

OBJECT-ORIENTED PROGRAMMING

Advanced Programming

Author: Eng. Carlos Andrés Sierra, M.Sc.
cavirguezs@udistrital.edu.co

Computer Engineer
Lecturer
Universidad Distrital Francisco José de Caldas

2024-III



Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



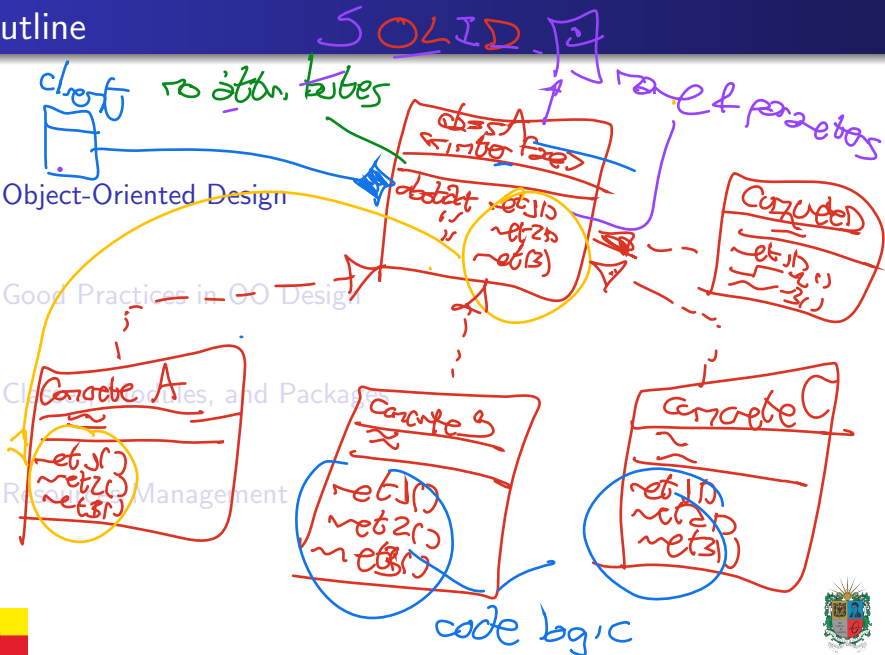
Outline

1 Object-Oriented Design

2 Good Practices in OO Design

3 Class Modules, and Packages

4 Resource Management



Requirements

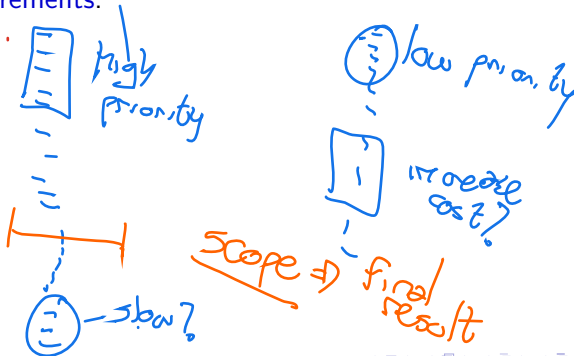
Groove shark

- Requirements Analysis:** It is the process of defining the requirements of a system.

client's need → *Contract* → *User stories*
- Requirements:** They are the functional and non-functional specifications of a system.

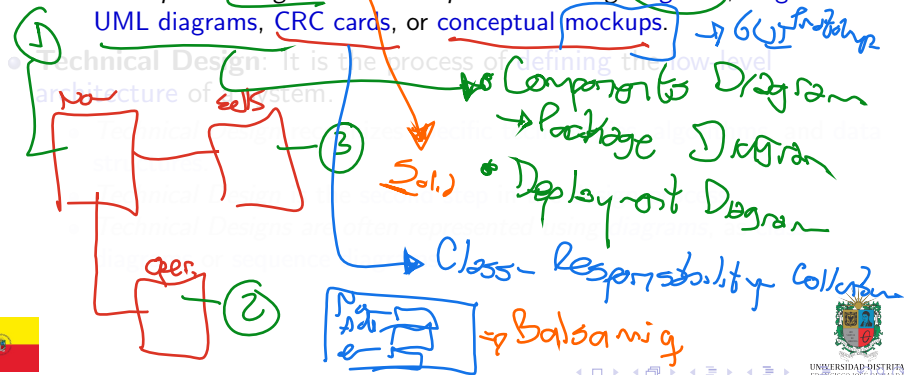
time *performance*
- Trade-offs:** They are the compromises that have to be made between different requirements.

client/user



Conceptual Design, Technical Design

- ① **Conceptual Design:** It is the process of defining the high-level architecture of a system.
 - ② Conceptual Design recognizes appropriate components, relationships, and responsibilities.
 - Conceptual Design is the first step in the design process.
 - Conceptual Designs are often represented using diagrams, as general UML diagrams, CRC cards, or conceptual mockups.



Conceptual Design, Technical Design

→ client

social composition
connected to interfaces

- **Conceptual Design**: It is the process of defining the high-level architecture of a system.
 - *Conceptual Design* recognizes appropriate components, relationships, and responsibilities.
 - *Conceptual Design* is the first step in the design process.
 - *Conceptual Designs* are often represented using diagrams, as general UML diagrams, CRC cards, or conceptual mockups.
- **Technical Design**: It is the process of defining the low-level architecture of a system.

details

objects

- *Technical Design* recognizes specific technologies, algorithms, and data structures.
- *Technical Design* is the second step in the design process.
- *Technical Designs* are often represented using diagrams, as class diagrams or sequence diagrams.

- couple
+ cohesion

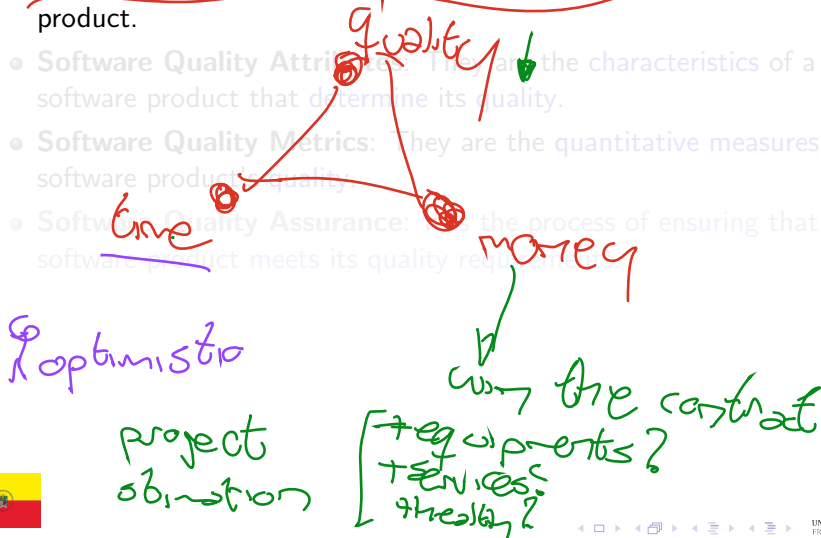
responsibilities



Software Quality

- **Software Quality:** It is the degree of excellence of a software product.

- **Software Quality Attributes:** They are the characteristics of a software product that determine its quality.
- **Software Quality Metrics:** They are the quantitative measures of a software product's quality.
- **Software Quality Assurance:** It is the process of ensuring that a software product meets its quality requirements.



Software Quality

- **Software Quality:** It is the **degree of excellence** of a software product.
- **Software Quality Attributes:** They are the **characteristics** of a software product that **determine** its **quality**.
- **Software Quality Metrics:** They are the **quantitative measures** of a software product's **quality**.
- **Software Quality Assurance:** It is the process of **ensuring** that a software product **meets** its **quality requirements**.



Software Quality

- **Software Quality:** It is the **degree of excellence** of a software product.
- **Software Quality Attributes:** They are the **characteristics** of a software product that **determine** its **quality**.
- **Software Quality Metrics:** They are the **quantitative measures** of a software product's **quality**.
- **Software Quality Assurance:** It is the process of **ensuring** that a software product **meets** its **quality requirements**.



Software Quality

- **Software Quality:** It is the **degree of excellence** of a software product.
- **Software Quality Attributes:** They are the **characteristics** of a software product that **determine** its **quality**.
- **Software Quality Metrics:** They are the **quantitative measures** of a software product's **quality**.
- **Software Quality Assurance:** It is the process of **ensuring** that a software product **meets** its **quality requirements**.



CRC Cards

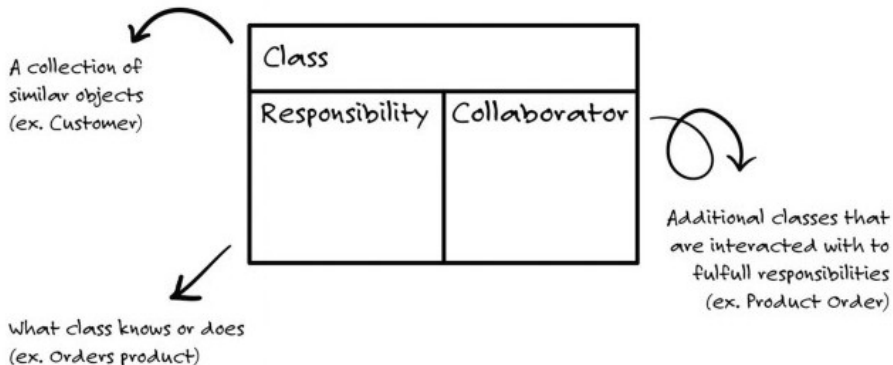


Figure: Prompt: Easy way to design with Classes & Objects.



Basics of Object-Oriented Design I

- **Object-oriented** has become one of the **most traditional and popular paradigms** in software development.
- It is based on the concept of **objects**, which can contain data, in the form of **fields** (often known as **attributes** or **properties**), and code, in the form of **procedures** (often known as **methods**).



Figure: Prompt: Draw several objects sorted by size.



Basics of Object-Oriented Design I

- **Object-oriented** has become one of the **most traditional and popular paradigms** in software development.
- It is based on the concept of **objects**, which can contain data, in the form of **fields** (often known as **attributes** or **properties**), and code, in the form of **procedures** (often known as **methods**).



Figure: Prompt: Draw several objects sorted by size.



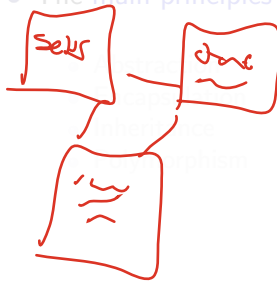
Basics of Object-Oriented Design II



Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The main principles of OOD are:



Basics of Object-Oriented Design II



Figure: Prompt: Draw several objects sorted by size.



- The idea is to design a **system modularly**, and to make it easier to maintain, and to understand. Also the idea is emphasize the **reuse of code**.
- The **main principles** of OOD are:
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism



Objects Categories

- **Concrete Objects:** They are objects that can be instantiated.
- **Abstract Objects:** They are objects that cannot be instantiated.
- **Active Objects:** They are objects that can perform actions.
- **Passive Objects:** They are objects that store data.
- In general, there are three types of objects in OOD:

• **Entity Objects:** They represent the entities of the system, focus on the data that they store.

• **Control Objects:** They represent the control flow between the system and the user.

• **Boundary Objects:** They represent the interaction between the system and the user.



Objects Categories

- **Concrete Objects:** They are objects that can be **instantiated**.
- **Abstract Objects:** They are objects that **cannot be instantiated**.
- **Active Objects:** They are objects that can **perform actions**.
- **Passive Objects:** They are objects that **store data**.
- In general, there are three types of objects in **OOD**:
 - **Entity Objects:** They represent real-world entities, focus on the problem space.
 - **Control Objects:** They represent the logic that manages the system.
 - **Boundary Objects:** They represent the interface between the system and the user.



Objects Categories

- **Concrete Objects:** They are objects that can be **instantiated**.
- **Abstract Objects:** They are objects that **cannot be instantiated**.
- **Active Objects:** They are objects that can **perform actions**.
- **Passive Objects:** They are objects that **store data**.

• In general, there are three types of objects in OOD:

Product :

cls
str

; gether



Objects Categories

DTO → Data Transfer Object
 DAO → Data Abstract Object

- **Concrete Objects:** They are objects that can be **instantiated**.
 - **Abstract Objects:** They are objects that **cannot be instantiated**.
 - **Active Objects:** They are objects that can **perform actions**. *logic*
 - **Passive Objects:** They are objects that **store data**. *no logic*
 - In general, there are three types of objects in **OOD**:
 - Entity Objects: They represent real-world entities, focus on the problem space.
 - Boundary Objects: They represent the interface between the system and the external world.
 - Control Objects: They represent the control logic of the system.
- ↳ business*



Objects Categories

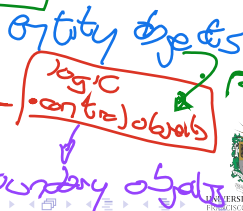
- **Concrete Objects:** They are objects that can be **instantiated**.
- **Abstract Objects:** They are objects that **cannot be instantiated**.
- **Active Objects:** They are objects that can **perform actions**.
- **Passive Objects:** They are objects that **store data**.
- In general, there are three types of objects in **OOD**:
 - **Entity Objects:** They represent **real-world entities**, focus on the **problem space**.
 - **Boundary Objects:** They represent the **interface** between the system and the **external world**.
 - **Control Objects:** They represent the **control logic** of the system.

→ DB connection
→ APIs external



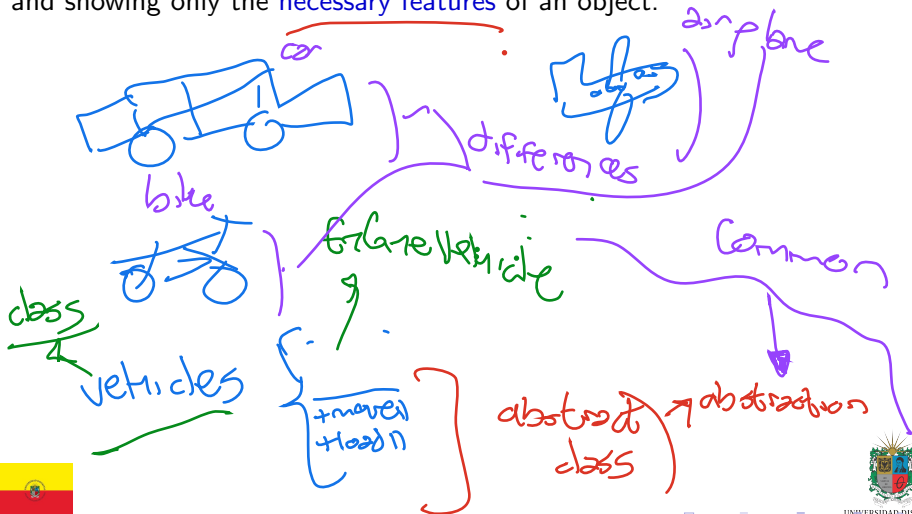
Objects Categories

- **Concrete Objects:** They are objects that can be **instantiated**.
- **Abstract Objects:** They are objects that **cannot be instantiated**.
- **Active Objects:** They are objects that can **perform actions**.
- **Passive Objects:** They are objects that **store data**.
- In general, there are three types of objects in **OOD**:
 1. **Entity Objects:** They represent **real-world entities**, focus on the problem space.
 2. **Boundary Objects:** They represent the **interface** between the system and the **external world**.
 3. **Control Objects:** They represent the **control logic** of the system.



Abstraction in OOD

Abstraction is the process of **hiding** the **complex** implementation details and showing only the **necessary features** of an object.



Encapsulation in OOD

Encapsulation is the process of hiding the internal state of an object and requiring all interactions to be performed through an object's **methods**.

private attributes + getters & setters

```
private int age;
```

```
public void setAge(int newAge) {
```

```
    if (newAge > 0) {
```

```
        this.age = newAge;
```

```
    }
```

data integrity

```
public String convertRate() {
    if (this.age < 10) {
        return "A";
```

```
    } else if (this.age < 15) {
        return "B";
```

```
    } else {
        return "C";
    }
}
```

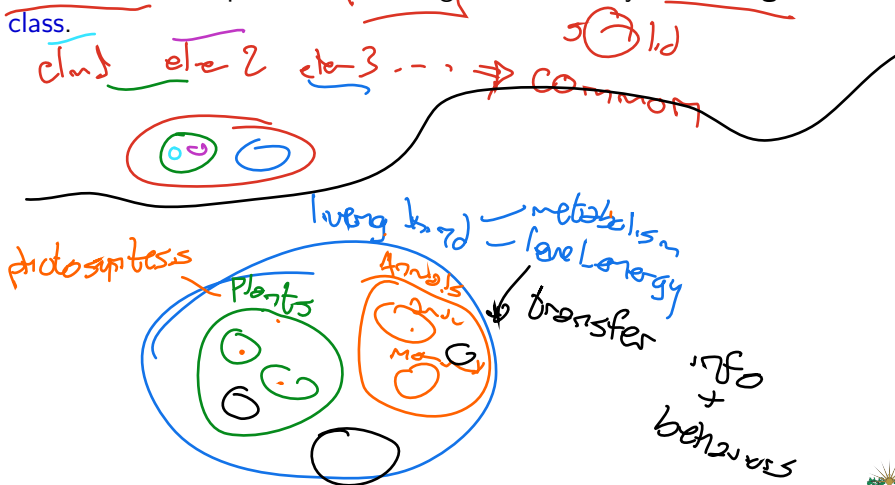
```
public boolean isAdult() {
```

```
    return this.age >= 18 ? true : false;
}
```



Inheritance in OOD

Inheritance is the process of creating a new class by extending an existing class.



Polymorphism in OOD

3 student catch (par)

Polymorphism is the ability of an object to take on many forms. The most common use of **polymorphism** in OOP occurs when a parent class reference is used to refer to a child class object.

methods overload
class Student:

def catch(soccer):
~

def catch(soccer):
~

def catch(tennis):
~

no Python !!
Java !!

methods override
class Mother:

def catch():
~

class Child(Mother):

def catch():
~



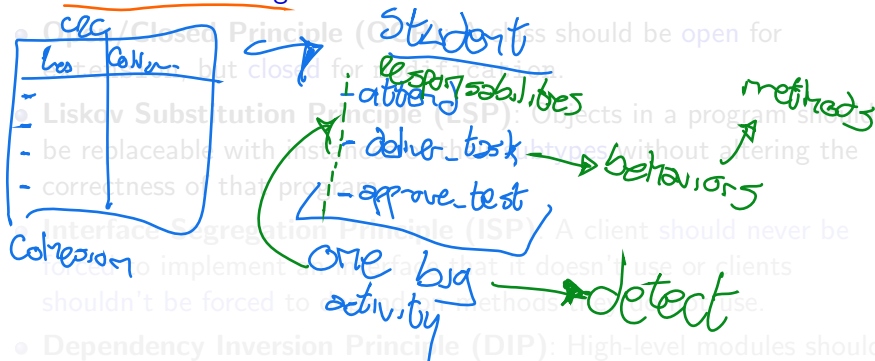
Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



SOLID Principles

- **Single Responsibility Principle (SRP):** A class should have only one reason to change.

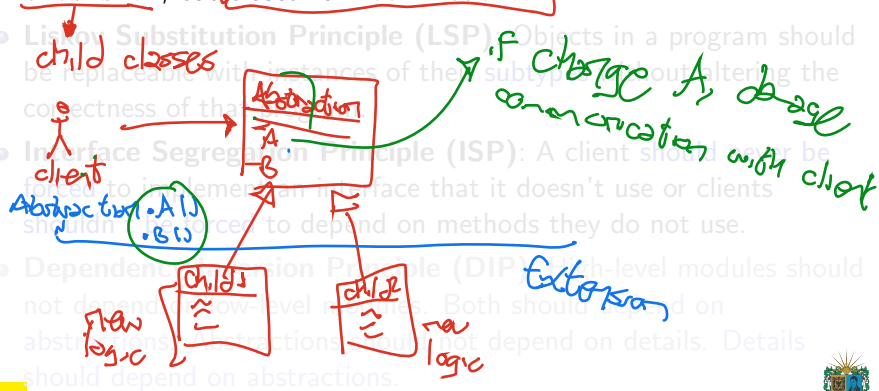


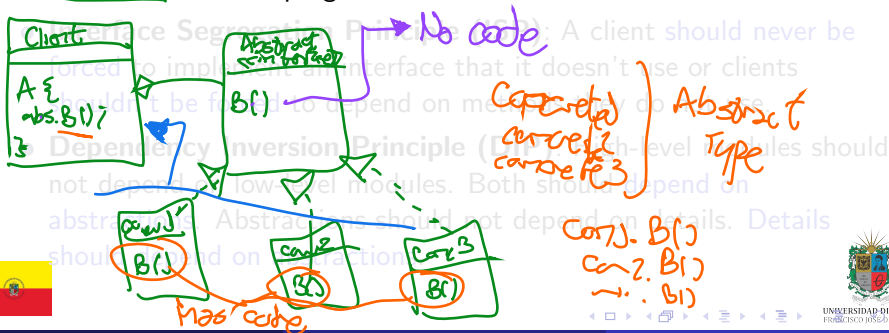
- **Dependency Inversion Principle (DIP):** High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.



SOLID Principles

- **Single Responsibility Principle (SRP):** A class should have only one reason to change.
- **Open/Closed Principle (OCP):** A class should be open for extension, but closed for modification.





SOLID Principles

- **Single Responsibility Principle (SRP):** A class should have only **one reason to change**.
- **Open/Closed Principle (OCP):** A class should be **open** for extension, but **closed** for modification.
- **Liskov Substitution Principle (LSP):** Objects in a program should be replaceable with instances of their **subtypes** without altering the correctness of that program.
- **Interface Segregation Principle (ISP):** A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.



SOLID Principles



- **Single Responsibility Principle (SRP):** A class should have only one reason to change.
- **Open/Closed Principle (OCP):** A class should be open for extension, but closed for modification.
- **Liskov Substitution Principle (LSP):** Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- **Interface Segregation Principle (ISP):** A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
- **Dependency Inversion Principle (DIP):** High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

child implements abstractions



Good Practices

①

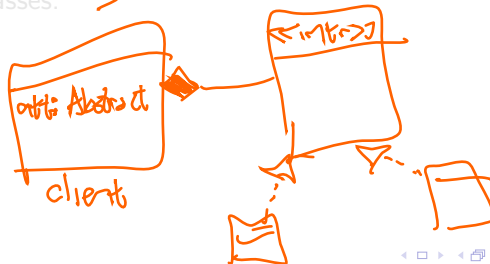
↗ attribute: AnotherClass (Abstract Data type)

- **Composition over Inheritance:** Inheritance should be used **only** when there is a **clear relationship** between the base class and the derived class. In other cases, **composition** should be used. Inheritance is a powerful tool, but it is not always the best tool for the job. Inheritance is a way to achieve polymorphism, but it is **not the only** way to achieve **polymorphism**.

↓ override

- **Code to Interfaces, not Implementations:** This principle is about designing your classes so that they **depend on interfaces** rather than concrete classes.

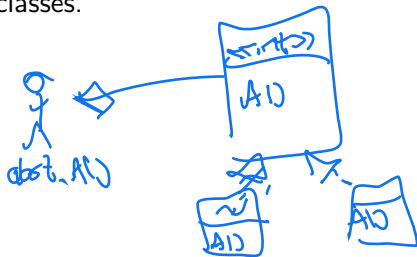
↙ overcharge



Good Practices

- Composition over Inheritance:** Inheritance should be used *only when there is a clear relationship* between the base class and the derived class. In other cases, *composition* should be used. Inheritance is a powerful tool, but it is not always the best tool for the job. Inheritance is a way to achieve polymorphism, but it is *not the only way to achieve polymorphism*.
- Code to Interfaces, not Implementations:** This principle is about designing your classes so that they *depend on interfaces* rather than concrete classes.

not concrete classes



Outline

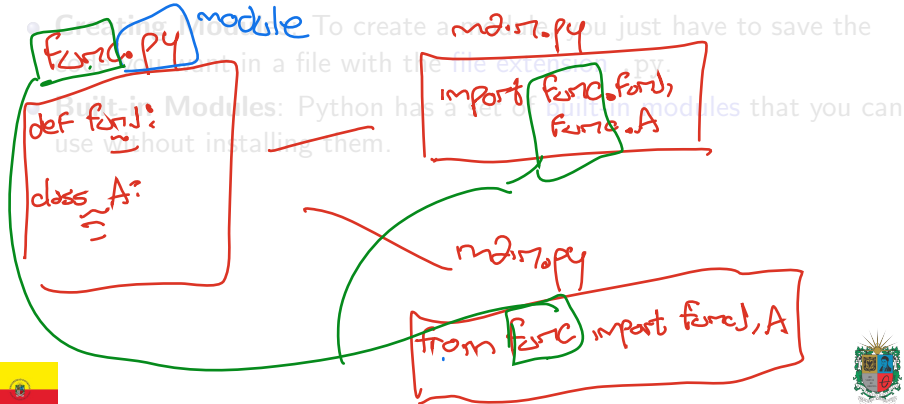
- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 **Classes, Modules, and Packages**
- 4 Resources Management



Modules in Python

Modules are **Python files** that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to **import** it using the import statement.



Modules in Python

Modules are **Python files** that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to **import** it using the `import` statement.
- **Creating Modules:** To create a module, you just have to save the code you want in a file with the **file extension** `.py`.
- **Built-in Modules:** Python has a set of **built-in modules** that you can use without installing them.



Modules in Python

Modules are **Python files** that consist of Python code. They can define functions, classes, and variables.

- **Importing Modules:** To use a module, you have to **import** it using the `import` statement.
- **Creating Modules:** To create a module, you just have to save the code you want in a file with the **file extension** `.py`.
- **Built-in Modules:** Python has a set of **built-in modules** that you can use without installing them.

`list()`
`map()`
`int()`
`max()`

`min()`
`input()`
`print()`

by default
in Python



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with an `__init__.py` file.

- **Importing Packages:** To import a package, you can use the import statement.

- **Virtual Environments:** Virtual environments are a way of creating isolated environments for your Python project.

- **Third-party Packages:** Python has a set of third-party packages that you can install using pip.

Handwritten notes and diagrams illustrating package structure and imports:

- `main.py` (blue)
- `sells_package` (blue)
- `sellers.py` (blue)
- `shopping-cart.py` (blue)
- `payments.py` (blue)
- `--init--.py` (red)
- Diagram: A purple line connects `main.py` to `sells_package`, and an orange arrow points from `sells_package` to the word "folder".
- Diagram: A purple line connects `sells_package` to `from sells_package.sellers` (purple).
- Diagram: A purple line connects `from sells_package.sellers` to `import Seller` (purple).
- Text: `could be empty` (green) with an arrow pointing to `--init--.py`.



- **Creating Packages:** To create a package, you just have to create a `directory` with an `__init__.py` file.
- **Importing Packages:** To import a package, you can use the import statement.

- **Third-party packages:** import classes, functions, ... that you can install using pip.

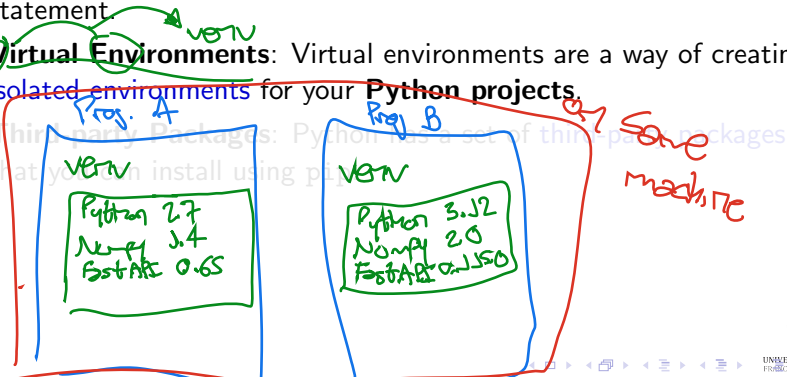
```
import package_name.module_name.classes
           "           "           "
           . functions
```



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with an `__init__.py` file.
- **Importing Packages:** To **import a package**, you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating **isolated environments** for your **Python projects**.



Packages in Python

Packages are a way of structuring Python's module namespace by using *dotted module names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with an `__init__.py` file.
- **Importing Packages:** To **import a package**, you can use the `import` statement.
- **Virtual Environments:** Virtual environments are a way of creating **isolated environments** for your **Python projects**.
- **Third-party Packages:** Python has a set of **third-party packages** that you can install using **pip**.

PyPI.org search

Console (powershell)
pip install tp-package



Packages in Java

Packages are a way of structuring Java's namespace by using *dotted package names*. *project.org - 1 - 1 - java*

- **Creating Packages:** To create a package, you just have to create a directory with a package-info.java file.
- **Importing Packages:** To import a package, you can use the import statement.
- **Third-party Packages:** Java has a set of third-party packages that you can use in your projects.
- **Maven:** Maven is a build automation tool used primarily for Java projects.



Packages in Java

Packages are a way of structuring Java's namespace by using *dotted package names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with a `package-info.java` file.
- **Importing Packages:** To **import a package**, you can use the **import** statement.

• **Third-party Packages:** Java has a set of third-party packages that you can use in your projects.

import project.org.src.package.class-name

- **Maven:** Maven is a **build automation tool** used primarily for **Java projects**.



Packages in Java

Packages are a way of structuring Java's namespace by using *dotted package names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with a `package-info.java` file.
- **Importing Packages:** To **import a package**, you can use the `import` statement.
- **Third-party Packages:** Java has a set of **third-party packages** that you can use in your projects.
- **Maven:** Maven is a **build automation tool** used primarily for Java projects.

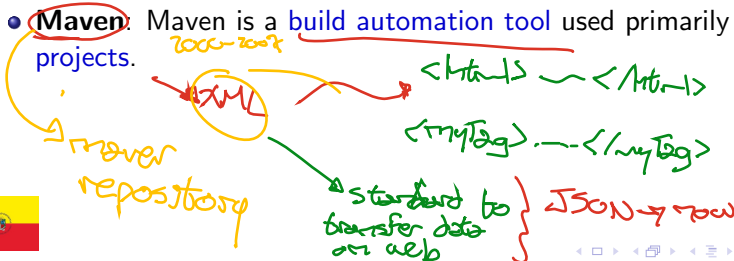
less common
than in Python



Packages in Java

Packages are a way of structuring Java's namespace by using *dotted package names*.

- **Creating Packages:** To create a package, you just have to create a **directory** with a `package-info.java` file.
- **Importing Packages:** To **import a package**, you can use the `import` statement.
- **Third-party Packages:** Java has a set of **third-party packages** that you can use in your projects.
- **Maven**: Maven is a **build automation tool** used primarily for **Java projects**.



Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management**



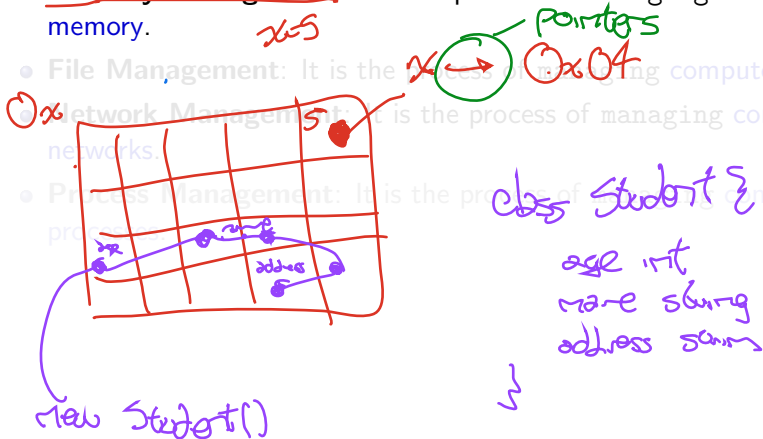
Resources in Computation

- **Memory Management:** It is the process of managing computer memory.

- **File Management:** It is the process of managing computer files.

- **Network Management:** It is the process of managing computer networks.

- **Process Management:** It is the process of managing computer processes.



```
class Student {
    age int
    name string
    address sum
}
```



Resources in Computation

- **Memory Management:** It is the process of managing computer memory.
- **File Management:** It is the process of managing computer files.

• **Network Management:** It is the process of managing computer networks.

• **Process Management:** It is the process of managing computer processes.

• CSV (Comma separated value)

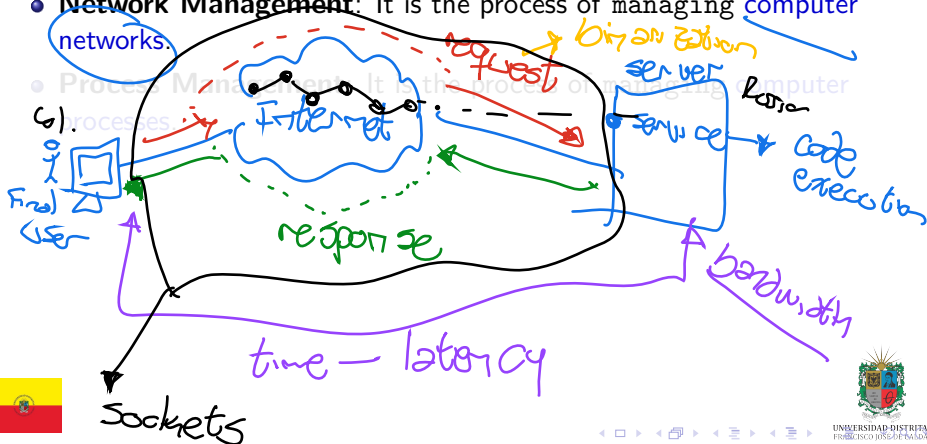
• JSON

Latency { Mem RAM 10~s
HDD 20~s



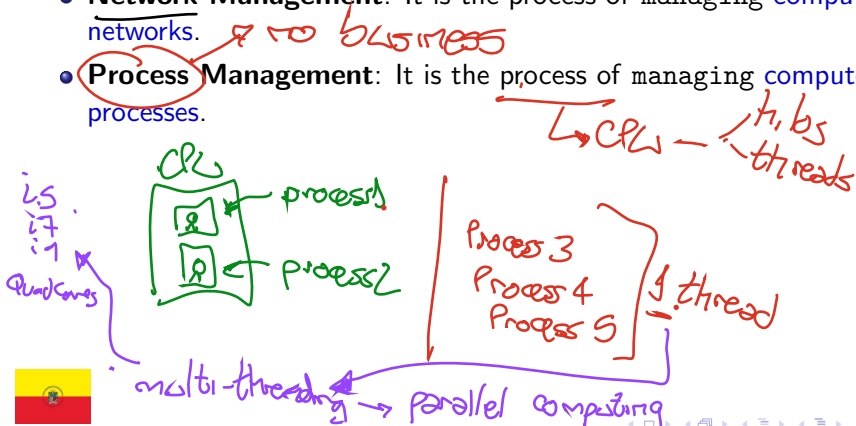
Resources in Computation

- **Memory Management:** It is the process of managing computer memory.
- **File Management:** It is the process of managing computer files.
- **Network Management:** It is the process of managing computer networks.



Resources in Computation

- **Memory Management:** It is the process of managing **computer memory**.
- **File Management:** It is the process of managing **computer files**.
- **Network Management:** It is the process of managing **computer networks**.
- **Process Management:** It is the process of managing **computer processes**.



- let save an object as a file

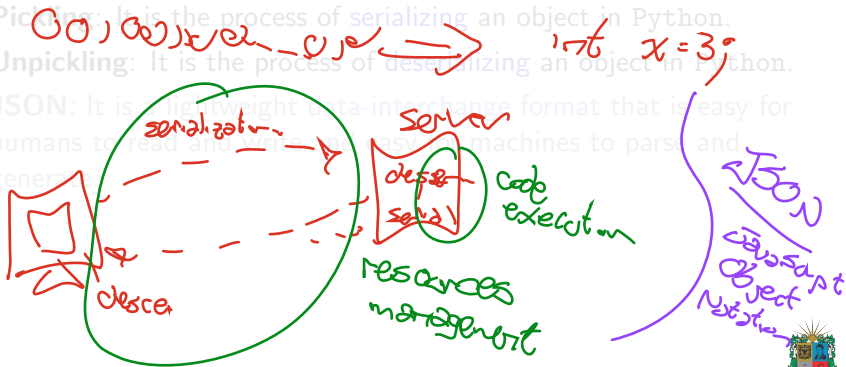


- **Serialization:** It is the process of converting an object into a **stream of bytes**.
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.

-
- The diagram illustrates the process of serializing and deserializing an object in Python. It shows a flow from 'Object' to 'Serialized Object' via a 'Serializer' and back via a 'Deserializer'. The diagram is annotated with 'Pickling' and 'Unpickling'.
- ```

graph LR
 Object -- "Pickling (Serializer)" --> SerializedObject[Serialized Object]
 SerializedObject -- "Unpickling (Deserializer)" --> Object

```
- Handwritten notes and annotations include:
- Pickling:** It is the process of serializing an object in Python.
  - Unpickling:** It is the process of deserializing an object in Python.
  - JSON:** It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.
  - Serializer:** A box labeled 'Serializer' with an arrow pointing from 'Object' to 'Serialized Object'.
  - Deserializer:** A box labeled 'Deserializer' with an arrow pointing from 'Serialized Object' back to 'Object'.
  - Code:** A handwritten note 'Code' is present near the 'Serialized Object' box.
  - JSON:** A handwritten note 'JSON' is present near the 'Serialized Object' box.



# Objects Serialization

- **Serialization:** It is the process of converting an object into a **stream of bytes**. *Java*
  - **Deserialization:** It is the process of converting a **stream of bytes** into an object.
  - **Pickling:** It is the process of **serializing** an object in Python. *import pickle*
  - **Unpickling:** It is the process of **deserializing** an object in Python.
- JSON:** is a lightweight **data-interchange** format that is easy for humans to read and write and easy for machines to parse and generate.



# Objects Serialization

- **Serialization:** It is the process of converting an object into a **stream of bytes**. *Julia*
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.
- **Pickling:** It is the process of **serializing** an object in Python.
- **Unpickling:** It is the process of **deserializing** an object in Python.
- **JSON:** It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. *Python*



# Objects Serialization

- **Serialization:** It is the process of converting an object into a **stream of bytes**.
- **Deserialization:** It is the process of converting a **stream of bytes** into an object.
- **Pickling:** It is the process of **serializing** an object in Python.
- **Unpickling:** It is the process of **deserializing** an object in Python.
- **JSON:** It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

all keys must be strings

```
{
 "name": "Andres",
 "phone": "12345",
}
```

key-value

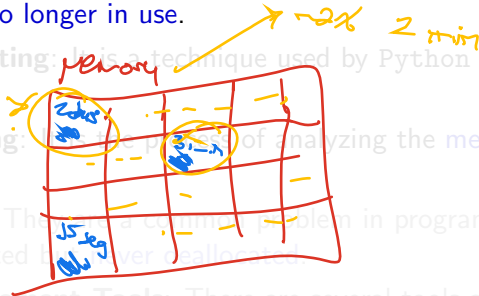
JS → JSON  
 Python → Dict  
 Java → Map



# Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.

- **Reference Counting:** It is a technique used by Python to manage



- **Memory Profiling:** It is the process of analyzing the memory usage of a program.
- **Memory Leaks:** There is a common problem in programming where memory is allocated but never deallocated.
- **Memory Management Tools:** There are several tools available for memory management in Python.



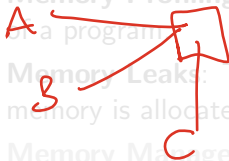


# Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage** **memory**.

- **Memory Profiling:** It is the process of analyzing memory usage of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but **never deallocated**.
- **Memory Management Tools:** There are several tools available for memory management in Python.

fail → no change spc



# Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage** memory.
- **Memory Profiling:** It is the process of analyzing the **memory usage** of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but **not deallocated**.
- **Memory Management Tools:** There are several tools available for memory management in Python.



# Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage** memory.
- **Memory Profiling:** It is the process of analyzing the **memory usage** of a program. *→ LEX / server*
- **Memory Leaks:** They are a common problem in programming where memory is allocated but **never deallocated**.
- **Memory Management Tools:** There are several tools available for memory management in Python.



# Memory Management

- **Garbage Collection:** It is the process of automatically *reclaiming* memory that is **no longer in use**.
- **Reference Counting:** It is a technique used by Python to **manage** memory.
- **Memory Profiling:** It is the process of analyzing the **memory usage** of a program.
- **Memory Leaks:** They are a common problem in programming where memory is allocated but **never deallocated**.
- **Memory Management Tools:** There are several tools available for **memory management** in Python.

↳ monitoring — Prometheus  
                            Grafana



# Threds, Processes, and Concurrency

- **Threads:** They are the smallest unit of execution that can be scheduled by an operating system.

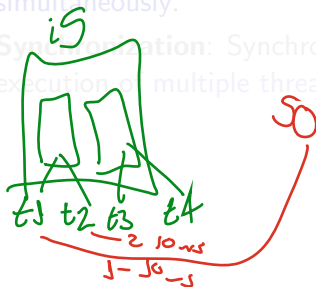
- **Processes:** They are the smallest unit of execution that can be scheduled by an operating system.

CTF1 + ALE + Super  $\Rightarrow$  Task Manager

- **Parallelism:** It is the ability of a program to execute multiple tasks simultaneously.

Firefox Process  
 $\sim \vdots$  Threads

- **Synchronization:** Synchronization is the process of coordinating the execution of multiple threads or processes.



| CR   | thread | time |
|------|--------|------|
| apt1 | t1     | 10ms |
| cpu1 | t5     | 2ms  |
| cpu2 | t3     | 15ms |
| cpu3 | t4     | 10ms |



- Proc1  $\leftarrow$  t<sub>h1</sub>  
                   $\leftarrow$  t<sub>h2</sub>  
Proc2  $\leftarrow$  t<sub>h3</sub>  
                   $\leftarrow$  t<sub>h4</sub>  
                   $\leftarrow$  t<sub>h5</sub>

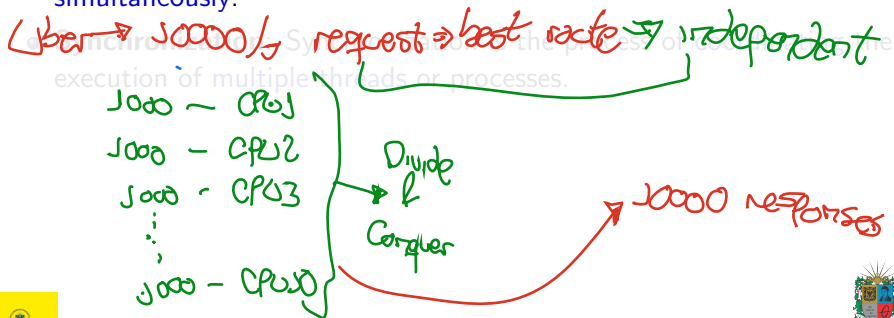
↓  
process

— it could be split into  
on threads



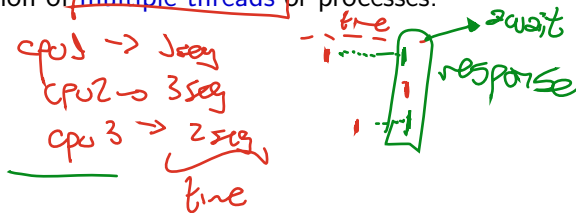
# Threds, Processes, and Concurrency

- **Threads:** They are the **smallest unit of execution** that can be scheduled by an operating system.
- **Processes:** They are the **largest unit of execution** that can be scheduled by an operating system.
- **Parallelism:** It is the ability of a program to execute **multiple tasks simultaneously**.



# Threds, Processes, and Concurrency

- **Threads:** They are the **smallest unit of execution** that can be scheduled by an operating system.
- **Processes:** They are the **largest unit of execution** that can be scheduled by an operating system.
- **Parallelism:** It is the ability of a program to execute **multiple tasks simultaneously**.
- **Synchronization:** Synchronization is the process of **coordinating** the execution of **multiple threads** or processes.

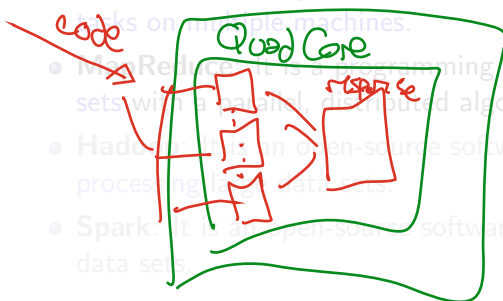




# Parallel Computation

- **Parallel Computation:** It is the process of executing multiple tasks simultaneously.

→ Same machine



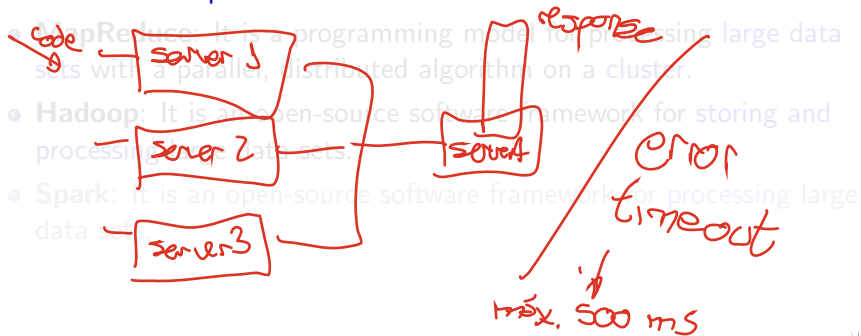
→ On-premises

- **Distributed Computation:** It is the process of executing multiple tasks on multiple machines.
- **Map-Reduce:** It is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
- **Hadoop:** It is an open-source software framework for storing and processing large data sets.
- **Spark:** It is an open-source software framework for processing large data sets.



# Parallel Computation

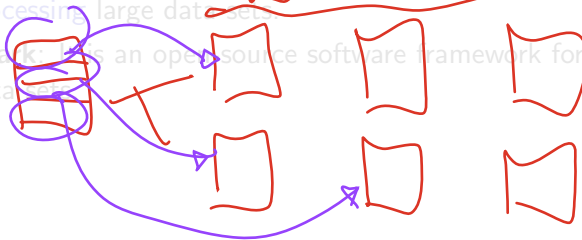
- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.



# Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a parallel, distributed algorithm on a **cluster**.

- **Hadoop:** *Kubernetes* is an open-source software framework for storing and processing large datasets.
- **Spark:** is an open-source software framework for processing large data sets.



# Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a parallel, distributed algorithm on a **cluster**.
- **Hadoop:** It is an open-source software framework for **storing and processing large data sets**.

Spark: It is an open-source software framework for processing large data sets.  
 Apache ← Not Beans  
               ← RDDs  
               ← Kafka



# Parallel Computation

- **Parallel Computation:** It is the process of executing **multiple tasks simultaneously**.
- **Distributed Computation:** It is the process of executing **multiple tasks on multiple machines**.
- **MapReduce:** It is a programming model for processing **large data sets** with a parallel, distributed algorithm on a **cluster**.
- **Hadoop:** It is an open-source software framework for **storing and processing** large data sets.
- **Spark:** It is an open-source software framework for **processing** large data sets.

← *real-time* {  
    Uber  
    FB  
    X  
    Netflix



# Outline

- 1 Object-Oriented Design
- 2 Good Practices in OO Design
- 3 Classes, Modules, and Packages
- 4 Resources Management



# Thanks!

## Questions?



Repo:

*[github.com/engandres/ud-public/tree/main/courses/  
advanced-programming](https://github.com/engandres/ud-public/tree/main/courses/advanced-programming)*



UNIVERSIDAD DISTRITAL  
FRANCISCO JOSÉ DE BALBUENA