Primera Unidad

EVIDENCIAS

UNIDAD UNO

1. **LABORATORIO.**
2. **PARTICIPACIÓN FORO.**
3. **TRABAJO AUTONOMO INDIVIDUAL.**
4. **TRABAJO AUTONOMO GRUPAL.**
5. **PRUEBA PRIMER PARCIAL.**
6. **EXAMEN PRIMER PARCIAL.**

Contenido

Laboratorio Parcial Uno:

NOTA: 0/3

OBJETIVO

Conocer lo que es excepciones en java asi como se pueden solventar en el código a través de las clases dadas por el docente y también de la investigación del estudiante para la aplicación de este tema en posibles códigos futuros

INSTRUCCIONES

i) Poner atención a la clase dada por el docente.
ii) Crear el método y sus formas sobrecargadas
iii) Indicar la diferencia entre métodos en el programa principal

ACTIVIDADES

1. **Ubicación de recursos**
   i) Formar grupos de máximo 2 personas por computador
   ii) Instalar el IDE Netbeans o Eclipse

2. **Planteamiento del problema**
   a) Crear el código que lleva los métodos deseados
   b) Ver como funciona el programa con cada método que se llama y lo que muestra al usuario
   c) **Ejecutar el programa.**

3. **Entregable (s)**
   i) Cada grupo deberá entregar captura de pantallas de la ejecución del programa, realizando varias corridas con diferentes capturas.
   ii) Identificar los nombres y apellidos de los estudiantes que conforman el grupo.
   iii) También se deberá entregar un marco conceptual o antecedentes y las fuentes e información que contiene la información.

DOCENTE RESPONSABLE	COORDINADOR DE ÁREA
Ing. Rubén Arroyo. Mgs.	Ing. Silvia Arévalo Msc.

UNIVERSIDAD DE LAS FUERZAS ARMADAS
"ESPE"
CARRERA DE INGENIERÍA ELECTRÓNICA Y AUTOMATIZACIÓN

Nombre: Tomás Vilaña Vivanco
Asignatura: P.O.O
Fecha: 21/12/2021
No. Lista: 35
Período: Pregrado
NRC: 7492

1. TEMA:
Sobrecarga de Métodos

2. OBJETIVOS:

2.1. OBJETIVO GENERAL:
Comprender el concepto de excepciones en java en P.O.O. a través de la guía del docente y la investigación de fuentes confiables por parte del estudiante para poder usar estas herramientas en proyectos futuros

2.2. OBJETIVOS ESPECÍFICOS:
- Identificar las características de las excepciones
- Conocer la diferencia entre try-catch y throws.

3. DESARROLLO:
En P.O.O existen varias "errores lógicos" que muchas veces pueden hacer que el programa principal se detenga y todo su funcionamiento se vea afectado por este error que puede ser un ingreso de datos erróneo o operaciones que no pueden ser resueltas para estos casos existen lo que denominados excepciones estas no permiten que sucedan los casos enlistados anteriormente pero con la distinción que le programa seguirá en funcionamiento a pesar de que este "error" se de mostrando un mensaje u otra alerta que nos ayudara a identificar que hubo una línea o ingreso que no es el correcto pero existen 2 formas de que esta verificación se de y se usan el try catch y throws:

- Throws: Esta sentencia se pone para verificar a todo un bloque sin ser especifico pero esto hará que la calidad del código se reduzca haciendo que el programador de a entender que no es capaz de acertar con la línea del problema
- Try catch : Aquí se pone la sentencia try a las líneas que se desea verificar y luego catch donde vera el tipo de error y mostrara un mensaje en el case de que se de ese error algo parecido a un if else solo que no se cumple una condición sino un error

4. CONCLUSIONES:
- Las características más notorias de un método con sobrecarga se podrían definir con los parámetros ya que estos a pesar de tener el mismo nombramiento requieren

diferentes datos para que su llamada ya sea en el programa principal o en otra clase

- La diferencia entre estos es que la sobreescritura cambia el funcionamiento de método, pero sin cambiar los parámetros por el contrario la sobrecarga hace que el funcionamiento sea parecido pero sus parámetros son distintos

5. RECOMENDACIONES:

- El método de la sobrecarga reduce mucho el tiempo necesario de la verificación del correcto funcionamiento del programa por lo tanto ayudara al programador en el adecuado empleo de su plazo de trabajo
- El limite de sobrecargas que puede tener una función no puede pasar la siguiente formula la cual es: 2^n ya que este cuanta con todas las variables posibles de sobrecarga ya sean que se pongan todos o algunos parámetros

6. BIBLIOGRAFIA:

Cecilio, Á. C. (2021). *Java Sobrecarga de métodos y constructores - Arquitectura Java*. https://www.arquitecturajava.com/java-sobrecarga-de-metodos-y-constructores/

Ramiro, D. la V. (2021). *▷ Guía de métodos de sobrecarga en Java | Pharos*. https://pharos.sh/guia-de-metodos-de-sobrecarga-en-java/

Participación Foro Primer Parcial:

NOTA: 0.7/1

Buenos días ingeniero

El tema de encapsulamiento hace ilusión a la ocultación de objetos a través de las distintas formas de restringir este acceso tales como lo son el: public, private o protected esto ayuda a los programadores que no se modifique el código de manera arbitraria ya que podría afectar el proceso que realiza el el código para su correcto funcionamiento, esto es sumamente util al momento de trabajar con un equipo de personas

Al momento de poner el termino private delante de un atributo este se "ocultara" y la unica forma de tener acceso a su valores ya sea para modificarlos o mostrarlos será a través de los métodos getter y setter los cuales al momento de utilizarse fuera de una clase que no sea donde se sitúan los atributos cumplirán con las funciones antes mencionadas sin necesidad de utilizar directamente el atributo como se hacia al inicio

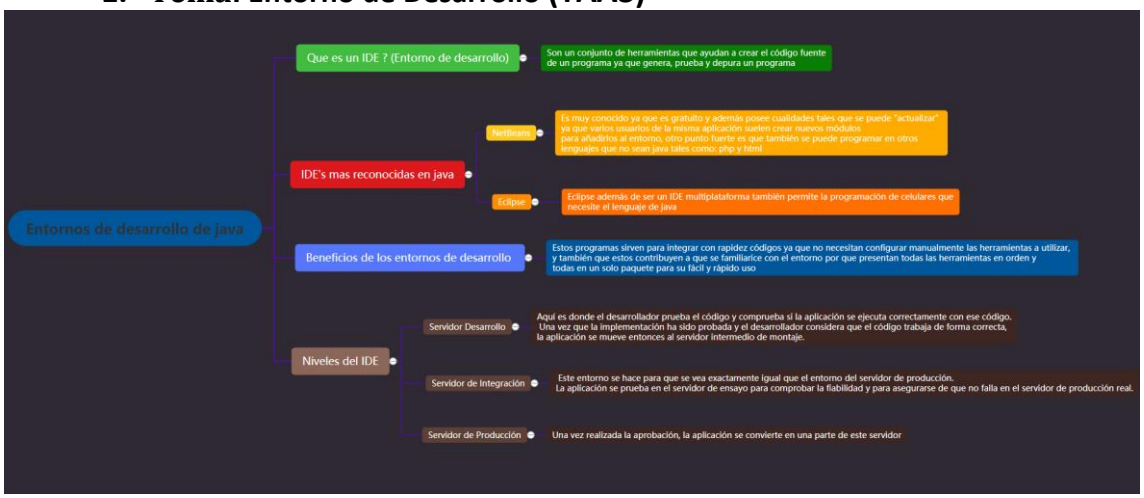
Conclusión:

Esto ayuda a mantener las partes de un programa tal y como se encuentra y solo se tendrá acceso a estas características pro medio de métodos que específicamente sean para ese rol ya que si cualquiera editara los atributos podría causar un fallo en el programa inutilizando todo el código

Trabajo Autónomo Individual Primer Parcial:

NOTA: 3.72/4

1. **Tema: Entorno de Desarrollo (TAA3)**



https://gitmind.com/app/doc/5e45558157

Apellidos Nombres.
Estudiante NRC1234

1. Tema: Revisión de conceptos generales de la POO. (TAA4)

2. Objetivos:

2.1. Objetivo General:

- Conocer las características de los conceptos básicos de la programación orientada a objetos, así como sus definiciones para la construcción de códigos en un IDE utilizado el lenguaje java

2.2. Objetivos Específicos:

- Conocer los elementos de una clase y como se relacionan con el concepto de objeto en programación
- Reconocer la diferencia entre dato primitivo y objeto

3. Desarrollo:

En la programación orientada a objetos existen términos relativamente nuevos que ayudaran a referirse a una determinada parte del código, pero estos conceptos ya son conocidos sim embargo en este paradigma cambian su nombramiento, pero cumplen funciones similares y también se añaden nuevos términos.

- **Objeto:** Un objeto puede ser un dispositivo, un vehículo u otras cosas, pero este es algo único ya que tiene características (Atributos) únicos que los distinguen de los demás de su clase
- **Clase:** Es la abstracción de un objeto, si bien el objeto es algo especifico la clase se refiere a todos los elementos que tengan la misma característica, pero dentro de ellas pueden surgir subclases. Las clases en programación suelen constar de dos partes o elementos que se conocen como atributos y elementos
- **Atributos:** En P.O.O. las variables que se conocían pasan a llamarse atributos estos pueden ser de distintos tipos que son, byte, short, boolean, char, int, long, float, double y String los primeros datos se encuentran en minúsculas ya que si son tipos de datos primitivos estos no cuentan como objetos
- **Métodos:** Al igual que los atributos estos se conocían antes como funciones, pero ahora pasaron a llamarse métodos y estos indican las acciones que realiza un objeto, estos pueden tener argumentos y estos argumentos pueden ser de cualquier tipo de datos, pero estos métodos únicamente recibirán datos de ese tipo o marcara como error

Estas definiciones ayudaran al programador a orientarse sim embargo existen diferencias que no son tan explicitas teóricamente y una de esas es la diferencia entre dato primitivo y objeto. Un dato primitivo puede ser de muchos tipos, pero cuando nos referimos a el hablamos de un solo atributo y no tiene funciones en él, pero la hablar de un objeto debemos tomar en cuenta que consta de una clase la cual se compone de atributos y métodos y por lo general estos elementos son escritos con la primera letra mayúscula indicando que sin un objeto

4. Conclusiones:

- Las clases en java constan de 2 partes los atributos y métodos y cuando un dato se le asigna esa clase en el interior de su memoria se guardará todos los atributos que sean de dicha clase dando como resultado un objeto
- Si bien java tiene una variedad de datos primitivos estos son diferentes a los objetos por que los datos primitivos en su memoria solo guardan el valor asignado

por el usuario o el programador y como se menciono antes un objeto trae consigo mas cosas detallas

5. Recomendaciones:

- Cuando se nombra a una clase es importante utilizar el método camel el cual dice que se debe escribir todo unido, pera marcar el final de una palabra y el inicio de otra con la primera letra en mayúscula (ejercicioVehiculo)
- Cuando se declara un atributo se lo hace con el tipo de dato (int, char, short) y su valor, pero cuando se declara un objeto se lo hace poniendo el nombre de la clase que se requiera y después el nombre del elemento seguido de un igual ponemos la palabra *new* seguido nuevamente de la clase y poniendo entre paréntesis los argumentos de ese método

6. Bibliografía:

Canelo, M. M. (02 de 11 de 2020). *Profile*. Obtenido de profile.es: https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

Tema: UML: Diagramas de Casos de Uso y Diagramas de Clase (TAA5)

2. Objetivos:

2.1. Objetivo General:

- Reconocer el uso de un diagrama de clases y el uso de los casos de usos a través del material de lectura dejado por el docente e información investigada en la web para poder aplicarlo en situaciones que ameriten hacerlo

2.2. Objetivos Específicos:

- Diferenciar cualidades de cada técnica ya sea el uso de casos o el diagrama de clases para aplicarlas correctamente
- Conocer los componentes de cada técnica y su correcto proceso de realización

3. Desarrollo:

Casos de uso:

Esto se emplea generalmente cuando se necita tener una idea no muy detallada de lo que hará el programa y con qué fin lo hará, entre especialistas y cliente se arma este escrito ya que en esta técnica no se usa lenguaje muy complejo o técnico por que ambas partes que interviene, las dos personas deben tener una idea del funcionamiento del programa y como interactúa con cada una sim embargo la explicación del programa no será tan detallada no se verán clases en profundidad ni mucho menos como estas trabajan lo que se enfoca aquí es saber si el código cumple su función ya sea con el usuario y también si es capaz de interactuar con otros programas pero siempre se tiene que tomar en cuenta que el objetivo principal se cumpla

Diagrama de Clases:

Los diagramas de clases suelen tener varios elementos que lo componen, pero la parte mas representativa de estos como su nombre lo dice son las clases ya que esta herramienta ayuda a que se observen todas las clases y sus componentes, también aquí se añade como estas mismas clases se relacionan con las demás dando origen al programa y viendo como es su funcionamiento detalladamente, estos esquemas juegan un papel importante porque se

realizan antes del programa para previsualizar como los objetos interactuaran con su entorno ayudando a darse una idea general de como debe ser el programa y sus funciones

4. Conclusiones:

- Ambos trabajan con conceptos conocidos en java pero se diferencian en sus componentes en el detalle que le dan a cada uno para que el esquema respectivo tenga diferentes funciones
- El diagrama de clases se realiza antes de la escritura del programa para poder dar un panorama general de cómo será el programa y los métodos que se ocuparan por otro lado el caso de usos se lo realiza después del código comprobando que el programa sea efectivo con su objetivo y amigable con el usuario

5. Recomendaciones:

- Es aconsejable que se use un nivel de abstracción un poco alto para el caso de uso ya que en este mapeado no es necesario que el usuario conozca a detalle cada proceso interno que realiza la máquina, pero en los diagramas de clase la abstracción no debe de ser tan elevada para que las clases sean las específicas y se comprenda mejor el fin del programa
- Para el caso de uso es necesario que se haga con 2 grupos de personas aquellos que conocen el programa y el funcionamiento acompañados de las personas que desconocen o serán las personas que ocuparan o harán uso de este programa

6. Bibliografía

S.N. (24 de 07 de 2020). *Digital Guide Ionos*. Obtenido de iones.es: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/

1. Tema: Estándares de implementación, buenas prácticas de programación. (TAA6)

2. Objetivos:

2.1. Objetivo General:

- Aprender buenas normas de escritura para los programas a desarrollar a través del material proporcionado por el docente y el material investigado por el estudiante para los programas sean legibles y de fácil comprensión para cualquier persona que desee leerlo

2.2. Objetivos Específicos:

- Reconocer buenos hábitos que ayudan al programador a que su programa sea legible
- Evitar cometer errores logísticos al momento de la escritura dificultando mantenimientos futuros

3. Desarrollo:

En programación no existen normas inamovibles que dicten como escribir un programa pero si algunas reglas que pueden ayudar a un programador que la calidad de su software aumente y no directamente están relacionados a la realización del programa o su complejidad sino también a la manera de presentación de este ya que al presentar un código limpio y ordenado la lectura del mismo será mucho más fácil y estas acciones no ayudaran a una solo persona sino también a las siguientes que tengan que leer o dar mantenimiento a un código ajeno.

Estas prácticas son relacionadas en su mayoría con la sintaxis del código:

- Cada estructura deberá tener definido una “tabulación” para que sea notable cuando se dé anidamientos o se quiera escribir más bloques de códigos dentro de un método esta regla se da para que no todo el texto este pegado al lado izquierdo de la pantalla, sino que se pueda observar que indicaciones se dan dentro de una sola función y no en el programa principal
- Otra norma es para los nombres sim bien se puede utilizar el método camelCase el cual indica que toda la palabra sea en minúscula pero que cuando se inicie otra en vez de poner un espacio en blanco se escriba junto, pero con la primera letra en mayúscula diferenciado las palabras otra forma de escribir estos nombres es el snake_case el cual nos indica que se puede escribir estos nombres de la forma común pero en reemplazando el espacio en blanco con un guion bajo indicando que son palabras diferentes
- Otra guía que se nos da es sobre los comentarios ya sean de una o varias líneas estos siempre deben comenzar con una línea en blanco y luego debajo de esa línea poner todo el comentario si es de más párrafos la separación entre estos párrafos debe ser una línea en blanco y al finalizar puede poner una línea en blanco de igual manera solo que esta es opcional

4. Conclusiones:

- Los hábitos que ven los comentarios, la jerarquía de las estructuras y el nombre de los elementos ayudan a mejorar la legibilidad de los programas para futuros programadores
- El error humano siempre estará presente esto ayudará a que si en un momento se comete un error sea más fácil encontrarlo y solucionar el problema de manera eficaz

5. Recomendaciones:

- Estas normas si bien no son obligatorias para todo el mundo estas ayudan a crear un sistema propio sin alejarse demasiado de estos para poder seguir con la legibilidad
- Una manera de evitar errores es nombrar a las variables con todo su nombre y no abreviaciones por que el fin de estos es que el programa sea auto explicativo sin la necesidad de la intervención de nadie

6. Bibliografía:

Ant. (13 de 04 de 2013). *decodigo.com*. Obtenido de decodigo.com: https://decodigo.com/2013/04/mejores-practicas.html

Rrabajo Autónomo Grupal:

NOTA: 1.9/2

1. Objetivos:

1.1. Objetivo General:

Conocer lo que es un VCS, para tener un gran conocimiento sobre el tema mediante una amplia investigación.

1.2. Objetivos Específicos:

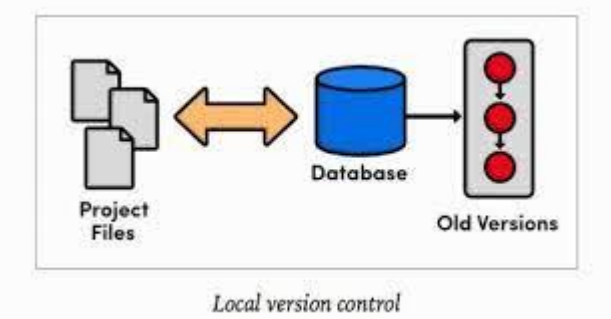
- Conocer el concepto de VCS, para utilizarlo en un ejemplo.
- Mostrar un ejemplo en el cual se aplique al Git
- Sintetizar la información pertinente que se investiga.

2. Tema: Sistema de control de versionamiento (VCS)

El sistema de control de versiones es un sistema que permite registrar los cambios que se realizan sobre un archivo o sobre un conjunto de archivos a lo largo del tiempo, lo cual permite al usuario recuperar versiones específicas en cualquier momento. Además de ser utilizado para guardar cualquier documento que está en constante cambio, como por ejemplo un código fuente, ficheros de configuración entre otros.

- **Sistemas de control de versiones locales**

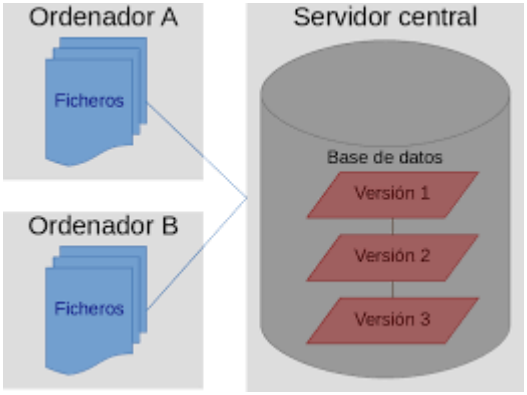
Este es uno de los métodos de control de versiones que suele ser usado con frecuencia por varios usuarios, ya que su método consiste en copiar los archivos a otro directorio, el mismo puede indicar la fecha y hora en que se realizó la copia, a pesar de ser una forma sencilla de manejar, este tiene una mayor probabilidad de contener errores, esto debido a que muchas veces accidentalmente se puede guardar un archivo en un directorio equivocado e incluso sobrescribir archivos no deseados, esta fue la principal razón para la creación del VCSs locales, que se basan en una base de datos simples el cual lleva el registro de todos los cambios realizados sobre los archivos.



Local version control

- **Sistemas de control de versiones centralizados**

Este método nace a partir del problema que tienen los usuarios para colaborar con desarrolladores en otros sistemas, por la cual este sistema de control de versiones centralizado tiene un servidor único el cual contiene todos los archivos versionados y tiene la facilidad de descarga de los archivos almacenados desde la central, por ello este es considerado un estándar para el control de versiones.



- **Sistemas de control de versiones distribuidos**

En un sistema de control distribuido DVCS como Git, Bazzar o Darcs, los usuarios tienen la posibilidad no solo de descargar la última versión de los archivos, también replican completamente el repositorio o llamado almacén, de esta manera si un servidor muere y estos sistemas estaban en colaboración con el servidor, cualquier de los repositorios de los usuarios pueden copiarse en el servidor para ser restaurados, por lo cual cada vez que se descarga una instantánea, esta hace una copia de seguridad total de todos los datos.

2.1. Preguntas

¿A partir de que problema nace el sistema de control de versiones centralizados?

A) Nace a partir del problema que tienen los usuarios para colaborar con desarrolladores en otros sistemas
B) Nace a través del inconveniente que tiene el programador al desarrollar otros sistemas
C) Se desarrolla el problema por que los usuarios no pueden descargarse la ultima versión de los archivos.

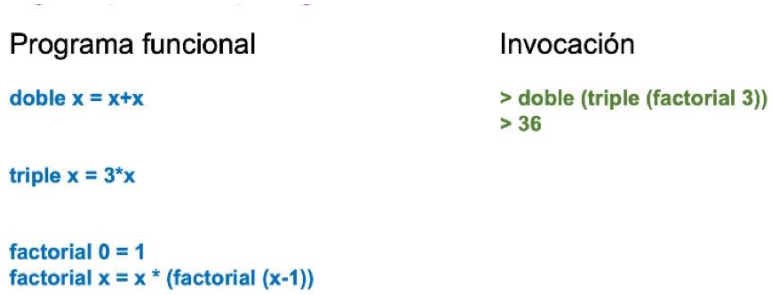
Respuesta: A

3. Tema: Paradigmas de programación

Un paradigma se caracteriza por definir un conjunto de reglas, patrones y estilos de programación, los cuales son utilizados por un grupo de lenguajes de programación y por ello se dividen en diferentes tipos:

- **Paradigma Funcional**

En este paradigma la computación se ejecuta mediante la evaluación de expresiones, se encarga de definir las funciones, funciones como datos primitivos, se encuentra presente la programación declarativa, los valores sin efectos laterales, es decir no existe la asignación. Lenguajes: LISP, Scheme, Hakall.



- **Paradigma Lógico**

Se encarga de definir reglas, unificar como elemento de computación, es una programación declarativa y se encuentra en los lenguajes Prolog, Mercury, Oz.

```
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'debora').

hijode(A,B) :- padrede(B,A).
abuelode(A,B) :-  padrede(A,C), padrede(C,B).
hermanode(A,B) :- padrede(C,A) , padrede(C,B), A \== B.

familiarde(A,B) :- padrede(A,B).
familiarde(A,B) :- hijode(A,B).
familiarde(A,B) :- hermanode(A,B).

?- hermanode('juan', 'marcela').
yes
?- hermanode('carlos', 'juan').
no
?- abuelode('pablo', 'maria').
yes
?- abuelode('maria', 'pablo').
no
```

- **Paradigma imperativo**

Se caracteriza por definir los procedimientos y los tipos de datos, además realiza una revisión de los tipos en tiempo de compilación, también puede hacer el cambio del estado de las variables y mostrar el paso de ejecución de un proceso.

```c
#include <stdio.h>

int main()
{
   int a, b;

   printf( "Introduzca primer numero (entero): " );
   scanf( "%d", &a );
   printf( "Introduzca segundo numero (entero): " );
   scanf( "%d", &b );

   if ( a + b > 0 )
      printf( "LA SUMA SI ES MAYOR QUE CERO." );
   else
      printf( "LA SUMA NO ES MAYOR QUE CERO." );

   return 0;
}
```

- **Paradigma orientado a objetos**

Se encarga de definir las clases y herencias, presenta objetos como abstracción de datos y procedimientos y por último el polimorfismo y chequeo de tipos en tiempo de ejecución.

```java
public class vehiculo {
    String patente;
    int velocidad;
    public void VerDatos()
    {
        System.out.println("Patente: "+patente);
        System.out.println("Velocidad: "+velocidad);
    }
}
```

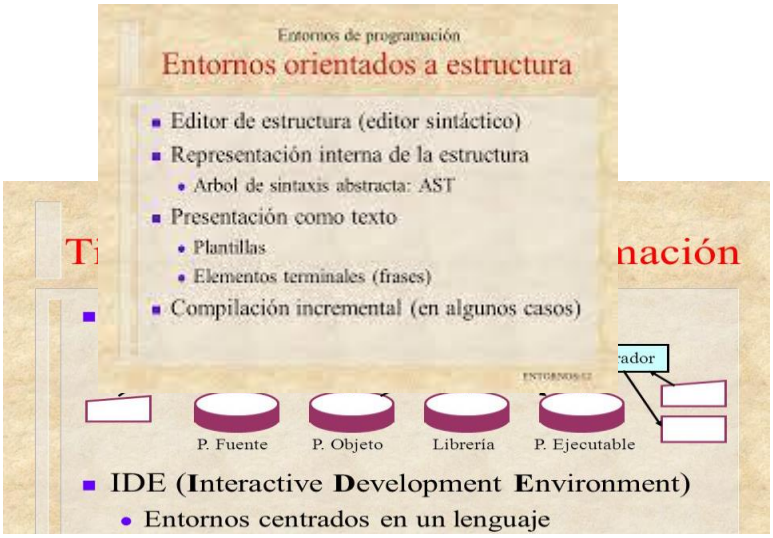
4. Tema: Entornos de desarrollo

Un entorno en programación es aquel que puede estar concebido y organizado de maneras muy diferentes.

Algunas de ellas como las primeras etapas de la preparación de programas se realiza mediante una cadena de operaciones para un lenguaje procesado a través del compilador, cada herramienta debe ser invocada de forma manual y por separado, bajo estas condiciones no se puede hablar de un entorno de desarrollo en programación, sin embrago, un entono de programación propiamente dicho es aquel que combina las diferentes herramientas como el editor, copilador, montador y depurador, pero mejoradas y mejor integradas para mejorar su integración con el usuario.

En cuanto a su clasificación, esta no se puede establecer de manera sencilla tomando en cuenta la variedad de entornos de programación que existen. Para ello se describieron las siguientes clases de entornos basados en un criterio pragmático:

- **Entornos centrados en un lenguaje:**



Son específicos para un lenguaje de programación en particular, son fuertemente integrados, son sencillos de manejar, sin embargo, son poco flexibles para la ampliación de sus funciones y solo se basan en interpretar y representar el código fuente como texto.

- **Entornos orientados a estructuras:**

Suelen ser específicos para un lenguaje de programación, mantienen relación con los entornos centrados en un lenguaje pues estos comparten sus características, pero mantienen una diferencia clara pues estos soportan un único lenguaje de programación.

- **Entornos colección de herramientas:**

Llamados toolkit environments consiste en una combinación de diversas herramientas con la capacidad de interoperar entre ellas de alguna manera y estas requieren de elementos adicionales para integrarlas en un solo conjunto que tenga coherencia entre sí, además están son más flexibles y se pueden adaptar a las necesidades de los usuarios mediante la creación de nuevas herramientas.

5. Tema: Revisión de concepto generales de la POO

Los programas se moldean al ámbito a objetos, el cual consiste en el conjunto de todas las funcionalidades que se relacionan con ellos, puesto que en POO se crean clases, la cuales representan entidades que se quiere manejar en el programa. Por ello el primer concepto que es el más importante de la POO es la diferencia entre clase y objeto.

La clase es simplemente una abstracción que sola no se la puede ejecutar, para poder utilizar una clase en un programa se tiene que crear un objeto concreto de la misma. Para poder reducir la complejidad se debe tener en cuenta algunos principios que consiste en los cuatro pilares de la POO.

- **Encapsulamiento:**

Consiste en que todo lo referente a un objeto que de aislado dentro de éste y solo se puede acceder a ellos por medio de los miembros que la clase proporciones como propiedades y métodos. Dentro de cada clase se puede definir otro encapsulamiento sobre los datos, se clasifican en public, protected y private, los cuales limitan el acceso a las variables y métodos.

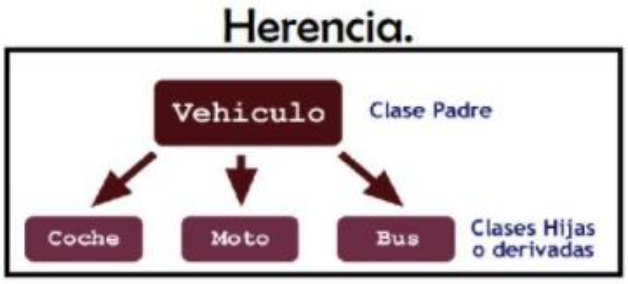
```
public class MiClase {

    public int tipo;
}

class AccesoDirecto {

    public static void main(String[] args) {

        MiClase mc = new MiClase();
        mc.tipo = -5; //1

    }

}
```

- **Abstracción:**

Es una clase en la cual se debe presentar las características de la entidad hacia el exterior, pero se debe ocultar la complejidad que tienen, es decir es prescinde de la complejidad que hay al interior, mosteando una serie de atributos y comportamientos que se puede utilizar sin ninguna dificultad y preocupación de lo que sucede al interior. Tiene mucha relación con el encapsulamiento, puesto que no solo controla el acceso de la información, sino que oculta la complejidad de los procesos que se implementan.

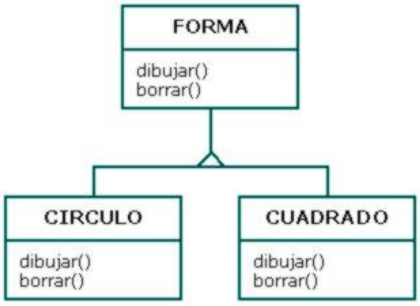
- **Herencia:**

Es una de las cualidades más importantes del sistema de POO, la cual dará mayor potencia y productividad, puesto que ayuda en la depuración de errores y ahorra horas de programación, puesto que unas clases pueden derivar de otras y así se puede aprovechar su forma de trabajo sin tener la necesidad de reescribir el código. Cuando una clase hereda de otra obtiene todos los rasgos que tiene la primera, sin embargo, se puede añadir otros nuevo e incluso modificar algunos de los que se ha heredado.



- **Polimorfismo:**

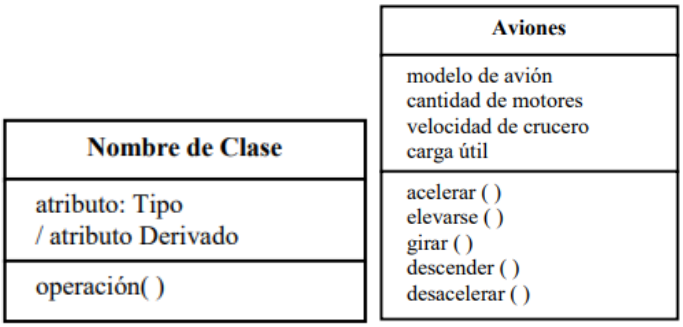
El significado de la palabra es cualidad de tener muchas formas, esto quiere decir que varios objetos de diferentes clases, pero con una base en común, se pueden usar de manera indistinta, sin tener la necesidad de saber de qué clase exacta son para poder hacerlo. Sin embargo, puede ser más complejo, dado a que se puede dar por la sobrecarga de método y sobre todo por medio del uso de interfaces. Permite utilizar a los objetos de manera genérica, aunque por dentro que comportan según su variedad específica.



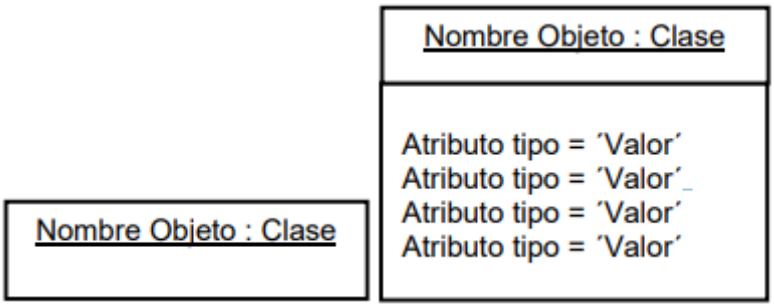
6. Tema: Modelamiento de clases y objetos

El diagrama de clase consiste en describir la estructura estática de un sistema, las cosas que nos rodean y existen se unen naturalmente en categorías. A una clase se le considera una categoría p grupo de cosas que tienen atributos, es decir propiedades y acciones similares.

El símbolo para representar una clase es un rectángulo, el cual se divide en tres áreas. Por ello el diagrama de clase está conformado por algunos rectángulos que se conectan por líneas, las cuales representan las asociaciones o la forma en que las clases se relacionan entre sí.



El diagrama de objetos se vincula con los diagramas de clases, puesto que un objeto es una instancia de una clase. Estos tipos de diagramas describen la estructura estática de un sistema es un momento particular y son usados para probar la precisión de los diagramas de clases. Para ello se coloca un nombre a los objetos, el cual es representado por un rectángulo que tiene el nombre objeto y su clase subrayada. Los atributos se los coloca en la parte inferior en lista, estos atributos deben tener un valor asignado.



7. Código Limpio

Se lo puede definir como una serie de buenas prácticas con las que el programador puede dejar un código legible, elegante y eficaz, entro otros halagos, para que en un futuro el programador le llegue a dar mantenimiento, desde el nombre de una variable, hasta concretas pruebas unitarias, son lo que se debe tener en cuenta para tener un código limpio, puesto que no se debe programar para sí mismo sino para otra persona que puede comprender el código empleado, además existen una reglas o buenas prácticas dentro del código limpio.

- **Nombres con sentido**

Se refiere a redactar variables, objetos, propiedades que tengan sentido, lo cual permita aclarar cuál es la función de dicho objeto de una manera simple. Los nombres de las clases u objetos deben tener nombres o frases, no debe ser un verbo por ejemplo cliente, cuenta, en cambio en el nombre de métodos deben tener nombres de verbo como EnviarPago.

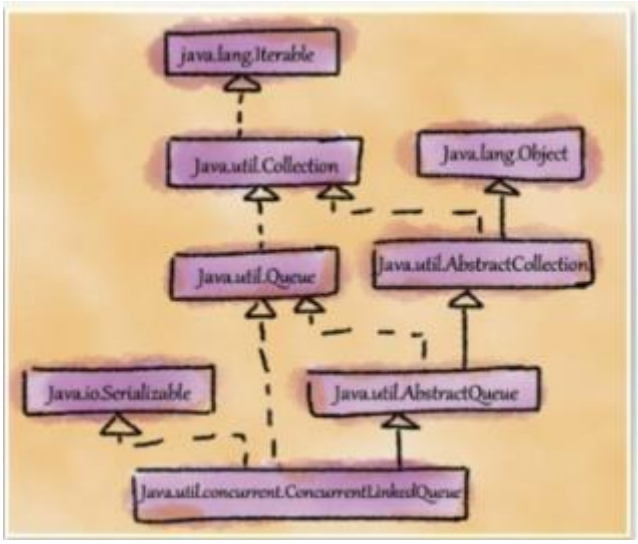
```
#region Objetos

private int aux = 0; ✗
private int contadorDeIteracionesPorUsuario = 0; ✓

#endregion
```

- **Nombre repetir lo que no se debe repetir**

El código que se repite es uno de los factores que hace que sea demasiado cansado el mantenimiento de los códigos, por ello no es correcto estar copiando y pegando los mismos códigos en todo el programa, lo recomendable es realizar métodos compartidos en donde el manejo de parámetro sea dinámico y con eso el mantenimiento se realiza directamente a un solo código.

- **Las funciones hagan lo que se supone deben hacer**

Se refiere a que cada función debe realizar la tarea por la que fue diseñada, si se tiene un método en el que se obtiene un resultado numérico, se debe asegurar que cualquier otra operación debe estar separada en su propio método, de esa forma se puede hacer unirt test por cada método y así asegurando su correcto funcionamiento.



8. Tema: Estructura general de un programa

Los modificadores de acceso son private, default, protected, public.

- **Private**:

La clase sólo puede referirse dentro del mismo archivo en el que se define, además puede definirse dentro de otras clases en el mimo archivo, pero no es recomendable su uso, en un método solo puede ejecutar dentro del mismo archivo en el que se define, se puede definir métodos private dentro de cualquier clase y no pueden heredarse, en el caso de la variable solo se la puede usar para leer o escribir dentro del mismo archivo donde se define y de igual manera no puede heredarse, es decir no es accesible a la subclase.

- **Default**:

No puede usarse en el mismo paquete y no se escribe.

- **Protected**

Se puede referir a la clase desde cualquier parte del código dentro del mismo paquete y además en cualquier clase heredada, así como subclases, icluso puede haber una o más clases con este modificador, el método tiene referencia en el mismo paquete y la variable hace referencia de igual manera en el mismo paquete y es recomendable su uso.

- **Public**

Se puede referenciarse desde cualquier parte del código no importando el paquete en el que esté, además puede o no haber clase de este tipo.

La clase es una parte fundamental de programa, puesto que es el modelo de un objeto que está describiendo, su sintaxis es la siguiente:

```
[mod. acceso][mod. Comportam.] class Identificador{
  // declaración de atributos
  // declaración del constructor
 // declaración de métodos.

}
```

El método de igual manera es el elemento funcional de un objeto y su sintaxis es la siguiente:

```
[mod. acceso][mod. Comportam.]
   <tipo_retorno>Identificador ([argumentos]){
  // declaración de variables
  // acciones.

}
```

El objeto es una instancia actual de una clase, cada objeto se obtiene con la Keyword new. Su sintaxis es la siguiente:

```
NomClase Identificador =
    new NomClase([parámetros]);
```

El constructor es un conjunto de instrucciones, las cuales son diseñadas para inicializar una instancia, la sintaxis es la siguiente:

```
[mod. acceso] NombreClase ([argumentos]){
  // acciones.

}
```

Los paquetes cuando se tienen muchas clases se los puede usar, donde las clases se encuentran físicamente en directorios diferentes, debe ir en la 1ra línea y la sintaxis es la siguiente:

```
package directorio.subdirectorio1,subdirectorioN;
```

Cuando se presenta el caso del uso de muchas clases se puede implementar los paquetes, donde las clases están físicamente en directorios diferentes.

Existen diferentes tipos de variables como las locales, la cuales son declaradas dentro de un método o más que se tienen como parámetros o bloques de código, de igual manera hay las variables de referencia que son definidas fuera de un método y son creadas cuando el objeto es construido usando new.

De igual manera se encuentra la presencia de comentarios en el programa para poder especificar lo que se está realizando o utilizando.

Bibliografía

Anónimo. (27 de Diciembre de 2019). *Clean Code*. Obtenido de https://horneandocodigo.blogspot.com/2019/12/clean-code-que-es-como-usarlo.html

Cátedra de Proyecto . (15 de Marzo de 2019). *Diagramas UML*. Obtenido de https://www.teatroabadia.com/es/uploads/documentos/iagramas_del_uml.pdf

García, M. (7 de Febrero de 2019). *Estructura General de un programa* . Obtenido de https://www.utm.mx/~mgarcia/CursoJava3B-EstructuraGral.pdf

IONOS BY. (9 de Febrero de 2019). *Paradigmas de programación* . Obtenido de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/paradigmas-de-programacion/

NANOPDF. (1 de 7 de 2018). *NANOPDF.com*. Obtenido de NANOPDF.com: https://nanopdf.com/download/entornos-basados-en-combinacion-de-herramientas_pdf

Universidad autonoma del Estado de Hidalgo. (2019). Obtenido de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro15/index.html

Veliz, M. (17 de 7 de 2020). *SCRIBD*. Obtenido de SCRIBD: https://es.scribd.com/document/469500183/1-docx

Prueba Primer Parcial

Crear un menú con los siguientes indicadores

NOTA: 3/4

```
public class P1_Vilaña_Tomás_35 {


   //Atributos
   int num;



   public void mostrar()
   {
      System.out.println(num);
   }
   public P1_Vilaña_Tomás_35() {
   num = 0;
   }


   public P1_Vilaña_Tomás_35(int num) {
      this.num = num;
   }



   public static void main(String[] args) {
     P1_Vilaña_Tomás_35 objeto = new P1_Vilaña_Tomás_35();
    Scanner entrar =new Scanner(System.in);
   int eleccion;
```

```
    eleccion = entrar.nextInt();


    switch(eleccion){
    case 1:
        System.out.println("Ponga un numero: ");
        int dato = entrar.nextInt();
        objeto = new P1_Vilaña_Tomás_35(dato);
        break;
    case 2:
        objeto.mostrar();
        break;
    case 3:
        objeto = new P1_Vilaña_Tomás_35 ();
        break;
    case 4:
        System.out.println("Saliendo del programa..-");
        break;
        }
    }

}
```

Examen Conjunto Primer Parcial

Una vitrina que contenga artículos usando interfaces graficas

NOTA: 4.20/6

Evaluación 1ra Unidad
Puntaje total: 420.00
Puntaje de aprobación: 294.00
Incorrectas restan: No
Abierta: desde 30/11/2021 07:45 hasta 30/11/2021 08:30

Realización

Fecha: 30-nov-2021 07:45:19
Tiempo realización: 00:35:37
Cantidad de veces realizada: 1
Cantidad de respuestas correctas: 17 / 21

✔ Aprobada - 340.00

¿Juanito desea viajar de Miami a Guayaquil, para lo cual toma el vuelo de las 7h00, el cual lo realiza en un avión Aircraft Avianca Boeing 727-21, cuyo Registro es: HK-1803, indicar, que sería Guayaquil?

- Objeto
- Atributo
- Método
- Clase

¿Juanito desea viajar de Miami a Guayaquil, para lo cual toma el vuelo de las 7h00, el cual lo realiza en un avión Aircraft Avianca Boeing 727-21, cuyo Registro es: HK-1803, indicar, que sería Juanito?

- Objeto
- Atributo
- Método
- Clase

¿En un diagrama de casos de usos cual es el elemento que representa tanto a una persona como un sistema?

- (•) Actor
- ( ) Sistema
- ( ) Objeto
- ( ) Clase

¿En qué consiste el Código Limpio?

- ( ) Abrir una ventana en blanco en netbeans para iniciar un programa.
- ( ) Realizar la tabulación de atributos y métodos
- (•) Aplicar técnicas simples que facilitan la escritura y lectura de un código
- ( ) Digitar comentarios sobre los códigos escritos.

¿Cuáles son las partes o bloques que componen un programa?

- ( ) Entrada y Salida
- (•) Instrucciones y Declaraciones
- ( ) Principal y Secundario
- ( ) Atributos

¿Cuántos tipos de datos primitivos existen en Java?

- ( ) 5
- (•) 8
- ( ) 10
- ( ) 9

¿Cuáles son las maneras de lectura de datos en Java?

- (•) Buffered Reader, Scanner, Console
- ( ) Buffered Writer, Scanner, ConsoleReader, Printline, NextLine
- ( ) Scanner, System.in, Buffer Reader
- ( ) Buffered Reader, Input Stream Reader, printf

¿Que indica una excepción en un programa?

- ( ) El estado del programa
- (•) Un error en el programa.
- ( ) Los datos del programa
- ( ) La ejecución del programa.

¿Cuál es el beneficio principal de un encapsulamiento?

- Modificar los datos de una clase de manera más rápida
- Permite resolver los errores que se producen en el desarrollo de un programa
- Los datos de una clase no pueden ser modificados por alguien externo
- Mantiene datos variados que son usados en el desarrollo de todo un programa

¿Qué acción nos permite realizar el método getter?

- Establecer una clase
- Obtener un atributo
- Modificar un atributo
- Llamar a una clase

¿Qué tipo de almacenamiento permiten los arreglos?

- Dinámico
- Estático
- Variable
- Modificable

¿Qué tipo de almacenamiento permiten las colecciones?

- Dinámico
- Estático
- Variable
- Modificable

¿Cuáles son los tipos de colecciones?

- Public y Private
- Heredado y Protegido
- Desordenado y Variable
- List y Set

¿Cuántos elementos tengo en la siguiente declaración de un vector: float vector[]={(float)(3.5),4,9,1,444,20,25};?

- 6
- 7
- 8
- 9

¿Qué instrucción o sentencia es la adecuada para ordenar ascendentemente un vector?

- Arrays.sort(vector);
- Array.sort(vector);
- vector.sort();
- sort.arrays (vector);

¿Qué se debería cambiar en el Método de Ordenación de la Burbuja, para hacerlo más eficiente?

- Contador i
- Contador j
- El contador i y j
- La variable temporal

¿Cómo se debería declarar un vector para registrar las notas de un estudiante?

- ArrayList<Float> notas= new Notas<Float>();
- ArrayList<Float> notas= new ArrayList< >();
- ArrayList<int> notas= new ArrayList<int>();
- ArrayList<float> notas= new ArrayList<float>();

¿Qué método se debería utilizar para ver el tamaño de un vector dinámico tipo ArrayList?

- size()
- length()
- toLowerCase()
- toUpperCase()

¿Qué se debería utilizar para obtener el valor de un número real almacenado en un vector tipo ArrayList?

- vector.get(i);
- vector.getNumero(i);
- vector.getNumero();
- vector.getNumero[i];

¿Qué método se debería utilizar para eliminar un nodo del vector tipo ArrayList?

- delete;
- erase;
- cls;
- remove;

¿Cuál es el valor que se muestra por pantalla
int x=10;
int y=0;
while (y<x) {
 y += x;
}
System.out.println(y);
?

- 8
- 9
- (•) 10
- 11

Vilaña Tomás
CI: 1724725856