TEMA 1

PROGRAMACION ORIENTADA A OBJETOS TEMA 1

* CON ESTA METODOLOGIA, COBRA RELEVANCIA SOBRE QUE SE HACE Y QUE NO SE HACE, (LAS ESTRUCTURAS DE DATOS SON MÁS RELEVANTES QUE LOS PROCEDIMIENTOS).
* LOS DESARROLLOS SE ~~REALIZAN~~ ORGANIZAN ALREDEDOR DE LOS DATOS MANIPULADOS, Y NO ALREDEDOR DE LAS FUNCIONALIDADES. ESTA FORMA DE CONSTRUIR EL PROGRAMA RESULTA MUCHO MAS EFICAZ PUESTO QUE EN LA VIDA DE UN PROGRAMA LOS ELEMENTOS MAS ESTABLES SON LOS DATOS.

* EL SOFTWARE SE EMPAQUETA EN CLASES QUE PUEDEN REUTILIZARSE POSTERIORMENTE. → (AHORRA TIEMPO Y ESFUERZO)

COMPLEJIDAD DEL SOFTWARE

* ES UN CONJUNTO DE INSTRUCCIONES QUE PERMITE AL HARDWARE DE LA COMPUTADORA DESEMPEÑAR TRABAJO UTIL.
* UN SISTEMA DE SOFTWARE NO ES SOLO UNA SUMA DE PROGRAMAS, Y SU COMPLEJIDAD CRECE MAS QUE LINEALMENTE CON LA CANTIDAD DE PROGRAMAS QUE CONTIENE.

MAS CANTIDAD DE ESTOS ELEMENTOS → MÁS COMPLEJOS

1) COMPLEJIDAD DEL DOMINIO DEL PROBLEMA
   - CANTIDAD MUY GRANDE DE REQUISITOS
   - PROBLEMAS DE COMUNICACION ENTRE USUARIO Y DESARROLLADOR
   - CAMBIO DE REQUISITOS DURANTE EL DESARROLLO DEL SISTEMA: SE MODIFICAN REGLAS EN EL DOMINIO DEL PROBLEMA

2) DIFICULTAD PARA GESTIONAR EL PROCESO DE DESARROLLO:
   - SOFTWARE DE GRAN TAMAÑO
   - IMPOSIBILIDAD DE QUE UN SOLO DESARROLLADOR PUEDA COM-

PRENDER TODAS LAS SUTILEZAS DEL SISTEMA Y FUNDAMENTALMENTE LLEVARLO A CABO EN EL TIEMPO REQUERIDO.

- CÍCLOS DE DESARROLLOS LARGOS

FACTORES DE CALIDAD DE SOFTWARE:

PARA SER CONSIDERADO UN SOFTWARE DE CALIDAD DEBE CUMPLIR CON LA MAYORÍA DE LOS SIGUIENTES REQUISITOS.

- PERFORMANCE: (EFICIENCIA): Capacidad de un software para hacer un buen uso de los recursos que manipula. (EN LA ACTUALIDAD NO SE LE DA MUCHA IMPORTANCIA) pero cobra relevancia el tiempo de respuesta.

- PORTABILIDAD: (TRANSPORTABILIDAD): Facilidad con la que un software puede ser transportados a diferentes ambientes físicos o lógicos.

- INTEGRIDAD: Capacidad de un software para proteger sus propios componentes contra los procesos que no tengan derecho de acceso a ellos.

- FACILIDAD DE USO (AMIGABILIDAD/USABILIDAD): Un software es facil de utilizar si se puede comunicar con el de manera comoda.

- CONFIABILIDAD: Informalmente, el software es confiable si el usuario puede contar con el. (PROBABILIDAD DE QUE EL SOFTWARE FUNCIONE COMO DEBE).

- CORRECCIÓN: Capacidad de los productos software de realizar exactamente las tareas definidos por su especificación. (PROGRAMA CORRECTO SI FUNCIONA DE ACUERDO A LA ESPECIFICACION DE LAS FUNCIONES)

- ROBUSTEZ: capacidad para funcionar incluso en situaciones anormales.

- EXTENSIBILIDAD (EVOLUCIONABILIDAD): Se refiere a la facilidad que tienen los productos de adaptarse a cambios en su especificación, es: agregar nuevas funcionalidades.

- REPARABILIDAD: un sistema de software es reparable si permite la corrección de sus defectos.

- REUTILIZACION: capacidad de los productos de ser reutilizados, en su totalidad o en parte, en nuevas aplicaciones.

CARACTERISTICAS DE POO

1) ABSTRACCION: permite simplificar la complejidad del mundo real aislando las características esenciales de un objeto, ignorando los detalles irrelevantes. Esto ayuda a centrarse en lo que es importante para resolver un problema específico, como al conceptualizar un auto por sus funciones básicas (AVANZAR, FRENAR, GIRAR) y no sus detalles técnicos.

2) ENCAPSULAMIENTO: consiste en agrupar datos y métodos que operan sobre esos datos en una unidad llamada "OBJETO", ocultando los detalles internos ~~intrínsecos~~ interfiriendo solo a través de una interfaz pública. Esto protege los datos y facilita el mantenimiento del sistema.

3) MODULARIDAD: Implica dividir una aplicación en partes más pequeñas y manejables llamados modulos. Cada modulo debe ser independiente y cohesivo, permitiendo una mayor productividad en el desarrollo y un mantenimiento más sencillo.

4) JERARQUIA: Se refiere a la organización de objetos en estructuras jerarquicas, como la herencia, donde las subclases pueden heredar y extender el comportamiento de las superclases. Esto permite la reutilización de codigo y la creación de sistemas complejos de manera más eficiente.

BENEFICIOS POO

1) MEJORA LA PRODUCTIVIDAD

* Las clases de objetos son bloques basicos modulares, que permiten la reutilización. (ACORTA TIEMPO DESARROLLO Y COSTOS)

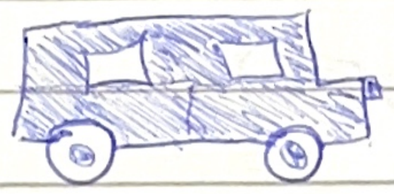
* Cada objeto es una "caja negra" con respecto a objetos externos, esto significa que pueden modificarse sin afectar otras partes. (MANTENIMIENTO SENCILLO Y RAPIDO).

2) MEJORA LA FIABILIDAD

TEMA 2

CONCEPTO DE OBJETO

Un objeto es un representación conceptual de una entidad del mundo real, que puede ser tangible o intangible. Los objetos tiene atributos (como color o tamaño) que describen su estado, y metodos que definen su comportamiento.



- ESTADO (ATRIBUTOS)
- COMPORTAMIENTO (OPERACIONES)
- IDENTIDAD (PROPIEDAD QUE LO DISTINGUE DE LOS DEMAS OBJETOS)

PROPIEDADES DE OBJETOS:

* ESTADO: define la situación del mismo en un momento específico en el tiempo. Los valores que toma un objeto se almacena en variables internas llamados atributos.

* COMPORTAMIENTO: describe las reacciones y acciones de un objeto, de la manera que responde a mensajes o estímulos recibidos. Los distintos forma de respuesta se llaman metodos, y toman la forma de procedimientos o funciones.

* IDENTIDAD: es lo que diferencia a un objeto de todos los otros del mismo tipo. Dos objetos son distintos incluso aun en el caso de que los valores de todos sus atributos coincidan.

- ATRIBUTOS: son las características que describen el objeto. Los atributos de un objeto puede ser datos simples o puede ser una estructura más complejas de datos (ARRAYS, MATRICES, ETC), puede ser de tipo primitivo (ENTERO, CARACTER, ETC) o puede ser objetos.

* ATRIBUTO PROPIO: creado dentro de la capsula del objeto.

* ATRIBUTO HEREDADO: definido en un objeto diferente, antepasado de este (PADRE, ABUELO, ETC)

- METODOS: operacions de un objeto que realizan el acceso a sus propios datos, y los procesa para ~~un objeto determinado~~ obtener un resultado esperado.

Los metodos especifican la forma en que actua el objeto, y permiten manipular y controlar sus propios datos.

No deben tener acceso directo a los estructuras de datos de otros objetos, para utilizar la estructura de dato de otro objeto, deben enviar un mensaje a este.

- CLASE: es una descripción de un conjunto de objetos (O TIPO DE OBJETOS) con características comunes
* Es una abstracción que describe un grupo de instancias con propiedades (ATRIBUTOS) comunes, comportamiento (OPERACIONES) comun, relacions comunes con los otros objetos.

INSTANCIAS DE UNA CLASE

Se pueden crear muchos objetos de la misma clase. Cada vez que se construye un objeto a partir de una clase, se crea lo que se llama una instancia de clase: OBJETO = (INSTANCIA DE UNA CLASE)

El proceso de instanciación consiste en crear un nuevo objeto (concreto) a partir de un molde o especificación de referencia (abstracto, conceptual).
La clase es justamente quién juega este papel de molde para crear el nuevo objeto o instancia.

OBJETO Y ENCAPSULACIÓN:

Un objeto es una unidad que combina datos (atributos) y las operaciones (metodos) que operan sobre esos datos. Los objetos encapsulan (agrupan y ocultan) sus operaciones y atributos.
Encapsulación es la propiedad que vincula operaciones y estado a un objeto, ocultando detalles internos y permitiendo acceso solo a través de metodos.

INTERFAZ PUBLICA

La encapsulación incluye ocultación de información:

* algunas partes son visibles (interfaz pública)
* otras partes son ocultas

La parte pública o interfaz describe que es lo que pueden hacer los objetos de esa clase.

Las partes privadas de esa clase describen la implementación de los metodos, es decir, como hacen lo que saben hacer.

TEMA 4

- **OBJETO**: En el paradigma de objetos, los objetos son los elementos primarios utilizados para construir programas. Dichos objetos son abstracciones de alguna entidad del mundo real (tangible o no).

- **COMPORTAMIENTO**: indica que sabe hacer el objeto, es decir cuales son sus responsabilidades. El comportamiento se especifica a traves del conjunto de mensajes que el objeto sabe responder.

OBJETO	depositar (unMonto)	3 METODOS
CAJA DE AHORRO	extraer (unMonto)	
	saldo()	

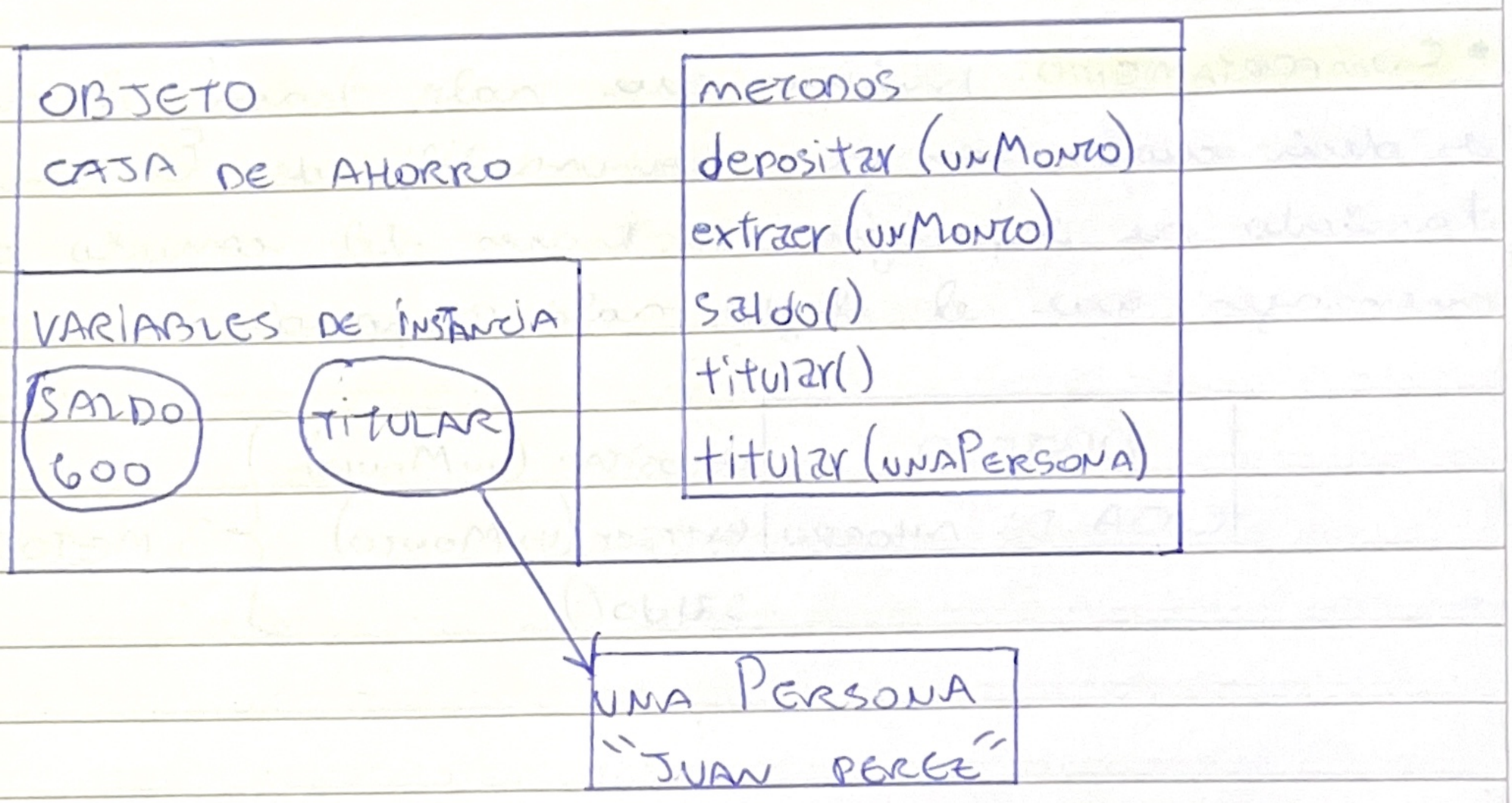
IMPLEMENTACIÓN: METODO VS MENSAJE

Cuando un objeto envia un mensaje a otro, el objeto receptor responde activando el metodo asociado a ese mensaje. Basicamente un metodo es la implementacion (codigo) asociado a un mensaje.

EL ESTADO

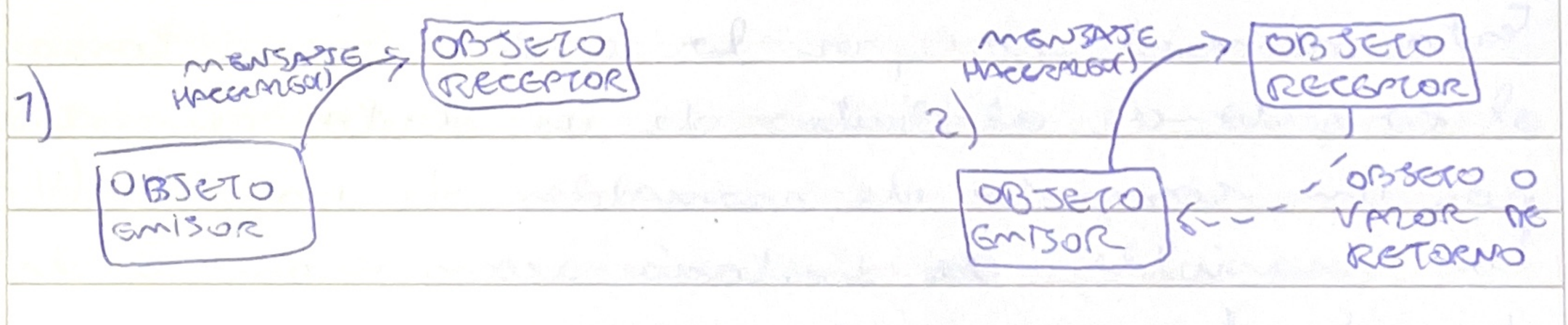
Esta representado por los valores que toma el conjunto de atributos de un objeto, representado por un conjunto de variables de instancia (V.i). Las variables de instancia acompañan al objeto durante todo su ciclo de vida.

Ej: Un objeto caja de ahorro, cuyo estado queda representado por sus V.i, toma un valor particular en el caso de saldo, e indica una relación de conocimiento con el objeto persona, que colabora con el a través de la V.i titular.
A su vez puede determinarse el comportamiento (QUE HACE EL OBJETO), a través del conjunto de metodos (depositar(), extraer(), saldo(), etc).



MENSAJES:

Para llevar a cabo la tarea de un sistema POO, los objetos deben colaborar entre si.
Los objetos interactuan enviandose mensajes.
Un mensaje es una peticion o solicitud que hace un objeto a otro para que realice una tarea.



FORMATO DE MENSAJES

Típicamente un mensaje tiene un nombre, puede o no tener argumentos, y se invoca ~~[illegible]~~ sobre un objeto.

OBJETO_RECEPTOR.NOMBRE_DE_MENSAJE(ARGUMENTOS).

EJ:

UNACUENTA.DEPOSITAR(100);

METODOS: es un conjunto de sentencias que llevan a cabo una acción. Pueden tener parametros de entrada que regulan dicha acción y, puede tener un valor de salida (o valor de retorno).

La diferencia con un procedimiento o función usados en la programación estructurada es que un metodo, al estar asociado con un objeto o clase particular, puede acceder y modificar los datos privados del objeto correspondiente de forma tal que sea consistente con el comportamiento deseado para el mismo.

cada metodo tiene: *

* UNA PARTE PUBLICA: (LA FIRMA DEL METODO) que constituye la interfaz, lo que los demas objetos pueden ver.

* UNA PARTE PRIVADA: (EL CUERPO DEL METODO) ~~que constituye la interfaz, lo que los demas objetos~~ el ocultamiento garantiza que no se propaguen los efectos de los cambios en los estructuras de datos encapsulados dentro del objeto, sobre otras partes del sistema.

El como es un metodo puede implicar tres cosas:

1) MODIFICAR EL ESTADO DEL OBJETO: Esto implica asignar a una V.i un valor distinto al que tiene. EJ:

```
PUBLIC VOID ASIGNARSALDO (DOUBLE UNMONTO){
    SALDO = UNMONTO;
}
```

2) RETORNAR Y TERMINAR: un metodo puede retornar un resultado. EJ:

```
PUBLIC double saldo(){
    return saldo;
}
```

3) COLABORAR CON OTROS OBJETOS: En la mayoria de los casos, para llevar a cabo una determinada responsabilidad, el objeto necesita la colaboración de otros objetos.
EJ: para alquilar una pelicula es necesario saber si el cliente adeuda alguna pelicula:

```
PUBLIC VOID ALQUILAR (UNAPELICULA, UNCLIENTE){
    if (UNCLIENTE.ESDEUDOR()){
        SYSTEM.OUT.PRINTLN("NO ALQUILAR");
    } else {
        UNAPELICULA.ALQUILARA(UNCLIENTE)
    }
}
```

SOBRECARGA DE METODOS:

Una clase puede tener varios metodos con el mismo nombre, siempre que se diferecien en cantidad, orden o tipo de los parametros. Cuando esto sucede se dice que el metodo esta sobrecargado.

La diferecia en los parametros permite al compilador determinar claramente que metodo activar, en funcion de los argumentos que se utilicen en la llamada al metodo(mensaje).

Se puede sobrecargar cualquier metodo miembro de una clase, incluso el metodo constructor:

ES:

```
PUBLIC CLASS MICLASE{
    MICLASE(){

PUBLIC INT SUMA (INT x, INT y) {RETURN x+y;}
PUBLIC INT SUMA (INT x, INT y, INT z) {RETURN x+y+z;}
PUBLIC DOUBLE SUMA (DOUBLE a, DOUBLE b) {RETURN a+b;}
```

NO PUEDEN TENER LA MISMA LISTA DE PARAMETROS

OBSERVADORES y MUTADORES (GETTERS y SETTERS)

Dado que el estado de los objetos es privado, se debe proveer el mecanismo para que los demas objetos puedan acceder a sus atributos. Para ello se utiliza lo que se conoce como observadores (GETTERS), que solo muestran los atributos, sin modificarlos, y los mutadores (SETTERS), que asignan valores o modifican los atributos. GET = OBTENER, SET = ASIGNAR

- DOBLE ENCAPSULAMIENTO:

Lo cual significa que el mismo objeto no accede a sus atributos directamente sino atraves de sus metodos. ES:

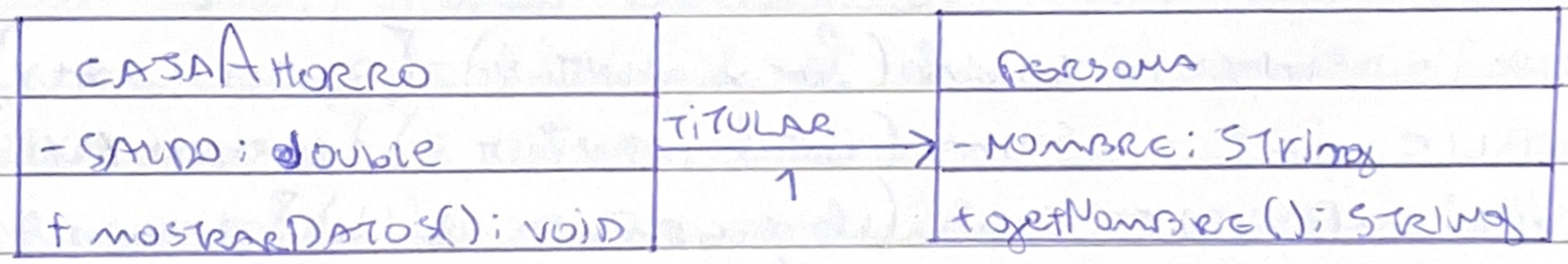
```
CASAAHORRO (INT P_CODIGO, DOUBLE P_SALDO){
        SETCODIGO (P_CODIGO);
        SETSALDO (P_SALDO);
```

FORMAS DE CONOCIMIENTO ENTRE OBJETOS:

1) VARIABLES DE INSTANCIA:

Definen una relación entre un objeto y sus atributos. Constituyen la estructura de la clase/objeto. Las variables de instancia acompañan al objeto desde su creación hasta que muere.

ES:



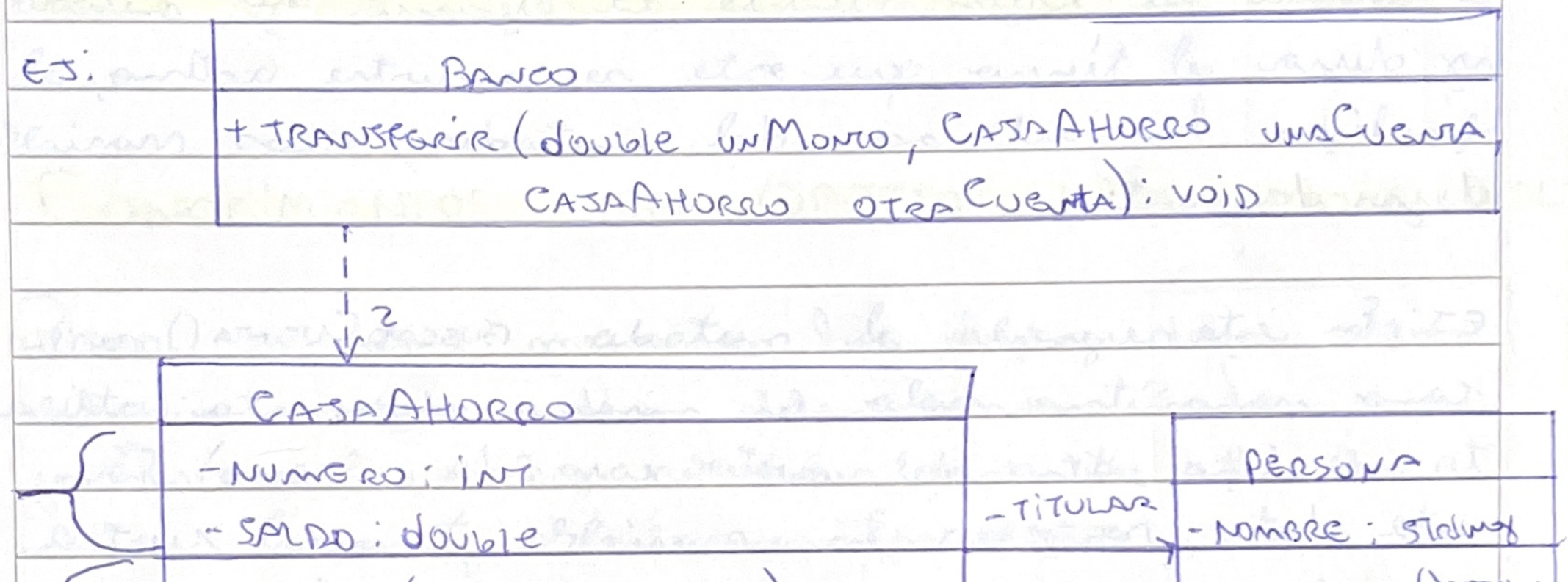

```
PUBLIC CLASS CASAAHORRO{
    PRIVATE DOUBLE SALDO,
    PRIVATE PERSONA TITULAR;
```

```
PUBLIC VOID MOSTRARDATOS(){
    SYSTEM.OUT.PRINTLN("NOMBRE: " + TITULAR.GETNOMBRE());
    //          ("SALDO: " + this.getSaldo());
```

2) PARAMETROS:

Se utilizan para mandar objetos que el objeto receptor necesita para cumplir un requerimiento. Se refiere a los parametros de un mensaje. La relación de conocimiento se establece del cuerpo dentro del cuerpo del metodo, xq dura el tiempo que el metodo se encuentra activo.

Ej:



V.i

Protocolo

```
PUBLIC ELASS CAJAAHORRO(){
  PRIVATE INT NUMERO;
  PRIVATE double SALDO;
  PRIVATE PERSONA TITULAR;
  PUBLIC VOID EXTRAER (double UNMONTO){
        SALDO = SALDO - MONTO;
  }
  PUBLIC VOID DEPOSITAR (double UNMONTO){
        SALDO = SALDO + UNMONTO;
  }
PUBLIC CLASS BANCO{
  PUBLIC TRASFERIR (double UNMONTO, CAJAAHORRO UNACUENTA,
CAJAAHORRO OTRACUENTA){
```

```
        UNACUENTA.EXTRAER(UNMONTO);
        OTRACUENTA.DEPOSITAR(UNMONTO);
    }
}
```

3) VARIABLES TEMPORALES:

Definen relaciones temporales dentro de un metodo. Estas variables se utilizan para nombrar objetos temporalmente. La relación con el objeto se crea durante la ejecución del metodo, y dura el tiempo que este se encuentre activo, al finalizar la activación del metodo, estas variables dejan de existir.

EJ: En este ejemplo, el metodo BUSCARCUENTA() recibe como parametro solo un numero de cuenta (atributo de tipo entero del objeto caja de ahorro). Con este dato, se crea la variable temporal cuenta de tipo CAJAAHORRO, que se utiliza solo como una referencia auxiliar.

```
PUBLIC CLASS CAJEROAUTOMATICO(){
    .....
    PUBLIC VOID REALIZARDEPOSITO(BANCO miBANCO, double unMonto,
INT NROCUENTA){
    CAJAAHORRO CUENTA = miBANCO.BUSCARCUENTA(NROCUETA);
    CUENTA.DEPOSITAR(UNMONTO);
    }
}
```

4) BEUDO-VARIABLES:

Así como un objeto necesita conocer a otro objeto para enviarle un mensaje, es necesario que tenga una forma de nombrarse a sí mismo para poder enviarse mensajes.

Se utiliza this que funciona de manera similar a una V.I, excepto porque no esta definida en ninguna clase ni no se le puede asignar un valor.

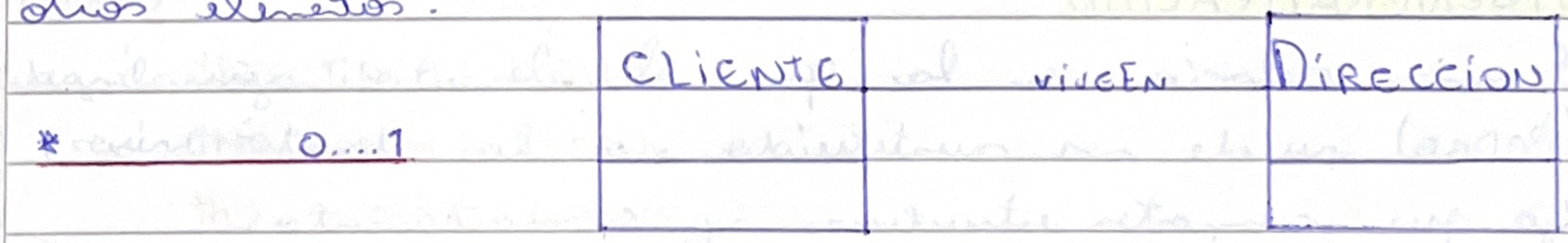
CONOCIMIENTOS ENTRE OBJETOS Y RELACIONES UML

Para enviarse mensajes, los objetos deben estar relacionados, es decir, deben "conocerse". Los mensajes "NAVEGAN" por las relaciones existentes entre las clases a las que pertenecen los objetos.

1) ASOCIACION:

Relación estructural entre entidades (una entidad se construye a partir de otras entidades). Una clase tiene en su estructura a otra clase. Al codificar esto se representa como un atributo que es una instancia de otra clase.

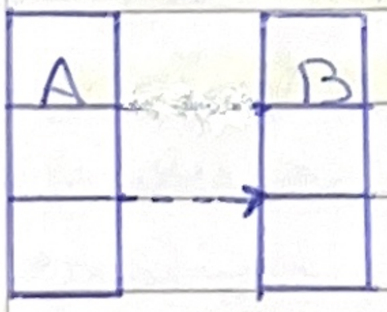
Se representa con una línea continua que a veces incluye una etiqueta y otros elementos.



2) DEPENDENCIA

Es una relación de uso entre dos entidades. Una clase depende de la funcionalidad que ofrece otra clase. Puede verse desde el punto de vista "CLIENTE/SERVIDOR", es decir que existe un servicio.

En UML se representa una flecha discontinua que va desde la clase "CLIENTE" hasta la clase que ofrece el "servicio".

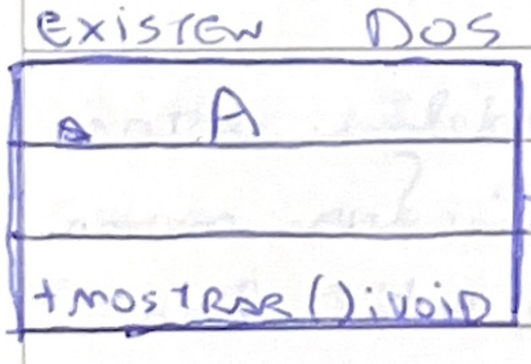


EN EL DIAGRAMA SE PUEDE VER:

- LA CLASE A USA LA CLASE B Y DEPENDE DE ELLA.
- LA A CONOCE A B PERO B NO A.
- LOS CAMBIOS EN B AFECTAN A.

EXISTEN DOS OPCIONES: (VARIABLE TEMPORAL)

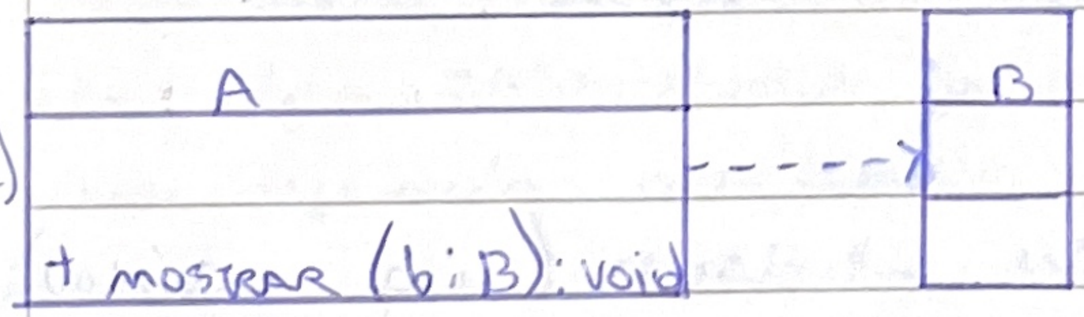
1)



A INSTANCIA UN OBJETO TIPO B

```
PUBLIC CLASS A{
PUBLIC VOID MOSTRAR(){
  b= NEW B();
  SYSTEM.OUT.PRINT(b);
ETC
```

2)



A RECIBE COMO PARAMETRO UN OBJETO TIPO B

```
PUBLIC CLASS A{
PUBLIC VOID MOSTRAR (B b){
  SYSTEM.OUT.PRINT(b).
ETC
```

3) REALIZACION

Es una relación de contrato con otra clase; una clase (INTERFAZ) especifica un contrato que otra clase garantiza que cumplirá. se usa para implementar una interfaz. - - - - - - - - ▷ ES: FIGURA (LO QUE SABE HACER) ⇓ CIRCULO (COMO LO HACE)

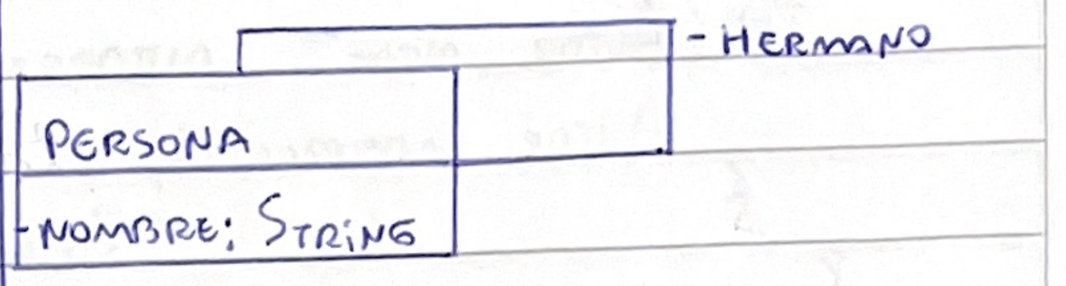
4) GENERALIZACION

Es una relación en la que el elemento generalizado (PADRE) puede ser sustituido por los elementos hijos, ya que comparten estructura y comportamiento.

——————▷

04/09/24

Relaciones involutivas



* Mismo tipo de entidad desempeñando distintos roles.

```
import java.util.ArrayList;

Public class Autor {
    private String nombre;

    Public Autor (String nombre) {
        this.nombre = nombre;
    }

    Public void mostrar (){
        System.out.println ("Autor: " + nombre);
    }

    public String getNombre (){
        return nombre;
    }
}

import java.util.ArrayList;

Public class Libro{
    private String titulo;
    private double precio;
    private ArrayList <Autor> autores;

    Public Libro (String titulo, double precio, ArrayList <Autor> autores){
        this.titulo = titulo;
        this.precio = precio;
        this.autores = autores;
    }
```

TP5

REUTILIZACION: CLASES PREDEFINIDAS

La reutilizacion es considerada como una posible solucion para dar respuesta a un mercado que exige productos y servicios cada vez más fiables, de bajo costo, y entregados a tiempo.

La reutilización en ingeniería del software puede ser analizada desde dos puntos de vista:

- El desarrollo para reutilización: implica desarrollar modulos que podran ser usados en otras ocasiones. El objetivo es elevar al máximo la cantidad de ocasiones y situaciones en que el modulo pueda ser usado.
- El desarrollo con reutilización: significa usar modulos ya desarrollados y probados.

REUTILIZANDO CONTENEDORES/COLECCIONES

El termino colecciones es utilizado para hacer referéncia a datos estructurados en los cuales cada elemento tiene un significado similar, aunque su valor dependa de la posición. También se llaman contenedores porque contiene otros objetos. Los ejemplos más conocidos son: arreglos, listas enlazadas y arboles.

Ventajas son la posibilidad de operar sobre un elemento en particular, sobre un subconjunto de elementos seleccionados mediante un filtro o sobre todos los elementos de la coleccion como conjunto.

TIPOS DE CONTENEDORES:

1A) HOMOGENEOS: Los homogeneos son aquellos que contienen elementos de la misma clase, es decir que cada uno de sus elementos es de la misma clase.

1B) HETEROGENEAS: Un contenedor heterogeneo permite almacenar objetos de diferentes clases.

2A) CONTENEDORES ESTATICOS: un contenedor es estatico o finito si tiene un tamaño predefinido. Se le asigna un tamaño al momento de la declaracion, ay solo puede almacenar un numero fijo de elementos. No se puede modificar el tamaño durante la ejecucion del programa.

2B) CONTENEDORES DINAMICOS: colecciones de elementos, que tienen la particularidad de crecer durante la ejecucion del programa. Este tipo de estructuras no reserva una zona estatica de memoria para su almacenamiento, sino que el espacio ocupado crece o decrece segun las necesidades en tiempo de ejecucion. Las estructuras dinamicas se dividen en:

- LINEALES: PILAS, COLAS, LISTAS.
- NO LINEALES: ARBOLES, GRAFOS.

3A) COLECCIONES ORDENADAS: Es un tipo de coleccion que contiene elementos en una secuencia concreta. Los listos son ejemplos de colecciones ordenadas. JAVA incluye dos clases que permiten trabajar con listos: ARRAYLIST ay LINKEDLIST

3B) COLECCIONES NO ORDENADAS: Es un tipo de coleccion de elementos que se almacenan sin ningun orden.

OPERACIONES CON COLECCIONES:

- ADD() → AÑADIR UN ELEMENTO
- CONTAINS() → COMPRUEBA SI EXISTE UN ELEMENTO
- isEmpty() → COMPRUEBA SI ESTA VACIO
- SIZE() → DEVUELVE EL NUMERO DE ELEMENTOS DEL CONTENEDOR
- REMOVE() → ELIMINA UN ELEMENTO DEL CONTENEDOR

PERSISTENCIA DE OBJETOS

Los objetos pueden ser, segun su persistencia:

- Transitorios o efimeros: el tiempo de vida de un objeto depende directamente del entorno del proceso que los instancio.
- PERSISTENTES: el tiempo de vida de un objeto es independiente del proceso que lo instancio, es decir, trascende el tiempo en memoria y es almacenado conservando su estado en un medio secundario de almacenamiento.

JERARQUIA DE CLASES

Existen muchas clases [illegible] para tratar todo tipos de archivos.

ALGUNAS ESTRUCTURAS DE DATOS EN JAVA

* LINKEDLIST:

Es una lista enlazada de recipientes (nodos) donde cada uno contiene elementos (objetos, otras listas, etc) y uno o dos punteros hacia posiciones de memoria que apuntan al anterior o siguiente nodo.

* ARRAYLIST:

Es una estructura de datos tipo ARRAY dinamica. A diferencia de los arrays clasicos (arrays estaticos), un

ArrayList permite aumentar el tamaño del vector índefinidamente (hasta lo que la memoria permita) y agregar o quitar elementos.
A diferencia de la LinkedList, la ArrayList permite acceder a cualquier elemento de la lista directamente mediante su índice, lo que especialmente adecuado para búsquedas rapidas

* HASHSET

Un HashSet es una estructura de datos que contiene un conjunto de objetos. Permite buscar un objeto dentro del conjunto de forma rapida y facil. Internamente gestiona un array y guarda los objetos utilizando un indice calculado con un codigo hash del objeto.

* HASHMAP

Un hashmap permite guardar elementos, donde cada elemento es un par clave/valor. A diferencia de un array simple donde se guarda el valor en un índice en concreto, un HashMap determina el índice al mismo buscandose en el valor hash (hashcode) generado a partir de la clave.

* TREESET

un treeset mantiene los objetos ordenados y lo que se conoce como un red-black tree, es decir, es un arbol binario balanceado (cada padre tiene como maximo 2 hijos, y cuando se inserta una entrada se auto-balancea de forma que quede un arbol binario sintetico).

03/10/24

POLIFORMISMO: OBJETOS DE DISTINTAS CLASES RESPONDEN A UN MISMO MENSAJE.

CARDINALIDAD: es un concepto en modelado de bases de datos y diagramas de clases en UML que describe el numero de instancias que pueden en una relación entre dos entidades o clases.

- 1 representa un unico
- 0...1 representa opcional
- 1...* representa uno o más
- * representa 0 o más

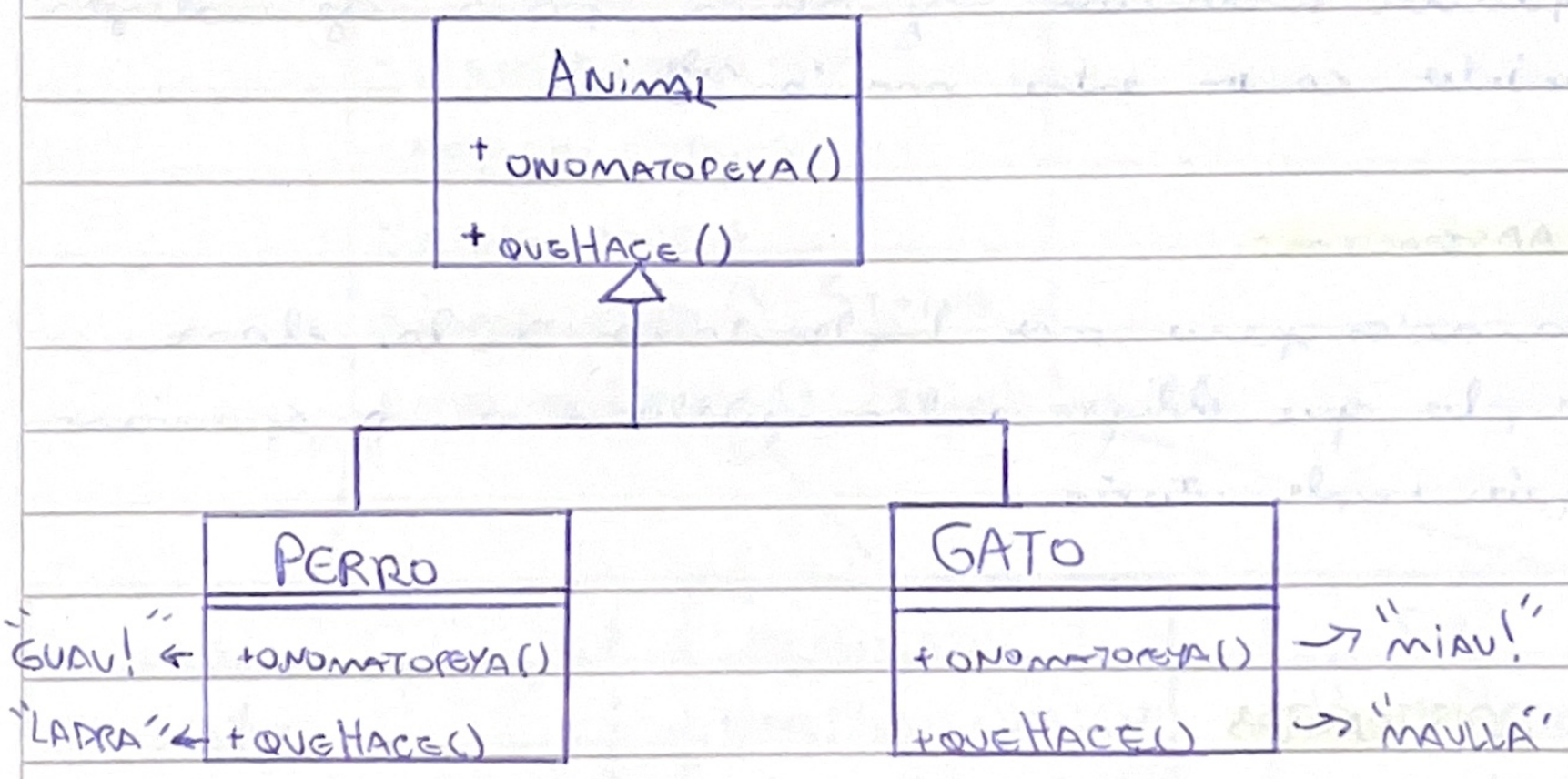
TPS HERENCIA Y POLIMORFISMO

POLIMORFISMO:

Es una característica de un lenguaje de programación que permite a los valores de diferentes tipos de datos ser manejados usando una interfaz uniforme.

EN EL PARADIGMA DE OBJETOS:

Se refiere a la posibilidad de que objetos de clases diferentes respondan a mensajes con el mismo nombre, cada uno de ellos con su propio comportamiento.



VENTAJAS DEL POLIMORFISMO

* CODIGO GENERICO
* OBJETOS DESACOPLADOS
* OBJETOS INTERCAMBIABLES
* OBJETOS REUTILIZABLES
* PROGRAMAR POR PROTOCOLO, NO POR IMPLEMENTACION.

HERENCIA:

Permite que una clase herede metodos y atributos de otra, en el TP utilizar herencia para crear jerarquias de clases (CUENTABANCARIA, CASAAHORRO y CUENTACORRIENTE). Lo que facilita la reutilizacion de codigo.

* REDEFINICION (OVERRIDE):

Utilizado cuando un metodo de la subclase necesita un comportamiento diferente, pero debe mantener la senatura del metodo de la clase padre.

CLASES ABSTRACTAS:

No se pueden instanciar y sirven para generalizar comportamiento comun entre varias subclases.

METODOS ABSTRACTOS:

son declarados pero no implementados en la clase abstracta, lo que obliga a las subclases a proporcionar su propia implementacion.

CLASES ABSTRACTAS

El proposito o utilidad de las clases abstractas es establecer el protocolo que debe implementar cualquier nueva clase que desee extender esa superclase.

DYNAMIC BINDING

Es cuando el programa decide que metodo ejecutar en tiempo de ejecucion, segun el tipo real del objeto, no en tiempo de compilacion. Esto ocurre cuando usas polimorfismo, permitiendo que una referencia a una clase (como ANIMAL) llame al metodo de una subclase (como PERRO) al ejecutar el programa.

TEMA 8: EXCEPCIONES

*El objtivo de este tema es aprender a prever los errores que se pueden predecir durante la ejecución de una aplicación y la forma de capturarlos desde el codigo.

TIPO DE ERRORES

1) ERRORES DE SINTAXIS:

ocurren durante la etapa de compilacion y estan relacionados con la estructura y el formato de codigo.

EJ:

* FALTA DE PUNTO Y COMA
* ERRORES GRAMATICALES (COMO NOMBRE INCORRECTO DE VARIABLES O METODOS).
* ASIGNACION DE UN VALOR INCOMPATIBLE A UNA VARIABLE

EJ PRACTICO:

```
// ERROR USAR "=" EN LUGAR DE "=="
if(x = y){
    ....
}
```

2) ERRORES DE LOGICA:

Son causados por un razonamiento incorrecto del programador.

EJ:

* COLOCAR UN PUNTO Y COMA DESPUES DE UNA CONDICION if O EN LA INICIALIZACION DE UN BUCLE.
* USAR EL OPERADOR "==" PARA COMPARAR OBJETOS O CADENA DE TEXTO, EN LUGAR DE "EQUALS()".

```
// ERROR DE LOGICA: PUNTO Y COMA FUERA DE LUGAR
FOR (INT i = 0; i < 5; i++);
SYSTEM.OUT.PRINTLN (i); // SE EJECUTA UNA VEZ
```

3) ERRORES DURANTE LA EJECUCIÓN - EXCEPCIONES:

En ocasiones, surgen situaciones inesperadas durante la ejecución de un programa.

EJ: Un elemento necesario para que el programa funcione puede faltar o la entrada de ~~de~~ datos puede no cumplir con las expectativas.

COMPROBACIÓN DE ESTADOS

ENFOQUES

* CONSERVADOR: Verifica precondiciones antes de operar.
* OPTIMISTA: realiza la operación y trata de errores mediante excepción sin algo falla.

MANEJO DE EXCEPCIONES

* TRY: contiene codigo que puede generar excepción
* CATCH: atrapa y maneja excepción específica.
* FINALLY: se ejecuta siempre, util para liberar recursos.

HERENCIA
MECANISMOS DE ABSTRACCION

ABSTRACCIÓN significa organizar la realidad para hacerla más aprehensible. Existen diferentes formas:

*CLASIFICACION: Es el más habitual y permite definir un objeto como perteneciente a un tipo o clase mediante instanciación. Es el que permite definir "CORRIENTES" como una provincia.

*GENERALIZACION: Permite definir una clase o tipo como un caso general de varias clases o tipos. Esta relacionado con conceptos de especie o familias. Ej. Se define animales a partir de perro y gato. La operación inversa se llama especialización.

*AGREGACION: Permite definir a un objeto como una simple agrupación de otros. Ej: CURSO DE ALUMNOS.

*COMPOSICION: similar a agregación, pero de modo que si el objeto deja de existir los objetos componentes dejan de tener sentido. Ej: EMPRESA CON EMPLEADOS.

HERENCIA SIMPLE

Mecanismo por el cual una clase puede atender las características de otra, pero puede heredar o tomar atributos de una sola clase padre.
Permite particionar tanto comportamiento como estructura.

HERENCÍA DE COMPORTAMIENTO:

* Una subclase hereda todos los métodos definidos en su superclase

* Una subclase puede redefinir un método heredado, cambiando la forma en la que se comportan sus instancias ante el envío de ese mensaje

* Una subclase no puede anular un método que esté definido en su superclase.

HERENCIA DE ESTRUCTURA

* Una subclase hereda todas las variables de instancia definidos en su superclase.

* Una subclase no puede redefinir un atributo o colaborador. Esto llevaría a tener que utilizar dos variables de instancia con el mismo nombre.

* Una subclase no puede anular la herencia de una V.i

REDEFINICION DE METODOS

En ocasiones es necesario redefinir metodos de la superclase, de modo que metodos con el mismo nombre y caracteristicas se comporten de forma distinta en la subclase. (manteniendo la semantica).

POLIMORFISMO

Se refiere a la posibilidad de que objetos de clases diferentes respondan a mensajes con el mismo nombre, cada uno con su propio comportamiento.

VENTAJAS DEL POLIMORFISMO

* CODIGO GENERICO: al utilizar esta técnica, el objeto emisor es independiente del tipo específico del objeto receptor, lo que permite agregar nuevos tipos de objetos que cumplan con el mismo protocolo sin impactar en el diseño del sistema.

* OBJETOS DESACOPLADOS: se trabaja con un bajo nivel de acoplamiento, es decir, escasa dependencia entre objetos, dado que el objeto emisor sabe lo mínimo indispensable. del objeto al que le esta enviando el mensaje, lo cual brinda flexibilidad al programa ya que evita propagación de cambios locales a el resto del sistema.

* OBJETOS INTERCAMBIABLES: en cualquier parte del programa en que pueda aparecer un objeto "CAJADEAHORRO" puede aparecer un objeto "CUENTACORRIENTE".

* OBJETOS REUTILIZABLES: un objeto bien definido, puede ser utilizado en varios programas. Ej: ~~programas~~ PERSONA

* PROGRAMAR POR PROTOCOLO, NO POR IMPLEMENTACION: El protocolo es el conjunto de servicios que el objeto puede ~~ofrecer~~, es decir lo que sabe hacer.
ofrecer

METODOS ABSTRACTOS

Es aquel que no tiene implementación, ni que por lo tanto nunca será ejecutado.
La utilidad de los metodos abstractos de una clase, es establecer el protocolo que debe implentar cualquier nueva clase que extienda de esta.

CLASES ABSTRACTA

Estas clases se denominan clases abstractos, ni tiene como unica facilidad la declaración de atributos ni metodos comunes que luego utilizaran las clases descendientes.