In partial fulfillment of the requirements for the

CS 123 - Object Oriented Programming

**“Four Principles of Object-Oriented Programming”**

**Presented to:**

Dr. Unife O. Cagas

Professor V

**Presented by:**

Anthony Macula

BSCS-2A2

**"Sports"**

Project Title

**Project Description**

A Python project called the Sports Information System is created to manage sports-related data and activities more efficiently. Using the fundamentals of object-oriented programming (OOP), this system provides a solid framework to manage different sports and their unique requirements. The Sports Information System, which has as its foundation inheritance, polymorphism, encapsulation, and abstraction, enables seamless interaction with a broad range of sports, from basketball and tennis to football and basketball. The system's classes each contain unique properties and features that encourage code modularity, extensibility, and clarity. The Sports Information System enables users to effectively handle sports-related data, such as the number of players in a football game, a basketball player's league affiliation, or the style of tennis court, by utilizing these OOP concepts.

**Objectives:**

1. This code's main goal is to make it easier to represent different sports in a modular and extensible manner. The code makes it simple to introduce additional sports with their own special characteristics in the future by building a base class called Sport and subclasses for each particular sport.
2. To provide special information about each sport, subclasses must implement the abstract methods defined in the base class Sport, which is defined by the code. This is an example of abstraction in action. Data integrity is ensured and restricted access to class characteristics is made possible by encapsulating attributes within each class.
3. The code allows common functionality to be written in the base class Sport, which can be inherited by subclasses, through inheritance and method overriding, which encourages reusability. This makes the codebase easier to manage and lessens code duplication.
4. The display\_info() function, which is defined in the base class Sport and implemented differently in each subclass, serves as an example of polymorphism in the code. This enables various behaviors from the same method call (display\_info()) based on the kind of sport object being worked upon.
5. Another goal is to offer a way to present details about every sporting item in an organized way. The display\_info() method accomplishes this by printing information about the sport, including its name, player count, and any particular characteristics that make it stand out from the others.

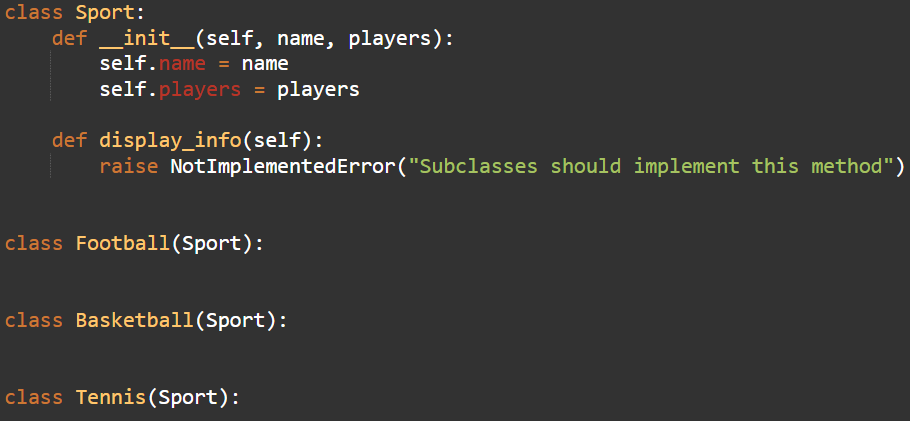
**Importance and Contribution of the Project**

This project uses Python object-oriented programming to create a hierarchy of sports. It describes a base class called "sport," with subclasses like "football," "basketball," and "tennis," each of which stands for a particular sport with special characteristics. Because common functionality is encapsulated within the base class and adding other sports is simple, the code encourages modularity, extensibility, and reusability. It is appropriate for teaching, organizing code, and creating sports administration apps since it offers a systematic way to display details about every sport object. In general, the project helps the developer community learn OOP ideas, organize code, customize it, and stimulate community participation.

**Four Principles of Object-Oriented Programming with code**

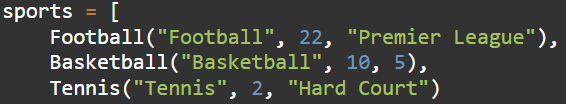
**Class:**

The class is the Sport class, serving as the base class, and its subclasses (Football, Basketball, Tennis). Each class encapsulates attributes and methods related to a specific sport, with Sport providing common functionality like the display\_info() method, which is intended to be implemented by its subclasses. Subclasses inherit attributes and methods from the Sport class, allowing for code reuse, while also enabling polymorphism, as each subclass can provide its own implementation of shared methods.



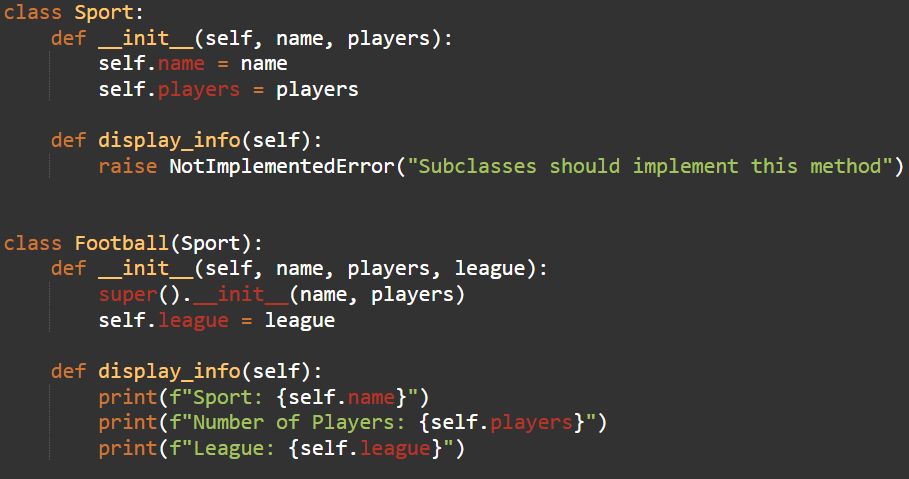
**Object:**

In the code, objects are instances of classes, each representing a specific sport. Three objects are instantiated: a Football object for Premier League football, a Basketball object for basketball, and a Tennis object for tennis. These objects are created using constructors Football(...), Basketball(...), and Tennis(...) respectively, with each constructor taking parameters relevant to the sport it represents. These objects encapsulate data about the sports, such as the number of players and league/court type, and are stored in a list named sports for further processing.



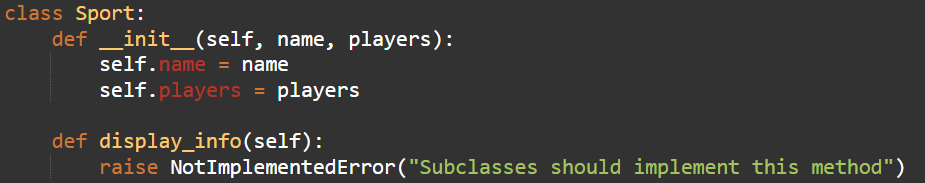
**Inheritance:**

Inheritance is demonstrated through the relationship between the Sport class and its subclasses (Football, Basketball, Tennis). The subclasses inherit attributes and methods from the Sport class, which helps in reusing code and organizing it hierarchically. For example, Football, Basketball, and Tennis inherit the display\_info() method from the Sport class.



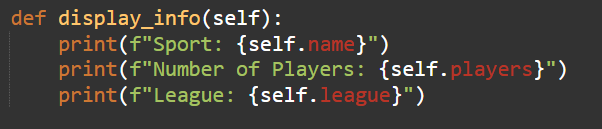
**Encapsulation:**

Encapsulation refers to the bundling of data (attributes) and methods that operate on the data into a single unit, often referred to as a class. In your code, each class encapsulates data related to a particular sport (name, players, etc.) along with methods that operate on that data. For instance, the Sport class encapsulates the name and players attributes along with the display\_info() method.



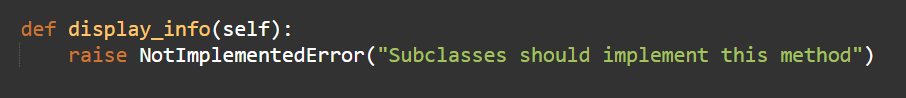
**Polymorphism:**

Polymorphism allows objects of different classes to be treated as objects of a common superclass. In your code, polymorphism is demonstrated when you iterate over a list of Sport objects and call the display\_info method on each object. Despite the objects being of different subclasses (Football, Basketball, Tennis), they can all be treated uniformly as Sport objects, thanks to the shared interface defined by the display\_info method.



**Abstraction:**

Abstraction refers to hiding the complex implementation details and showing only the necessary features of an object. In your code, the Sport class acts as an abstraction where it defines a generic interface (display\_info method) that all subclasses must implement. This allows users to interact with sports objects uniformly without worrying about their internal implementations**.**



**Hardware and Software Used**

**Hardware:**

1. Laptop
2. Cellphone

**Software:**

1. Online GDB

**Output:**

****

The code produces an output that shows information on three distinct sports: basketball, tennis, and football. Every sport is represented by an item from the appropriate class (tennis, basketball, football). The output for football contains the league name ("Premier League"), the sport's name, and the number of players (22). It contains the name of the sport, the quantity of players (10), and the size of the team (5) for basketball. Finally, for tennis, it contains the type of court ("Hard Court"), the sport's name, and the number of participants (2).

**Code Documentation**

class Sport:

# Define a parent class named Sport.

def \_\_init\_\_(self, name, players):

# Define an initialization method with parameters name and players.

self.name = name

# Assign the value of the name parameter to the instance variable name.

self.players = players

# Assign the value of the players parameter to the instance variable players.

def display\_info(self):

# Define a method named display\_info.

raise NotImplementedError("Subclasses should implement this method")

# Raise a NotImplementedError with a message.

class Football(Sport):

# Define a subclass named Football inheriting from Sport.

def \_\_init\_\_(self, name, players, league):

# Define an initialization method with parameters name, players, and league.

super().\_\_init\_\_(name, players)

# Call the superclass's initialization method.

self.league = league

# Assign the value of the league parameter to the instance variable league.

def display\_info(self):

# Define a method named display\_info.

print(f"Sport: {self.name}")

# Print the name of the sport.

print(f"Number of Players: {self.players}")

# Print the number of players.

print(f"League: {self.league}")

# Print the league.

class Basketball(Sport):

# Define a subclass named Basketball inheriting from Sport.

def \_\_init\_\_(self, name, players, team\_size):

# Define an initialization method with parameters name, players, and team\_size.

super().\_\_init\_\_(name, players)

# Call the superclass's initialization method.

self.team\_size = team\_size

# Assign the value of the team\_size parameter to the instance variable

team\_size.

def display\_info(self):

# Define a method named display\_info.

print(f"Sport: {self.name}")

# Print the name of the sport.

print(f"Number of Players: {self.players}")

# Print the number of players.

print(f"Team Size: {self.team\_size}")

# Print the team size.

class Tennis(Sport):

# Define a subclass named Tennis inheriting from Sport.

def \_\_init\_\_(self, name, players, court\_type):

# Define an initialization method with parameters name, players, and court\_type.

super().\_\_init\_\_(name, players)

# Call the superclass's initialization method.

self.court\_type = court\_type

# Assign the value of the court\_type parameter to the instance variable court\_type.

def display\_info(self):

# Define a method named display\_info.

print(f"Sport: {self.name}")

# Print the name of the sport.

print(f"Number of Players: {self.players}")

# Print the number of players.

print(f"Court Type: {self.court\_type}")

# Print the court type.

sports = [

# Create a list named sports.

Football("Football", 22, "Premier League"),

# Create a Football object with specified attributes and add it to the list.

Basketball("Basketball", 10, 5),

# Create a Basketball object with specified attributes and add it to the list.

Tennis("Tennis", 2, "Hard Court")

# Create a Tennis object with specified attributes and add it to the list.

print("Details of Sports:")

# Print a header for the displayed information.

print("===================")

# Print a separator.

for sport in sports:

# Iterate through the sports list.

sport.display\_info()

# Call the display\_info method for each sport object.

print("-------------------")

# Print a separator after displaying each sport's information.

**User Guide:**

1. Understanding the Structure:

• The script defines a base class Sport and three subclasses Football, Basketball, and Tennis, each representing a specific sport.

• Each subclass has its own attributes and implements a method display\_info() to print information about the sport.

1. Creating Objects:

• Instantiate the subclasses with specific parameters:

• For Football, provide the name of the sport, number of players, and league.

• For Basketball, provide the name, number of players, and team size.

• For Tennis, provide the name, number of players, and court type.

1. Displaying Information:

• After creating sports objects, call the display\_info() method on each object to display their information.

• Iterate over the sports objects and print their details.

1. Example Output:

• After running the script, you'll see information displayed for each sport, including its name, number of players, and additional attributes like league or court type.

1. Extending the Code:

• You can extend the script by adding more subclasses for other sports, following the same structure.

• Ensure to implement the display\_info() method for each new subclass to maintain consistency.

**References:**

"Python Tutorial" - W3Schools. [Online]. Available:

<https://www.w3schools.com/python/>

"ChatGPT - Conversational AI Chatbots for Developers" - ChatGPT. [Online]. Available:

<https://chatgpt.com/c/f3a3ff57-ac9d-4cfe-9d8b-17e04ff7fd0b>