

# OOP Explained through Checkers

Presented by:  
Asma Hammedi  
Godwin Chigozie  
Sachin Kumar  
Yusra Adeyeri



# Content

- 01 Introduction to checkers
- 02 What is Object-Oriented programming
- 03 Benefits of Using OOP
- 04 Coding Language Choice
- 05 Applying OOP to the Checkers
- 06 Conclusion

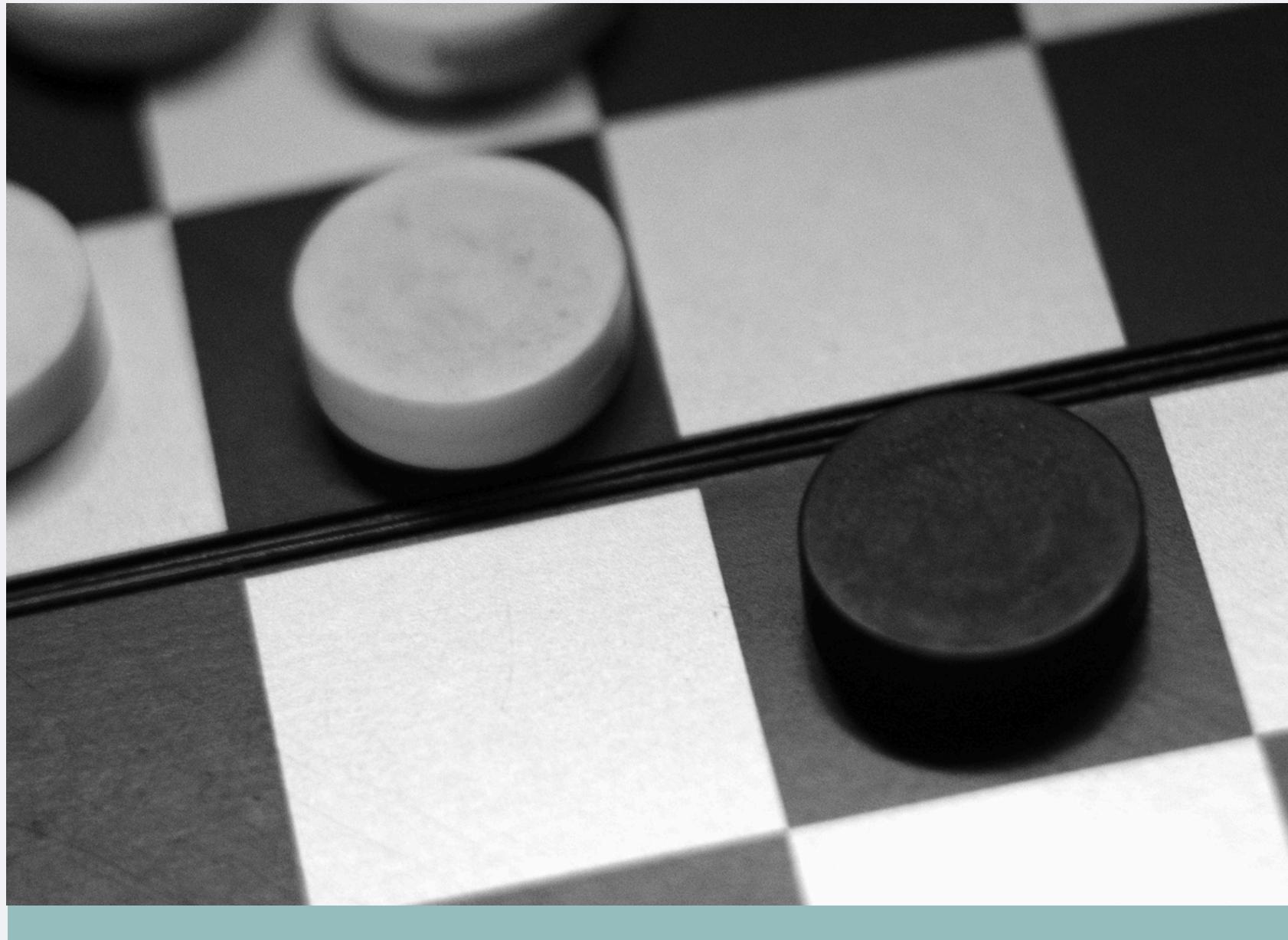
```
1 <div><input type="text" name="Nombre de Us" />
2 <input type="password" name="Contraseña" />
3 <input type="submit" name="Iniciar Sesión" />
4 <input type="button" name="Cancelar" />
5 </div>
6
7 <div>
8   <input type="text" name="Email" />
9   <input type="text" name="Contraseña" />
10  <input type="submit" name="Iniciar Sesión" />
11  <input type="button" name="Cancelar" />
12 </div>
13
14 <div>
15   <input type="text" name="Nombre" />
16   <input type="password" name="Contraseña" />
17   <input type="submit" name="Iniciar Sesión" />
18   <input type="button" name="Cancelar" />
19 </div>
20
21 <div>
22   <input type="text" name="Email" />
23   <input type="text" name="Contraseña" />
24   <input type="submit" name="Iniciar Sesión" />
25   <input type="button" name="Cancelar" />
26 </div>
27
28 <div>
29   <input type="text" name="Nombre" />
30   <input type="password" name="Contraseña" />
31   <input type="submit" name="Iniciar Sesión" />
32   <input type="button" name="Cancelar" />
33 </div>
34
35 </div>
```

Introduction to

# CHECKERS

# Introduction to Checkers

- Checkers is a two-player board game
- played on a checkered board with 64 squares.
- Players take turns moving their pieces diagonally forward.
- The objective is to capture all of the opponent's pieces by jumping over them or to block them so they cannot move.



# Pieces/Board

## Board

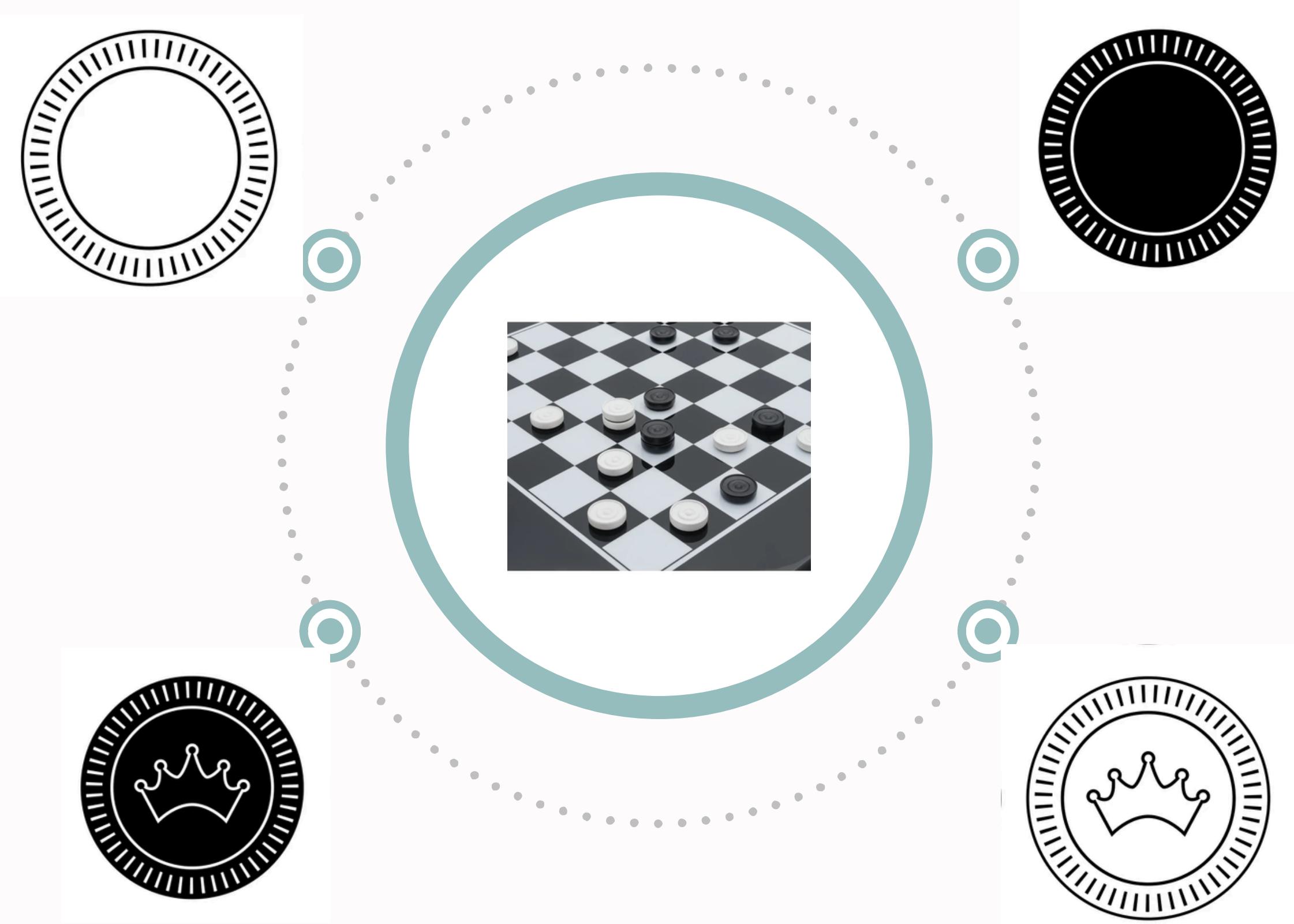
- Standard 8x8 board
- Only dark squares are used for playing
- Each player starts with 12 pieces
- Pieces are placed on the first 3 rows of each side

## Pieces

- All pieces start as "Men"
- Men can move diagonally forward
- Men can capture by jumping over opponent pieces diagonally

## King Promotion

- A piece becomes a King when it reaches the last row
- Men can move diagonally forward
- Kings can move and capture diagonally both forward and backward



# WHAT IS OOP ?

```
# now we can access the contents of the JSON file
12 if "title" in theJSON["features"]:
13     print(theJSON["features"])
14
15 # output the number of events, after filtering
16 count = theJSON["metadata"]["eventCount"]
17 print(str(count) + " events recorded")
18
19 # for each event, print the place where it happened
20 for i in theJSON["features"]:
21     print(i["properties"]["place"])
22     print("\n")
23
24 # print the events that only have a magnitude greater than or equal to 4.0
25 for i in theJSON["features"]:
26     if i["properties"]["mag"] >= 4.0:
27         print("A " + str(i["properties"]["mag"]) + " magnitude earthquake occurred at " + i["properties"]["place"])
28         print("\n")
29
30 # print out the events where at least 1 person reported feeling them
31 for i in theJSON["features"]:
32     if i["properties"]["felt"] != null:
33         print("Events where at least 1 person reported feeling them:")
34         print(i["properties"]["felt"])
35         print("\n")
```

# Object Oriented Programming

Object-Oriented Programming is a programming paradigm where software is organized around objects, which combine data (attributes) and behavior (methods).

## Characteristics of OOP

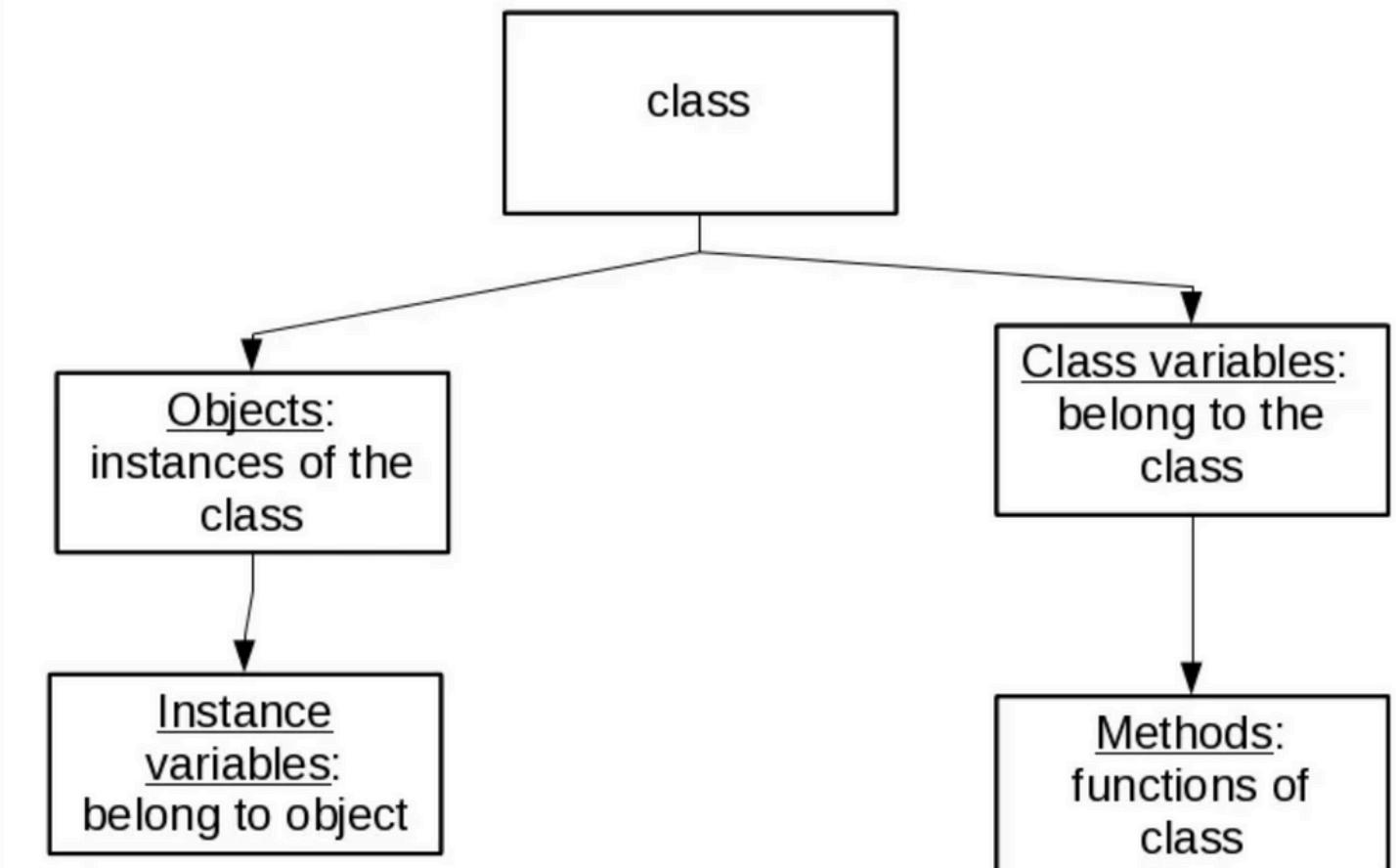
 **Encapsulation:** Data security and access control

 **Abstraction :** Showing only essential features and hiding the rest

 **Inheritance:** The capability to create new classes derived from existing ones

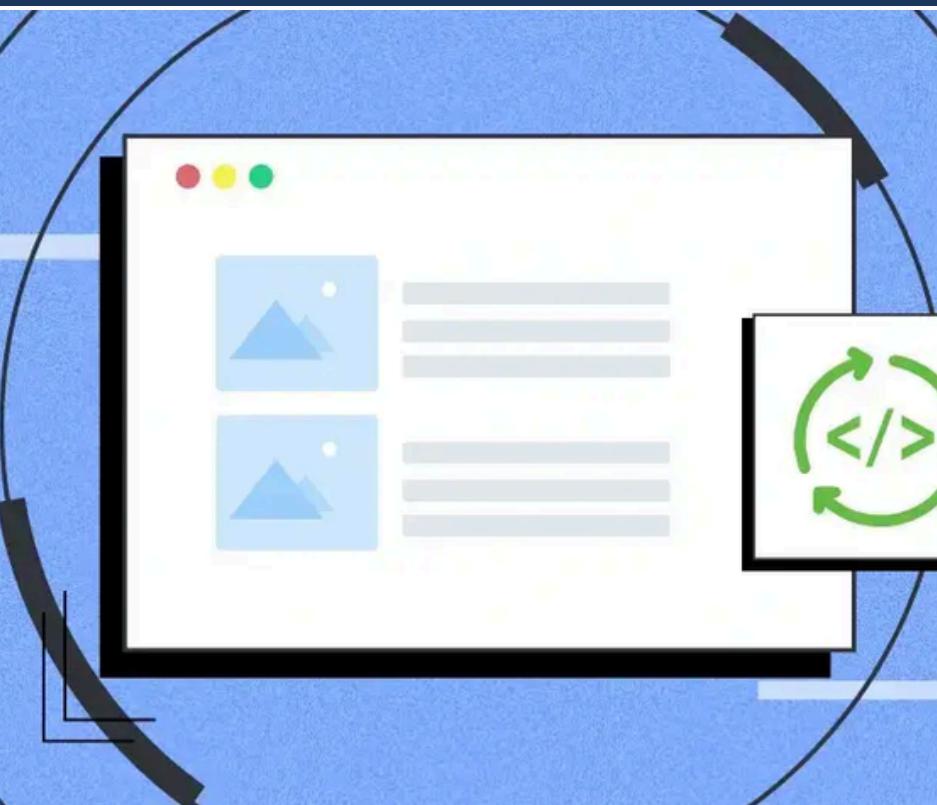
 **Polymorphism :** Same method name, different behavior depending on the object.

### Object Oriented Programming



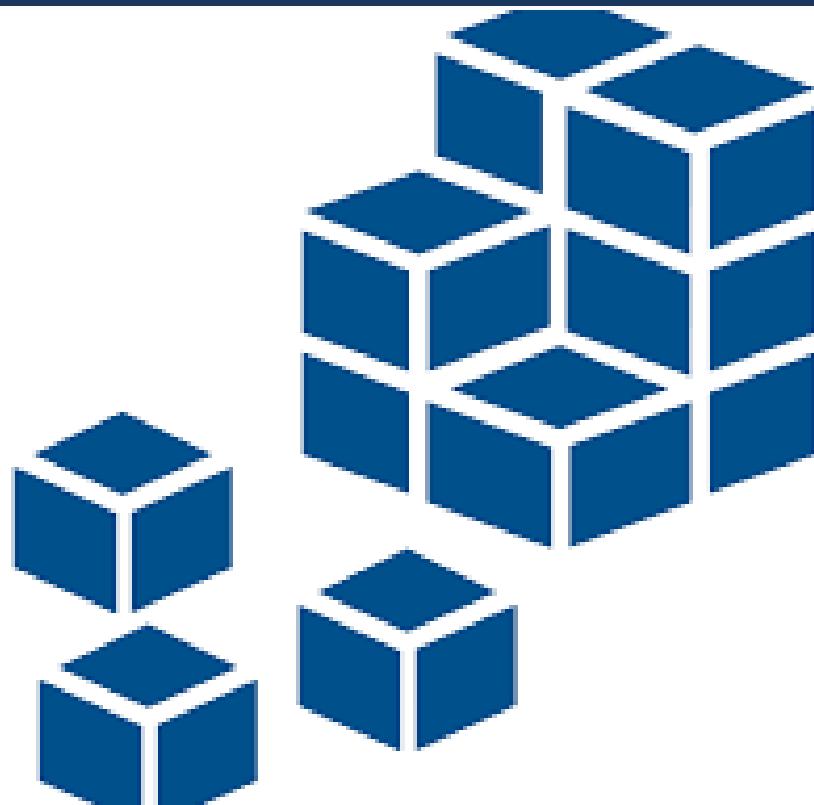
# Benefits of using OOP

## Code Reusability



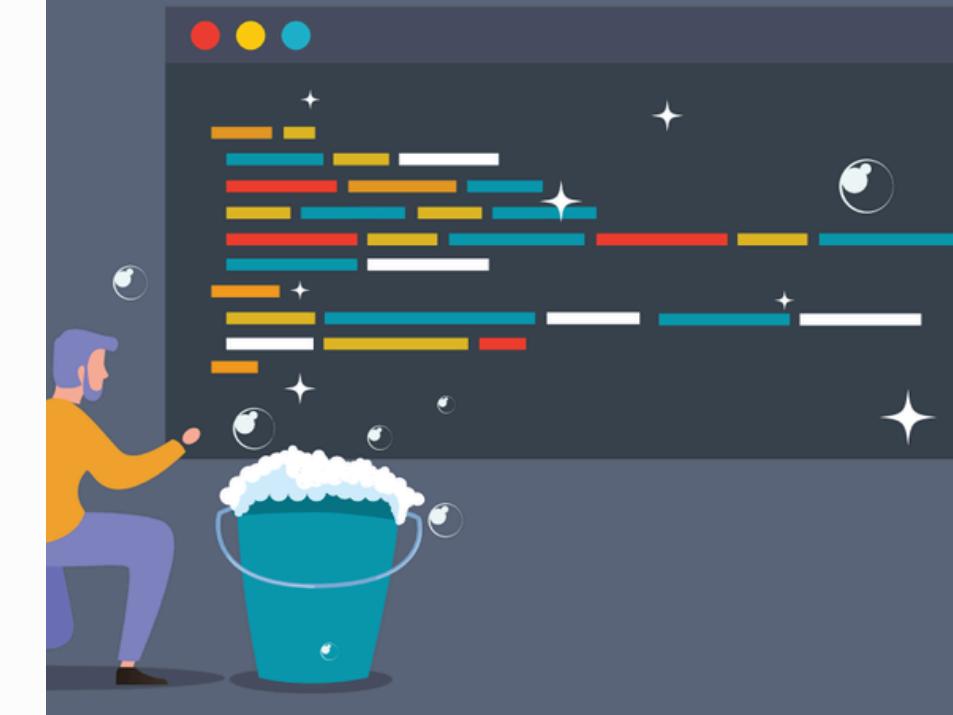
*Reuse existing classes through inheritance to avoid rewriting code.*

## Modularity



*Divide the program into independent objects, making it easier to manage and update.*

## Easier Maintenance



*Encapsulation keeps code organized, so fixing bugs or adding features is simpler.*

# Benefits of using OOP

## Scalability



*Easy to add new features or components without changing existing code too much.*

## Data Protection



*Keeps internal data safe by restricting direct access, improving security and reliability.*

## Collaboration Friendly



*Modular structure helps teams work on different parts of the project without conflicts.*

# CODING

## Language Choice

```
# now we can access the contents of the JSON file
if "title" in theJSON["features"]:
    print(theJSON["metadata"])

# output the number of events, after the first two lines
count = theJSON["metadata"]["count"]
print(str(count) + " events recorded")

# for each event, print the place name
for i in theJSON["features"]:
    print(i["properties"]["placeName"])
    print("\n")

# print the events that only have a magnitude greater than or equal to 2.0
for i in theJSON["features"]:
    if i["properties"]["mag"] >= 4.0:
        print("A " + str(i["properties"]["mag"]) + " magnitude earthquake occurred at " + i["properties"]["placeName"])
        print("\n")

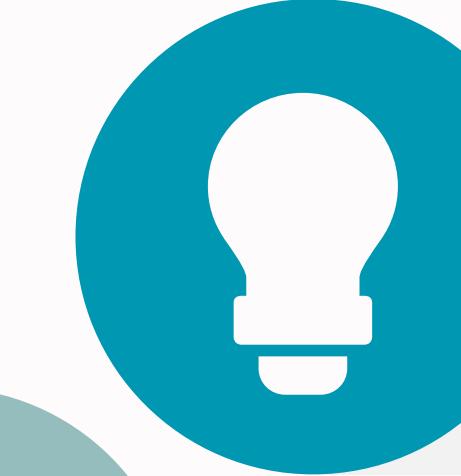
# print out the events where at least 1 person reported feeling it
for i in theJSON["features"]:
    if i["properties"]["felt"] != null:
        print("Events where at least 1 person reported feeling it")
        print(i["properties"]["felt"])
        print("\n")
```

# Python

- High-level, easy-to-read programming language
- Great for beginners and widely used in real-world applications
- Supports object-oriented programming (OOP)



High-level



Large Community Support



Rich Standard Library



Multi-Paradigm

```
7 attr_reader :control
8
9 # An Experiment.
10 attr_reader :experiment
11
12 # An Array of observations which didn't match the control, but were ignored.
13 attr_reader :ignored
14
15 # An Array of observations which didn't match the control.
16 attr_reader :mismatched
17
18 # An Array of Observations in execution order.
19 attr_reader :observations
20
21 # Internal: Create a new result.
22 #
23 # experiment - the Experiment
24 # observations: - an Array of Observations
25 # control: - the Control
26 #
27 def initialize(experiment, observations, control)
28   @experiment = experiment
29   @observations = observations
30   @control = control
31 end
```

# PYGAME

Pygame is a Python library used to create 2D games and multimedia applications. It provides tools for handling graphics, sound, and user input with ease.



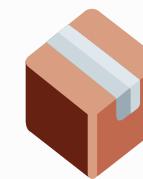
## 2D Graphics Support

Easily draw shapes, images, and game boards



## Input Handling

Detects keyboard and mouse actions (like clicks or movement)



## Simple & Lightweight

Easy to install and perfect for small game projects

# APPLYING OOP

To Checkers

# Class Overview



- Base class for Man and King
- Attributes: color, is\_king

## Methods:

- `__repr__()`
- `get_possible_moves()` → abstract (to be overridden)

# Class Overview



- Represents a normal piece

Overrides:

- `get_possible_moves()` – moves diagonally forward
- `__repr__()`

# Class Overview



- Represents a promoted piece

Overrides:

- `get_possible_moves()` – moves diagonally in all directions
- `__repr__()`

# Class Overview

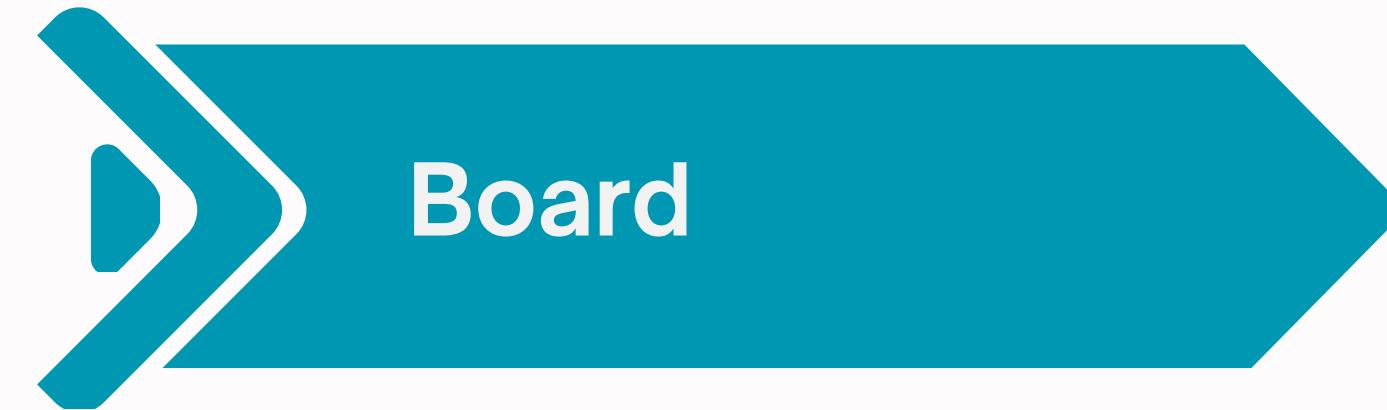


- Represents a player and their pieces
- Attributes: color, pieces

## Methods:

- add\_piece()
- remove\_piece()
- has\_pieces()
- get\_all\_possible\_moves()

# Class Overview

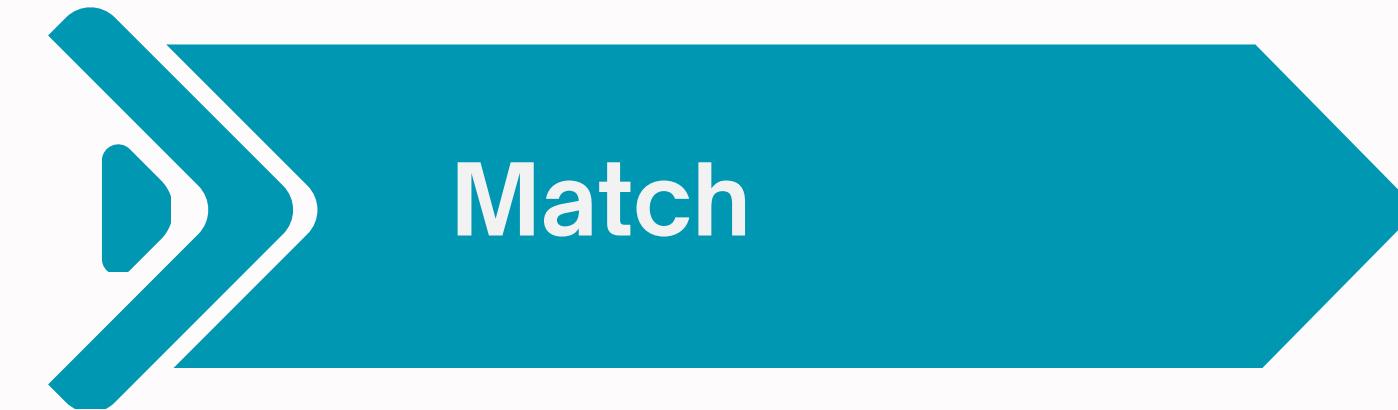


- Manages the 8x8 grid and piece logic
- Attributes: board (2D array)

## Methods:

- initialize\_board()
- get\_piece(), set\_piece(), remove\_piece()
- move\_piece()
- is\_valid\_move()
- get\_all\_pieces()
- display(), copy()

# Class Overview



- Controls the full game logic
- Attributes: board, player1, player2, current\_player, game\_over, winner

## Methods:

- start\_game()
- make\_move()
- switch\_player(), check\_game\_over()
- get\_winner(), get\_possible\_moves\_for\_piece()
- is\_game\_over()

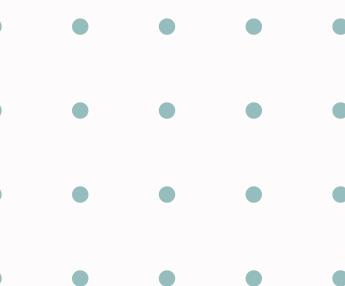
# Encapsulation

```
class Player:  
    def __init__(self, color):  
        self.color = color #data (attribute)  
        self.pieces = [] #data (attribute)  
  
    def __repr__(self):      # Method that manipulates internal data  
        return f'Player({self.color})'  
  
    def add_piece(self, piece):  
        self.pieces.append(piece)  
  
    def remove_piece(self, piece):  
        if piece in self.pieces:  
            self.pieces.remove(piece)  
  
    def get_pieces_count(self):  
        return len(self.pieces)  
  
    def has_pieces(self): # Method that gives access without exposing raw list  
        return len(self.pieces) > 0
```

→ Combining data & the functions that manipulate that data into a single entity.

Data is protected and accessed only through controlled methods.

✓ Encapsulation in action!



# Inheritance

```
# Base class for all pieces (Man and King)
class Piece:
    def __init__(self, color):
        self.color = color
        self.is_king = False
```

→ Man and King inherit from the base class Piece.

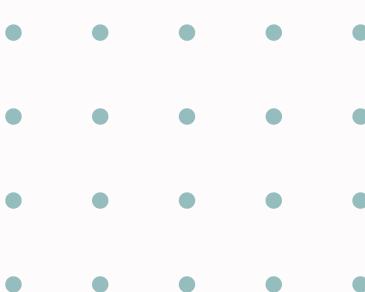
```
#Inheritance: Man class inherits from Piece
class Man(Piece):
    def __init__(self, color):
        super().__init__(color) # Man class inherits from Piece
```

Shared attributes like color and is\_king are reused from Piece.

```
#Inheritance: King class inherits from Piece
class King(Piece):
    def __init__(self, color):
        super().__init__(color)
        self.is_king = True # King-specific property set to True
```

Write once in Piece, reuse in both Man and King

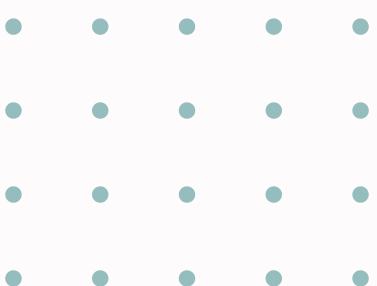
✓ Inheritance in action!



# Polymorphism

```
def get_possible_moves_for_piece(self, row, col):      #Polymorphism: behaves differently for Man and King
    piece = self.board.get_piece(row, col)
    if piece and piece.color == self.current_player.color:
        return piece.get_possible_moves(self.board.board, row, col)
    return []
```

- Python calls the correct version:  
Man.get\_possible\_moves()  
or King.get\_possible\_moves()
- ✓ Polymorphism in action!



# Abstraction

```
#abstraction:This method hides all the internal game logic  
if match.make_move(from_row, from_col, row, col):  
    selected_piece = None  
    possible_moves = []
```

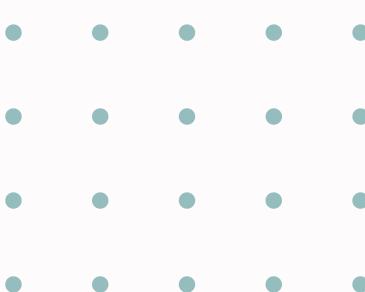
```
def make_move(self, from_row, from_col, to_row, to_col):  
    piece = self.board.get_piece(from_row, from_col)  
    if piece is None or piece.color != self.current_player.color:  
        return False  
  
    if self.board.move_piece(from_row, from_col, to_row, to_col):  
        self.update_players_pieces()  
        self.check_game_over()  
        if not self.game_over:  
            self.switch_player()  
        return True  
    return False
```

→ python calls :

match.make\_move(from\_row,  
from\_col, to\_row, to\_col)

Complex logic hidden behind one  
simple method.

✓ Abstraction in action!



# Comparison

## User Interaction

## Visual Feedback

## Real-Time Experience

*CLI*



Text commands



No visuals



Not real-time

*Rest API*



Programmatic  
only



No visuals



Not real-time

*Pygame GUI*



Mouse & visuals



Graphical



fully interactive

[PROBLEMS](#)[OUTPUT](#)[DEBUG CONSOLE](#)[TERMINAL](#)[PORTS](#)**== CHECKERS ==**

Welcome To Checkers!

**== COMMANDS ==**

1 - Move a piece

2 - Show piece details

3 - Show current board

4 - Leave the game

Help - Display this help

Move format: A1-H8 (ex: A1, B2, etc.)

**Player's turn: black**

	A	B	C	D	E	F	G	H	
8	□	●	□	●	□	●	□	●	8
7	●	□	●	□	●	□	●	□	7
6	□	●	□	●	□	●	□	●	6
5	■	□	■	□	■	□	■	□	5

8	□	●	□	●	□	●	□	●	8
7	●	□	●	□	●	□	●	□	7
6	□	●	□	●	□	●	□	●	6
5	■	□	■	□	■	□	■	□	5
4	□	■	□	■	□	■	□	■	4
3	○	□	○	□	○	□	○	□	3
2	□	○	□	○	□	○	□	○	2
1	○	□	○	□	○	□	○	□	1
	A	B	C	D	E	F	G	H	

What would you like to do?  
Your Choice:**CLI**

# REST API

```
← ⌂ ⌄ localhost:5000
Impression automatique □

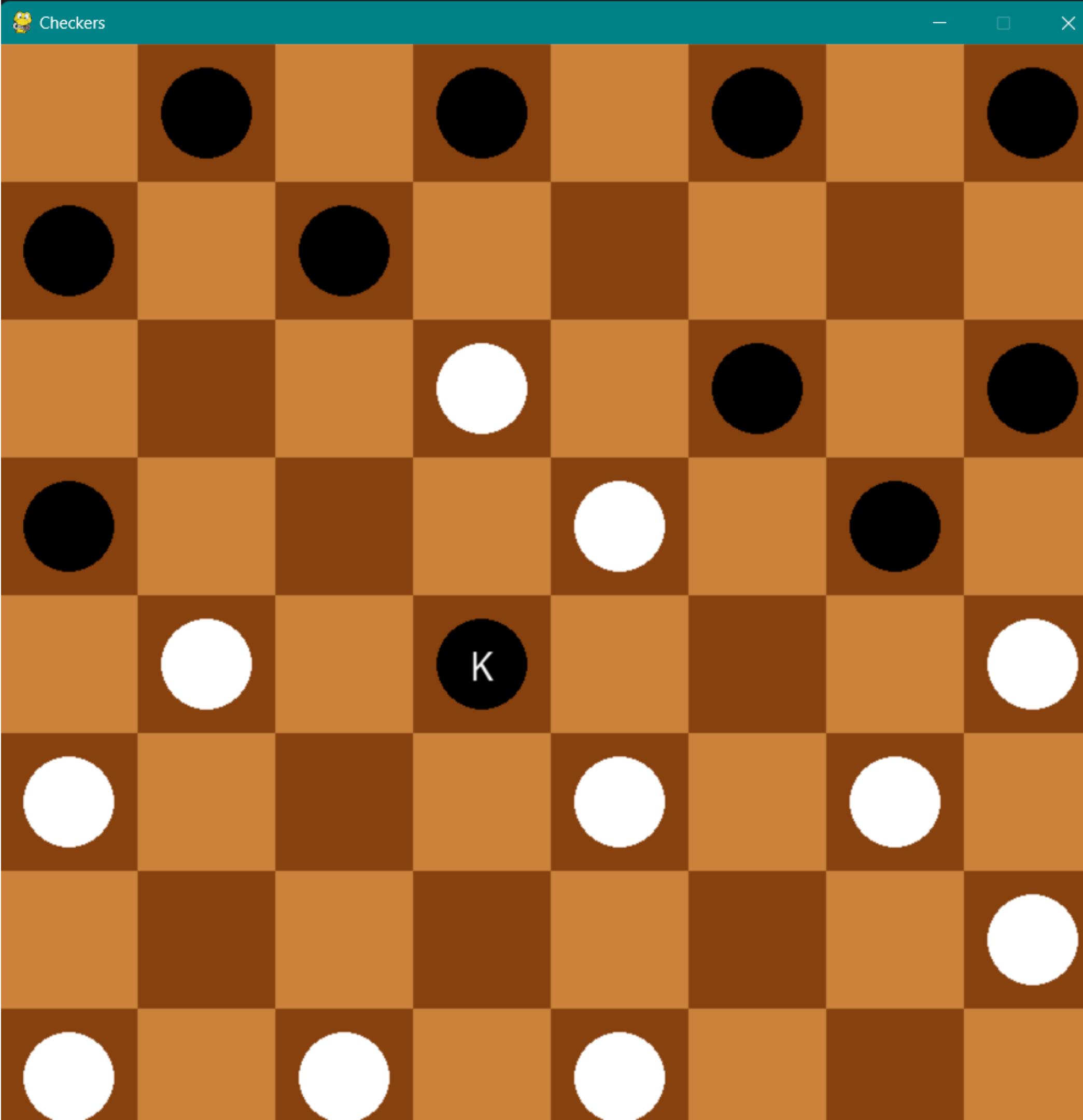
{
  "endpoints": {
    "GET /board": "Get current board state",
    "GET /info": "Get piece info (params: ?player=Black&piece=2A)",
    "POST /move": "Make a move (JSON: {player, from, to})",
    "POST /reset": "Reset the game"
  },
  "example_move": {
    "from": "2A",
    "player": "Black",
    "to": "3B"
  },
  "message": "Checkers Game API"
}
```

POST  Send

Query	Headers <small>2</small>	Auth	Body <small>1</small>	Tests	Pre Run	Response	Headers <small>6</small>	Cookies	Results	Docs	{ }	≡
<small>JSON</small>	<small>XML</small>	<small>Text</small>	<small>Form</small>	<small>Form-encode</small>	<small>GraphQL</small>	<small>Binary</small>	<small>1</small> { <small>2</small> "current_player": "white", <small>3</small> "game_over": false, <small>4</small> "message": "Move made: B6 -> A5", <small>5</small> "success": true, <small>6</small> "winner": null <small>7</small> }	<small>1</small> Content-Type: application/json <small>2</small> Host: localhost:5000 <small>3</small> Connection: keep-alive <small>4</small> Accept: */* <small>5</small> User-Agent: Postman/7.2.2 <small>6</small> Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c				
JSON Content						Format						
<pre>1 {   "player": "black",   "from": "B6",   "to": "A5" }</pre>												

Status: 200 OK    Size: 127 Bytes    Time: 7 ms

# Pygame



# CONCLUSION

```
# now we can access the contents of each event
12 if "title" in theJSON["features"]:
13     print(theJSON["metadata"]["title"])
14
15 # output the number of events, after filtering
16 count = theJSON["metadata"]["count"]
17 print(str(count) + " events recorded")
18
19 # for each event, print the place where it happened
20 for i in theJSON["features"]:
21     print(i["properties"]["place"])
22     print("-----\n")
23
24 # print the events that only have a magnitude greater than or equal to 4.0
25 for i in theJSON["features"]:
26     if i["properties"]["mag"] >= 4.0:
27         print(str(i["mag"]) + " " + i["properties"]["place"])
28         print("-----\n")
29
30 # print out the events where at least 1 person reported
31 # experiencing that were felt:
32 for i in theJSON["features"]:
33     if i["properties"]["felt"] >= 1:
34         print(i["properties"]["place"])
35         print("-----\n")
```

# Conclusion

This project demonstrates the flexibility of Object-Oriented Programming by implementing three different interfaces

Each interface serves a distinct purpose:

- CLI is lightweight and ideal for debugging or scripting.
- REST API enables integration with other systems or platforms.
- Pygame GUI offers the most engaging and intuitive experience for real-time users.

By abstracting the core game logic and separating it from the interface layer, the system becomes modular, testable, and easily extensible.

This multi-interface architecture showcases the power of encapsulation, abstraction, inheritance, and polymorphism — the key pillars of OOP.



# THANK YOU

```
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```