OBJECT ORIENTED PROGRAMMING

UNIT 1

28 - Oct - 2021

TOPIC: Introduction to Git

- VERSION CONTROL SYSTEMS (VCS):

VCS is a software system that helps teams and individual developers manage change to the work artifacts (toma los espacios de trabajo de cada uno y los maneja en un repositorio) (administra los cambios en todos los artefactos del desarrollo)

All information is in the WORKSPACE (espacio de trabajo)

→ Working Files
→ Staging Area (donde está el código)
→ Repository (Clone)

In the development of a program you get: The code, Data to be handled, description (what is going to be done), Data base, Analysis (how it is going to be solved), Data Files, Design, etc.

Workspaces have a clone or replica of the full repository, but each has a staging area.

Many VCS: most used "GIT"

The artifacts are: Analysis, Design, Project description, Project requirements, Code, Tests, User Stories, User interview, Any document.

→ GIT: is a distributed VCS (va a tener repositorios: uno en la nube y otros en la computadora) *Todos pueden ver los cambios

Track a type of files (SOURCE CODE) líneas de programación

→ BASIC GIT OPERATIONS:

Clone: make an initial copy of the repository (UNA SOLA VEZ)
pull: every time there are CHANGES (Bajo de la nube)
merge: join code (cuando se crea un conflicto entre 2 códigos)
add: add files to repository
commit: name the version (code) (nombre de lo que se hace en el código)
push: upload code to repository central → detallar cambio

Ready CODE: ADD → COMMIT → PUSH

→ JAVA:

JRE → run
JDK → compiler
Netbeans → IDE.

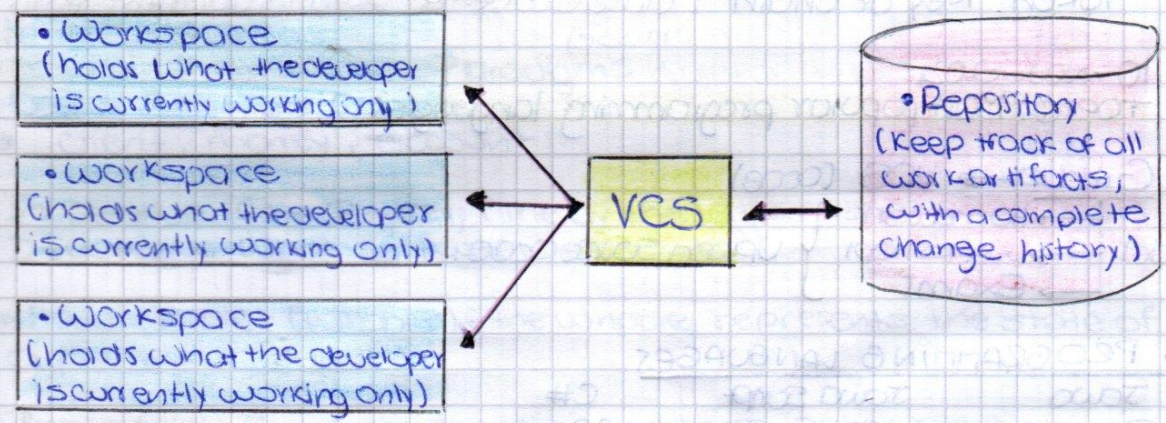
04 - Nov - 2021

TOPIC: Managing Software Development Artifacts.
(artefactos de desarrollo de software)

- **SOME COMMON ARTIFACTS:**

Source code: programming lines (código fuente)
Build scripts: program to compile the system (compilar)
Implementation Notes: programming (desarrollo)
Test cases: the program works or doesn't work (casos de prueba)
Deployment Scripts: installers → instalation (instaladores)
Data Files: information tables: (archivos de datos)
Design documents: graphics, plans (documentos de diseño)
Requirement definitions: (documentos de requerimientos)

VCS: allows the programmer to have WORKSPACES (GITHUB)
5 spaces + a cloud version (repository)



Centralized design → in one place
Github is distributed → each programmer has his own workspace
→ working with the cloud (only one clone)

Workspaces → working Files (archivos de trabajo)
→ Staging Area (temporal del código)
→ Repository (clone) → copia del repositorio

- **STRUCTURE OF THE PROJECT REPOSITORY IN GITHUB**

01. Definition (Definition of the project)
02. Requirements (SRS (IEEE-830) and other documents to gather customers)
03. Documentation (Any document needed: Not model or requirements)
04. UML Diagrams (01. Use cases) vpp and pdf
(02. Class Diagram) vpp and pdf.
05. Unit Tests (test cases document, unit tests)
06. Code (Netbeans code)
07. Other (any other document)

09-Nov-2021
TOPIC: GIT IGNORE (COMMAND PROMPT)

- **COMMANDS TO USE GIT / COMMAND PROMPT**

dir → casos que hay en la carpeta

dir/a → cosas que hay y archivos ocultos
. → directorio actual
.. → directorio que contiene código
more → para ver más dentro del archivo

git add
git add -A → añadir todos
git status → ver los cambios
git commit -m "..." → nombre a la versión del código

Ready code: git add -A → git commit -m" " → git push.

token: Key or chain (clave o cadena para iniciar github en la nube)

10-NOV-2021
TOPIC: Most popular programming languages.

Github → VCS (code)
add → Homework, Workshops, Exams } Upload source code.

- PROGRAMMING LANGUAGES

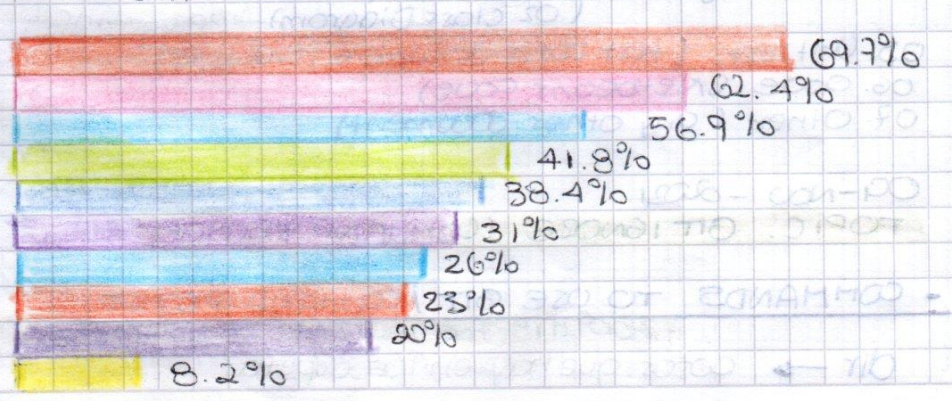
Java	Java Script	C#
Phyton	Kotlin	ADA
C++	Ruby	VB.net
Php	R	Perl.

- PROGRAMMING (POO)

Java → use in this course
↳ appears in the 90's
1st Object Oriented Programming → SIMULA (60's)
2nd Object Oriented Programming → SMALLTALK (70's)
↳ XEROX: Create Xerox Star (1st Graphic operating system)

- MOST POPULAR PROGRAMMING LANGUAGES (1965 – 2019)

LANGUAGES IN 2021:



APP → desktop
→ web
→ mover

2000 (Boom app web: PHP)
→ Base de Datos

Java Sun MicroSystems (Oracle) → IBM
→ My SQL (gestor base de datos)

Phyton (estructurado y orientado a objetos)

- JAVA (Development Enviroment)

1. JRE (JVM) → running applications
2. JDK (Compiler) → coding / compiling
3. Netbeans (IDE) → Integrated Development Enviroment

16-NOV-2021

TOPIC: Programming Paradigms (Objetos en el Sistema)

- Object Oriented (object)
  → Client, money, account.

- Imperative: C++, Pnyton; (line by line) (más usado actualmente)
  Declarative: What do I want him to do? (inteligencia artificial)
  → Handle states (variables): the variable represents the state of a program.

  Procedural: Give a data, write the process, get a result.
  Functional:

- CONCEPTS IN OBJECT ORIENTATION

Characteristics of an object:
→ Unique Identity
→ Encapsulation (private) (meter algo dentro de una caja)
→ Abstraction (public)
→ Classification (group objects): class
→ Subclassification (type)
→ Coupling (type): content, control, etc. (lo min posible)
→ Cohesion
→ Modularity (separate the program into components) (bajo acoplamiento alta cohesión)

- PRINCIPAL FOR OOP

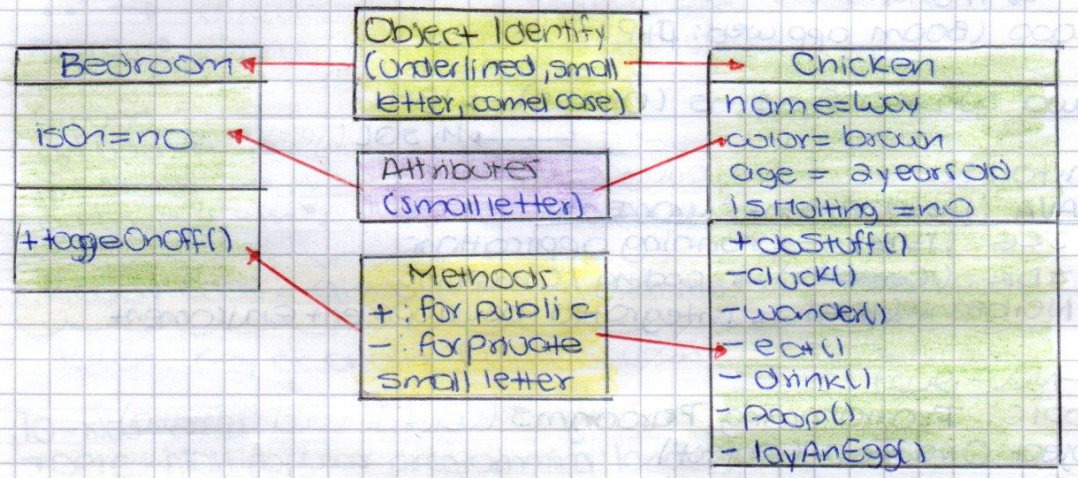
→ Object: anything specific (características propias)
→ Class: general
→ UML: model classes (Unifated Modeling Language)

CAPITAL CAMEL CASE: Classes name (mayúsculas)

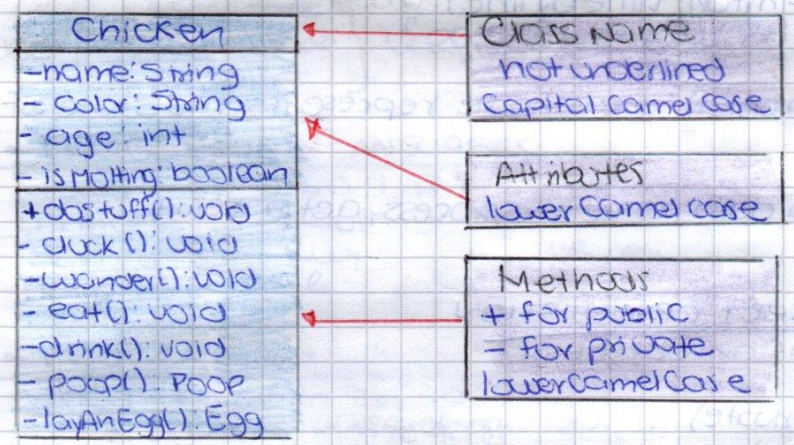
LOWER CAMEL CASE: attributes (variables): public
(minúsculas) functions (methods): public or private
↳ comportamiento

17-NOV-2021 / 18-NOV-2021

TOPIC: Software Objects



- Classes (example)



23-NOV-2021 / 24-NOV-2021

TOPIC: Associations and Multiplicity Constraints

- Clean code

LOWER CAMEL CASE: (minúscula)
UPPER CAMEL CASE: (mayúscula) Only Classes
SNAKE CASE / CAMMEL CASE / LOWER CASE / UPPER CASE /
SMALL LETTER / CAPITAL LETTER: They are different.
SMALL LETTER : lower case (minúscula)
CAPITAL LETTER: upper case (mayúscula)

Examples:
int numberOfWheels; (JAVA) → lower cammel case.
int NumberOfWheels; (CLASS) → upper cammel case
int numberofwheels; (package) → lower case
int number_of_wheels; (C) → snake case

int NUMBEROFWHEELS; (constant) → upper case
int NUMBER_OF_WHEELS; (constant) → upper snake case
int a; → small case letter
int A; → capital letter

- Introduction to UML

UML → Unifoted Modeling language
→ Model Projects
→ Use OOP
→ Diagrams (Analysis, Specification, Implementation)
→ Dependency – Association
RELATIONS

- RELATIONS

→ DEPENDENCY: Use another object
(SIMBOLO) - - - - → Relation to the FUNCTIONS OR METHODS of the object.
→ ASSOCIATION: one object contains one or more other objects
(SIMBOLO) ——→ Relation to the ATTRIBUTES of the object.
Use multiplicity constraint (fixed or variable)

- CLASS AND OBJECT

Role: Como se utiliza en la programación
Example → Vehicle (Class)
Car (Object)

Constructor
METHOD to create from initial values
→ DEFAULT: (por defecto) It is programmed by Java
→ EMPTY: (Nada) Initialize to 0
→ WITH PARAMETERS: (con parametros) Initialize with object values

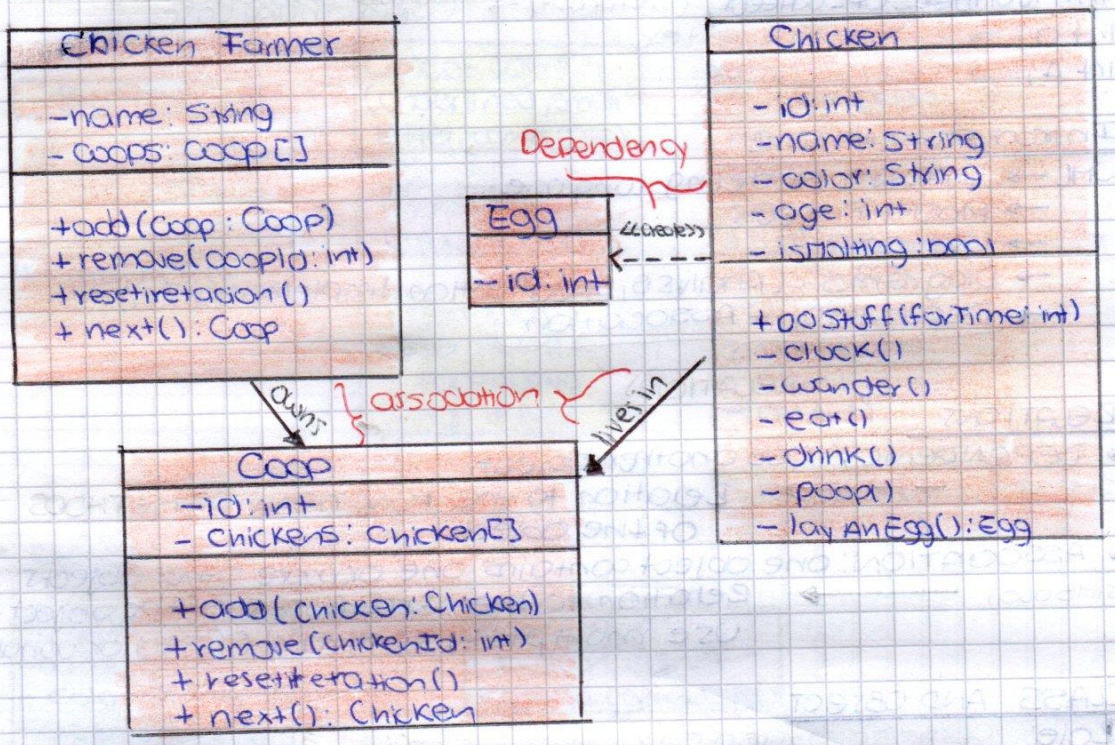
Initialize
→ SETER: set data → SetColor : verb + attribute
→ GETTER: get data → get Color : verb + attribute

Data Type
User Generated
Abstract Data
→ Example:
Employer employer (declarar)
↓ ↓
class variable (object)

employer = new Employer(); (instanciar / inicializar)
↓
method (function)

Employer employer = new Employer();
class Object constructor

→ Example in UML:



→ In Netbeans:

Class: Chicken

• FUNCTIONS:

```
public void doStuff(){
}

private void cluck(){
}

private void wander(){
}

private Egg layAnEgg(){
    return new Egg();
}

private void eat(){
}

private void drink(){
}

private Poop poop(){
    return new Poop();
}
```

• ATTRIBUTES

```
int id;
String name;
String color;
int age;
boolean isMolting;
```

* CREATE:
  - Constructor
  - getter / setter.

UNIT 1 REVIEW

- UML Diagram
  - → Class diagram: Object structure
  - → Use cases: system requirements (system goals)
- Encapsulation: Hide objects → PRIVATE
- Abstraction: more important → PUBLIC
- Objects: main componentes (attributes and methods)
- Software artifacts: code, model, analysis, requirements, others.
- Aggregation: One object contains another → Association
- Project review (inspection) → external entity / walkthrough
  FORMAL – INFORMAL
- Git clone: make a copy
- Remote (Update) → git push
- Create Object → From a class (instantiate – declare)
- NULL: The object is there but it has nothing
- Use case diagram: Organize requirements
- Sorting of various objects: ends in a class declaration
- Setter → methods that initialize values in attributes
- Classes in an object (noun), methods (verbs), attributes (noun)
  not plural

UNIT 2

07 – Dic – 2021

TOPIC: Aggregation

- Introduction

Aggregation is a special type of association that describes links, where one object includes another object as a subpart. (Objeto incluye a otro)

- RULES

→ IRREFLEXIVE: The parent and child object cannot be the same (Padre-hijo no iguales)

→ ASYMMETRICAL: The child object cannot also include the parent object (hijo-padre no incluye)

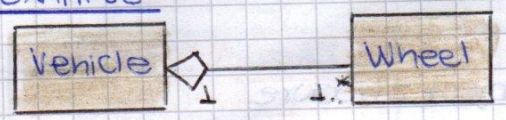
→ TRANSITIVE: If object a includes object b and object b includes object c, then a indirectly includes c

- VERBS

contains
includes
is a subpart of
belongs

SYMBOL: ——◇

• EXAMPLE



This aggregation can be read as

Vehicle < contains / includes > 1 or more wheels

Wheel < is a subpart of / belongs to > 1 vehicle

• COMPOSITION

→ Stronger relationship (relación + dura)
Remove parent object → remove child object.

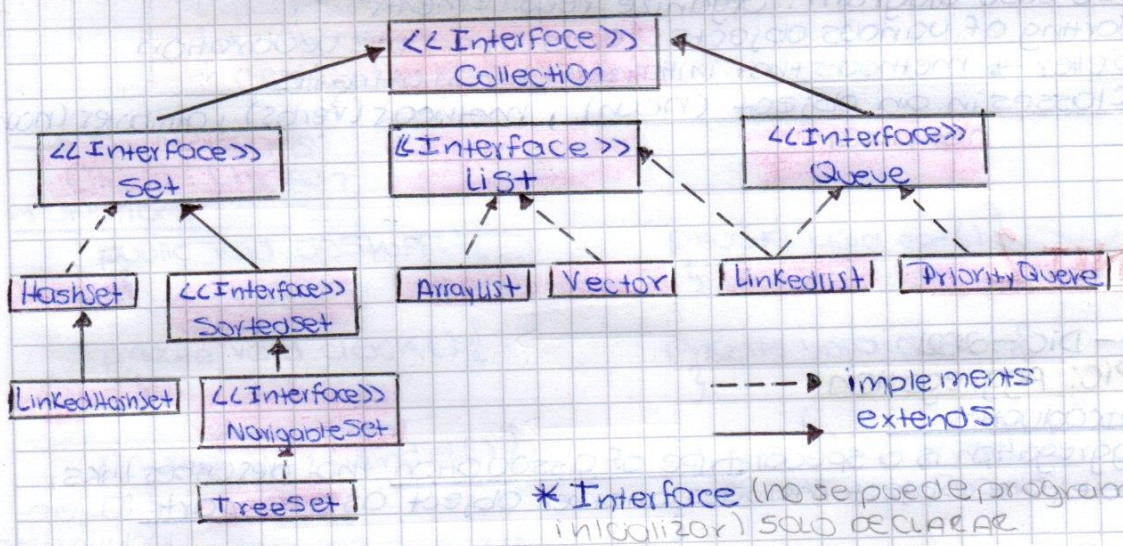
SYMBOL: ◆——

* Other type of association

• JAVA COLLECTIONS

Used for multiplicity from 0 to more (0 0 + ..)

Collection Interface



* Interface (no se puede programar ni inicializar) solo declarar

• EXAMPLES

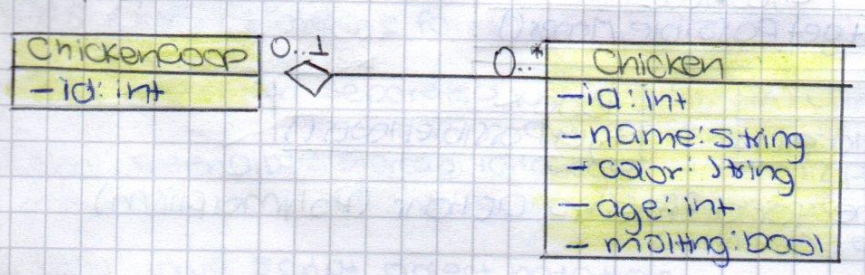
```
List wheels = new List();   (no es posible)
Collection wheels = new Collection(); (no es posible)

List wheels;   (si es posible)
wheels = new ArrayList();   (si es posible)

wheels = new HashSet;   (no es posible)

ArrayList <Wheel> wheels = new ArrayList();
ArrayList <String> names = new ArrayList();
```

09-Dic-2021

TOPIC: Continue → Aggregation

* In the aggregator class goes the rhombus ◇

Example:



→ Referenced variables are of any class

→ PRIMITIVE VARIABLES: int, float, double, boolean.

Chicken chicken;
class object

int a
variable (no tienen listas)

i → A list appears that are methods of the object.

→ Deprecates: Still exist in the system but may disappear at some point (existe pero puede desaparecer)

→ Truncate: remove what is useless (elimina lo que no sirve)

→ Concatenate: Group two strings (agrupar dos cadenas)

→ Remove → Object
→ The index or iterator

14-Dic-2021

TOPIC: Use cases

- Ideas that represent what the users need.

- ACTORS → (things) (los que realizan las acciones): Use the system
- ACTIONS → (verbs) Caso de uso
- ANSWER → what / why

- TOPIC 2:

- Encapsulation: (Private) tener todo guardado
- Inheritance: (herencia) + generic, reclassification
- Polymorphism: treat objects as if they were the same type

- 3 Pillars of OO: Encapsulation
  Inheritance (Specialization)
  Polymorphism (Abstraction)

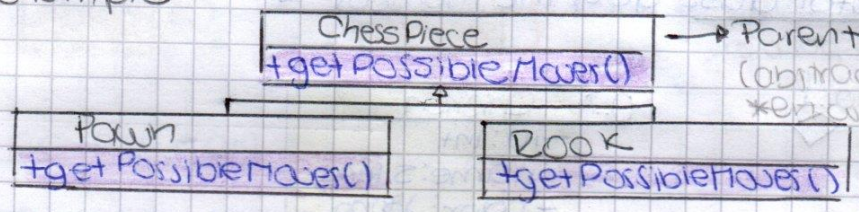
- Generalization / Specialization

- Parent
- Base
- Superclass

- Child
- Derived
- Subclass

- Abstraction: Children program parents
  → *class or methods

Example



→ Parent class (abstracta y metodo) *en cursiva*

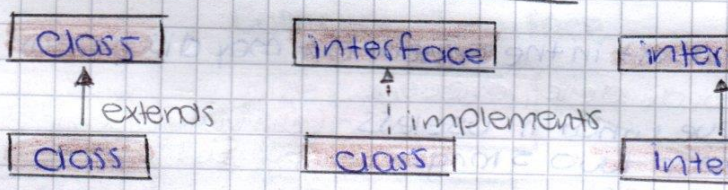
* Same name for differents actions (Polymorphism)

15-Dic-2021

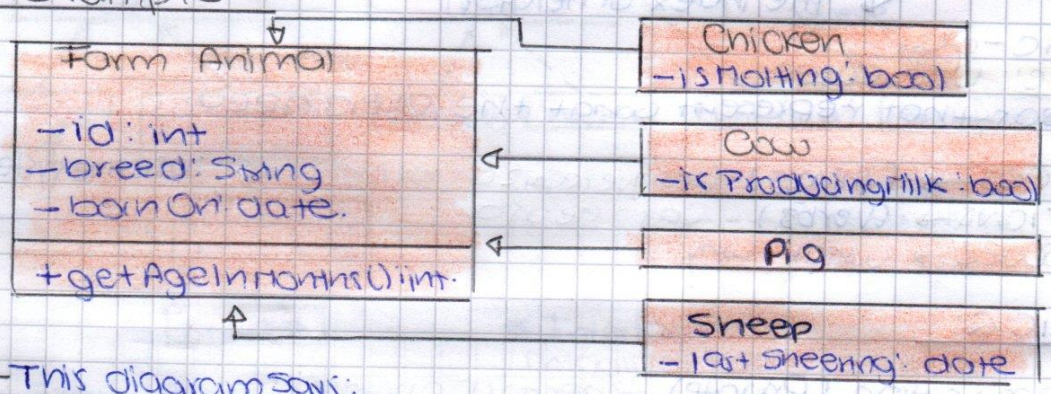
TOPIC: Introduction to Inheritance

In Java, the things that get inherited from A by B, include
- The class variable definitions
- The method definitions and their implementation

• Inheritance and Implementation



• Example



- This diagram says:
  - → A chicken is a farm animal
  - → A cow is a farm animal
  - → A pig is a farm animal
  - → A sheep is a farm animal

INHERITANCE KEY → extends

- Law transitive: inherit from parent class (~~attributes and methods~~ sehered C)

Clasification Revisited (volver a clasificar)

- subsets (cat breeds) → subclassification
- supersets (cats)

• Topic 2: FILES (programa se mantenga en el tiempo)

Files → Data Persistence
- live for a long time in my system
- in the time

→ TYPES (txt: there are forms:)

- CSV: common separated value → (simple) Camila, Teca
  - → common (separar informes en cuanto de texto)
- xml: extensible mercap language <person>
  - tags → brackets. <name>Camila</name>. <lastname>Teca</lastname></person>
- json: java script object notation
  - base → keyvalue {"name": "Camila", "lastname", "Teca"}
  - ↓ GSON

→ DATA FORMATS

→ Examples:

1) Michelle Alvarez General de Brigada 30 2390

2) Darling De las Mercedes | De la Torre | Soldier | 21 | 0

3) Jaime, Elvor, Teniente coronel, 40, 3142

4)
```
<person>
        <name> Edison </name>
        <lastname> Loscano </lastname>
        <position> Full professor </position>
        <birthDate> 12/17/1970 </birthDate>
        <age> 50 </age>
        <salary> 3920 </salary>
</person>
```

5)
```
{"person":
        {"name": "Edison",
        "lastname": "Loscano",
        "position": "Full professor",
        "birthDate": "12/17/1970",
        "age": 50,
        "salary": 3920.00
        }
}
```

• EXCEPTIONS → usado cuando hay errores.
• SUPERCLASS → super

19-Dic-2021

TOPIC: TUTORSHIP → JSON - CSV

- Static: Have functions without needing an Object.

CRUD OPERATIONS.
↓
File management
(manejo de archivos)

C → Create
R → Read
U → Update
D → delete

* Create a new class for file handling (File Manager) in package utils.

- Data persistence: Live data over time

- EXCEPTIONS
  Runtime error handling (manejo de errores en tiempo de ejecución)
  → 3 FORMS → Try: Code to try (a intentar) → File handling Code.
  Catch: Code to handle the error without the program crashing (manejar el error)
  → USE 2 OBJECTS
  IMPORT (Ctrl + Shift + i) { File file = new File(desde donde viene);
  Scanner reader = new Scanner (file);
  ↳ Read data from a file

→ Catch
Use a specific exception.
Catch(Exception ex)
↳ Blanket exception
ex.printStactTrace();
→ Errors for the programmer to see (barrido a final)
→ Is stored in the log (archivo histórico de lo que sucede)

- LIBRARIES
  JAR → (Java Archive)
  WAR → (Web Archive)
  EAR → (Enterprise Archive)

*Use.
GUI → Graphical User Interface (desktop applications)
API → Application Programming Interfaces (base de datos de mapas)
CLI → Command line User interface (desktop applications)
Data Bases → MongoDB (no-SQL): Unstructured.

21 - DIC - 2021

TOPIC: SUMMARY OF TOPICS VIEWED

→ Inheritance (Parent - Children)
→ Polymorphism
→ Abstract classes, abstract methods
→ Interfaces

• Data Formats
  Plain text, csv, xml, json

• Persistence
  Files, databases → Mongo DB (no-SQL)
  MySQL, PostgreSQL, SQL Server (SQL)

→ Expections (try - catch)
→ Unit Tests
→ CUI → GUI
  Command-line User interfaces (desktop App)
  Graphical user interfaces (desktop App)

→ Mobile app (apps) apk (Android), IOS
→ Web App: Python, PHP, JSP, Ruby, etc.
→ OOP Languages: C++, C#, Python.

22 - DIC - 2021

TOPIC: Polymorphism - NO SQL Databases

• Polymorphism

→ ① polymorphic Objects:
```
Animal animal;
  animal = new Tiger();
Device device;
  device = new TvBox();
```

→ ② Overriding
parent class: Device → public void sleep(int time);

Child class: TvBox → @Override
```
public void sleep(int time)
  {...}
```

→ ③ Overloading
Different data types of the parameters / number of parameters.
```
public class TvBox {
  public void sleep(int time) {...}
  public void sleep(int seconds) {...}
  public boolean sleep (int time) {...}
  public void sleep (float time) {...}
  public void sleep(int time, float milliseconds) {...}
}
```

• NO SQL Databases: Mongo DB

Mongo DB (Unstructured) → local data.

→ A database in the cloud is a service accessed through cloud platforms. It provides the same facilities as a traditional database, but with the flexibility of the cloud computing.

CHARACTERISTICS.

1. Access through the cloud
2. The company should not purchase dedicated hardware.
3. It can be managed by the user or it can be managed by the service provider
4. It supports relational databases such as MySQL, Postgres, and NoSQL such as MongoDB, Apache Couch DB (IBM cloudant)

→ DATABASES IN THE CLOUD: Use Mongo DB Atlas (cloud)

Requirements

1. internet connection
2. connection string
3. computer permission (IP)
   ↳ connected to the cloud.

Atlas (cloud) < Access through the cloud / Use as a service

1. Mongo Sheel (línea de comandos)
2. Mongo compas (interfaz gráfica)
3. Program (java netbeans)

Cluster → set of cyues.

Cluster → Data Base → Colecciones

Widgets → open data

04 - January - 2022

Topic: Unit Testing

REVISIONES DE CÓDIGO < WORTRUE / EQUIPO EXTERNO

Polimorfismo → mismo nombre para varias acciones.

Interfaces → Que hace, pero no como lo hace

Clase abstracta → métodos abstractos (no tienen cuerpo)

Widgets → controladores gráficos (botones, cajas de texto, listas, etc)

Formularios → ventana para ingresar datos

Tablas → presentan información tabulada.

• UNIT TESTING

Problems that we have in the software.

* bug: mistake that you do not realize (no se da cuenta).
(se van fundo con el entorno)

- Tucked into the code somewhere
Tester: try out
Black box: technique to use.
- Try out in the small: in the methods.

QUALITY MANAGEMENT

Quality Models → Quality Assurance (tester)
* Hacerle follar al sistema *

Organization Quality Manual → Quality Planning

Project Quality Plan → Quality control

→ Validation: see that the program does what it has to do (hago lo que tengo que hacer)

→ Verification: the program does what it has to do but fine (hago lo que tengo que hacer, pero bien)
is related to the debug.

→ Test Cases { Simple, Complex } Data.

- The person defines the data (INPUT DATA)
- EXECUTION → configure the system (INPUT / OUTPUT)
→ Stimulation: run processes by entering data
Automated.

→ Unit Test
- The program defines the data.
- Observe: compare the good result with the method was checked in the program

→ Test Case Selection
- Select appropriately
- Test System: everything runs.

* looking for a fail or a bug *

When should test be done? → At the beginning of the program development process

UNIT 2 REVIEW

DD MM AA

Review complete Unit 2: (09/01/2022)

Relación con:

- Encapsulamiento: Hacer privado algo para que nadie vea o modifique (PRIVADO)
- Abstraer / Abstracto: Ver solo lo que es necesario (PÚBLICO)
- R • Herencia: hijos comparten lo del papá
- Que se hace en una interface: Declara métodos (no se declara nada más) Les cuenta a otros programas que es lo que va a estar programando pero no les cuenta como
- R • Que implica que hay agregación: Un objeto contiene a otro.
- R • Composición: relación + dura, se elimina el objeto papá se elimina el objeto hijo
- instanciar un objeto: inicializar o crear un objeto
- En herencia el constructor de la clase papá: super
- MongoDB: NO SQL (se puede instalar en cualquier lado)
- MySQL: SQL.
- Mongo DB Atlas: NO SQLL EN la nube
- Operación basicas con bases de datos: Create Read Update Delete.
- Programación en la interfaz gráfica, los controles gráficos se llaman: Widgets
- Ejemplo de controles graficos: botones, cajas de texto, tablas, listas
- Tipos de controles gráficos: swings (estamos usando estos)

*Obligatorio menú's en cualquier App / awt (se usaban antes)

- Como conectarse desde la compu al mongo usando SQUIRL: conección estable a internet, permiso con el ip de la compu, drivers de conección a la base, se necesita la cadena de conección, permisos de acceso.
- R • Generalización / Especialización: concepto que habla de la herencia en la cual se comparten atributos y métodos desde la clase papá a las clases hijas (implements) → para implementar esta relación
- Revisión de código: verificar y validar
  - verificar → todo se haya hecho bien
  - validar → todo se haya hecho

  para esto se corre pruebas en el sistema, y despues hacer el devoling. *no se entiende
- Pruebas: hacer que el programa falle.
- * Polimorfismo: Utilizar el mismo nombre para diferentes acciones pero relacionadas entre sí
- Implementación del polimorfismo: sobrecarga de métodos (un método hacerle varias veces): varia el n° o tipo de argumento.
- Sobreescritura sobreescribir al método de la clase papá (hacer mi propio método)
- R • Objetos polimórficos: asignaciones de objetos a variables de su super clase. (Ejemplo animal Animal = new Lion → declarar)
- Interfaces de programación: permiten modelar el comportamiento de mi programa pero no programarlo, solo modelarlo
- Clases abstractas: puede tener algunos métodos con cuerpo / metodos abstractos (sin cuerpo)
- Arquitectura Modelo (vista controlada): modelo / vista controlada: funcionalidad necesaria pueda ser usado por modelo y vista
  - modelo → toda la estructura de los datos del sistema
  - vista → lo que ve el usuario
- Bases de datos NO SQL: Bases datos NO estructuradas (insertar datos como uno desee), ofrece las operaciones básicas CRUD
- Widgets: con estos se programan formularios
- Navegar: settear los menú's
- Reportes de inf: tablas
- Evento: click del ratón visto!

TIPOS DE RELACIONES:

Dependencias ---> Uso a otro objeto (funciones)

Asociaciones → objeto contiene a otro (atributos)

- Agregaciones ◇ Objeto incluye a otro
- Composiciones ◆ Relación + dura, se elimina objeto papa, se elimina objeto hijo

Implementación o realización ---▷

Generalización o especialización → comparte atributos y métodos de la clase papa a la o las clases hijos.

- herencia (+ especializado)
  - llave: (extends) / heredar del papa: LEY TRANSITIVA.

- Java collections: usado para multiplicidad de 0 a +.
- Interfaz: solo declarar, no inicializar, no programar.
- Uso de casos: ideas that represent what the users need.
  - Actores (things): los que realizan las acciones
  - Acciones (verbs): caso de uso

extra: * Answer what/why

- 3 pillares de OO: Encapsulamiento, inheritance (specialization), polymorphism (abstraction)

Clase es → Abstraction: hijos programan a los padres (clases o métodos)

- Files: programas se mantengan en el tiempo
  - data persistence live for a long time in system.
- Tipos de Files: txt y csv, xml, json (data formats)
- Exceptions: usado cuando hay errores.

UNIT 3

TOPIC: CONTINUED UNIT TESTING

• UNIT TESTING (PRUEBAS DE UNIDAD)

- Termino de programar el código se debe revisar:
  Se puede hacerlo a través de las PRUEBAS
  (verifica que el código haga lo que tenga que hacer, lo haga bien)

→ VALIDACIÓN: El programa haga todo
→ VERIFICACIÓN: El programa haga todo bien.

- Darle mantenimiento al programa (corregir errores)
- UNIT TEST: prueba cada una de mis funciones (métodos del sistema) → métodos devuelvan algún dato
- PRUEBAS DE UNIDAD: Programa que prueba mi programa
  Buscar fallas en el sistema

Ejemplo:
Suma de dos números flotantes)

Tabla de los casos de Prueba (TEST CASE)

Add(float, float)	addend 1	addend 2	Expected result	Actual result
Operación	Primera variable	Segunda variable	Resultado que yo espero	Resultado del Programa.

* Valores constantes NO VAN en la tabla.

→ DEBUG: Errores en tiempo de ejecución (depuración)
→ STEPOVER: Corre el siguiente paso
→ DEBUGGING: semantic error, línea por línea (depurar)

UNIT TEST PLAN
Every function
On the system
QUESTIONS →
- What to test?
- What technique to use?
- How will the test cases be selected?
- When does testing start?
- When will it finish?

→ CUSTOMER: By the system (pide el programa y sus requerimientos)
→ USER: Use the system
* Puede ser la misma o diferente persona.

DEBUG en Netbeans.

1. Botón al lado del RUN
2. F8
3. Abrir ventana de variables

* encontrar el error del programa (no de sintaxis) (no ortográficos) sino de FUNCIONAMIENTO

TOPIC: MODULARITY, COHESION, LOW COUPLING

- **MODULARITY (MODULARIDAD)**
  Poner en un solo sitio las cosas que van a ser útiles en varios lados (ya sea en paquetes (utils) o en una librería)
- **COHESION**
  se relaciona con el SINGLE RESPONSABILITY cuando una función solo hace lo que tiene que hacer.
- **LOW COUPLING (BAJO ACOPLAMIENTO)**
  Tratar de que los componentes conversen lo menos posible.

TOPIC: SOLID PRINCIPLES

- **SINGLE RESPONSABILITY (RESPONSABILIDAD ÚNICA)**
  - Una clase le corresponde una sola función
  - Se rompe este principio cuando haga lo que no tiene que hacer (respecto a la funcionalidad) y cuando no tenga relación con los datos
- **OPEN - CLOSED**
  - clases abiertas a la extensión (extender para conectarse a otro programa), cerrado para la modificación.
  - Serían componentes tan genéricos que puedan ser reutilizados.
- **LISKOV SUBSTITUTION**
  - Uso del polimorfismo (objetos polimórficos) el tigre se comporta como animal, el perro se comporta como animal
- **INTERFACE SEGREGATION**
  - Abrir puertas en otro lado, el usuario no puede ser obligado a usar métodos que no se utiliza, usa lo que necesita usar.
- **DEPENDENCY INVERSION**
  - Si hay clases abstractas no dependen de los componentes concretos de las clases (detalles)
  - Si se declara una clase abstracta llamada Animal y la clase se llama tigre (tigre depende del animal) (animal no depende del tigre)

TOPIC: DESIGN PATTERNS

• DESIGN PATTERNS (PATRONES DE DISEÑO)

- Más útiles, más usados para ingenieros de software
- Se van a utilizar solo 3 patrones.

PATTERS (CRISTOPHER ALEXANDER) Descubre los patrones
→ ① Singleton
→ ② Template Method
→ ③ Strategy

• PANDILLA DE LOS CUATRO (Agrupan 24 Patrones fundamentales)

• PATTERNS (Patrones)

- Cumplir con alguna funcionalidad
- Sirve para resolver un problema
- Concepto de patrones viene de la arquitectura
- Se habla de la REPETICIÓN
- Calidad de código: entendible y legible.

• Solución reusable en un contexto dado, capturar la experiencia de una manera accesible y enseñable, tomar decisiones explicando las consecuencias, mejora la productividad, mejora la legibilidad y mantenimiento del código.

TYPES

- Creational Patterns: Creación e instanciación de objetos
- Structural Patterns: Diseñar la estructura del objeto o proyecto
- Behavioral Patterns: Comportamiento, Cadena de responsabilidad.

* STRATEGY: Varias maneras de hacer lo mismo y quiero saber cual decidir.

• SINGLETON PATTERN

- Dar un punto de acceso único desde cualquier lado
- Puede modificarse una vez cuando se instancia el objeto

ATRIBUTOS → Instancia que se crea solo una vez (Oficina del Presidente)
FUNCIONES → Cambiar datos o procesar datos (llamar o cambiar al presidente)

• TEMPLATE METHOD PATTERN

- Producir variedad de modelos a partir de una estructura general
  CLASE GENERAL → Clase abstracta
  CLASES SECUNDARIAS → clases concretas

- Algunos procedimientos lo hacen las clases secundarias o concretas.

• STRATEGY PATTERN

- Delegar a alguien la decisión de estrategia
- Se debe tomar en cuenta las condiciones o datos antes de tomar la decisión

EJEMPLO:

(ORDENAMIENTO)

- BubbleSort (0 a 5)
- InsertionSort (5 a 10)
- QuickSort (10 a más)