodos que definen lo que cualquier animal puede hacer, pero no especificamos cómo es que lo hace, dentro de la clase Perro que implementa la interfaz Animal es que hemos sobreescrito los métodos que se definieron en la interfaz y le dimos un comportamiento un poco más específico de un perro, así mismo, creamos el método ladrar() en el cual se describe cómo es que un perro ladra, dicho método no fue definido en la interfaz Animal por que no todos los animales ladran.

Es importante mencionar algunos puntos importantes con respecto a las interfaces:

+ Todos los métodos de una interfaz son abstractos y deben de ser sobreescritos por la clase que implemente dicha interfaz, asimismo, todos los métodos de una interfaz son públicos, independientemente si se ha especificado manualmente o no con los modificadores public y abstract.

+ Todas las variables contenidas en una interfaz deben ser públicas, estáticas y finales, no existen variables de instancia dentro de una interfaz.

+ Los métodos de una interfaz NO deben ser estáticos.

+ Debido a que los métodos de una interfaz son abstractos, NO deben ser marcados finales.

+ Una interfaz puede extender una o más interfaces.

+ Una interfaz no puede extender o heredar nada que no sea una interfaz.

+ Una interfaz no puede implementar nada.

+ Una interfaz se define con la palabra interface.

+ Las interfaces pueden usadas polimórficamente.

Declarar constantes dentro de interfaces::

Es posible alojar constantes dentro de las interfaces, al hacerlo, se garantiza que todas las clases que implementen dicha interfaz tendrán acceso a la misma constante. Al momento de declarar una constante, esta implícitamente es pública, estática y final, aunque no se especifíque manualmente con los modificadores public, static y final, debido a lo anterior, es imposible cambiar el valor de una constante declarada dentro de una interfaz, al intentar hacerlo, el compilador arrojará un error. P. ej.:

interface MiInterfaz{

int CONST = 12;

void hacerAlgo();

}

class MiClase implements MiInterfaz{

public void hacerAlgo{

CONST = 14;//al llegar a esta línea aparecerá el error

}}