CENTENNIAL COLLEGE

# LAB #1

# Lab Overview

By completing this lab, students will be able to:

- Design **classes** using proper syntax
- Apply **encapsulation** with private data and public access
- Implement **inheritance** and reuse code effectively
- Demonstrate **polymorphism** using virtual functions
- Write and use **getters and setters**
- Understand **pass by value vs pass by reference**
- Use **macros and symbols** correctly
- Work with **constructors, destructors, and virtual destructors**
- Use **base class pointers** to derived objects
- Apply **const**
- Use **static class members**

## Follow the Steps below:

# TASK 1 — Core Game Objects & Encapsulation

**Step 1: Create GameObject (Base Class)**

Requirements:

- Private data members:
  - int id
  - std::string name
- Public:
  - Constructor
  - Virtual destructor
  - Getters and setters
  - Virtual function printInfo() const

**Purpose:**
**Establish a reusable base for all game entities.**

---

### Step 2: Create Player Class

Requirements:

- Inherits from GameObject
- Private data members:
  - int health
  - int score
- Public:
  - Constructor
  - Getters and setters with validation
  - Override printInfo()
  - Function takeDamage(int amount)
  - Function addScore(int amount)

---

### Step 3: Create Enemy Class

Requirements:

- Inherits from GameObject
  - Private data members:
    - int damage
  - Public:
    - Constructor
    - Getter and setter
    - Override printInfo()

---

### Step 4: Pass by Value vs Pass by Reference

- Create two free functions:
  - void healByValue(Player p, int amount);
  - void healByReference(Player& p, int amount);

**Students must:**

- **Invoke both functions**

**Questions:**

- **Observe which one changes the player's health**

**Heal by reference would change the players health because its actually referencing the initial player instead of a new one created by the function**

---

## Concepts Practiced

- Encapsulation
- Getters and setters
- Inheritance
- Pass by value vs reference

---

# TASK 2 — Power-Ups, Polymorphism & Macros

**Step 1: Create an Abstract PowerUp Class**

Requirements:

- Protected member:
  - std::string powerName
  - Public:
  - Virtual destructor
- Pure virtual function:
  - virtual void apply(Player& p) = 0;

---

**Step 2: Create Concrete Power-Ups**

Create at least two classes:

Examples:

- HealthBoost
- ScoreBoost
- Requirements:
  - Inherit from PowerUp
  - Override apply(Player&)

- Use macros to control values

---

**Step 3: Add Macros & Symbols**

- At the top of the file:
  - #define MAX_HEALTH 100
  - #define HEALTH_BOOST 25
  - #define SCORE_BOOST 50
- Use these macros inside your power-ups.

**Questions:**

- **Macro replacement happens before compilation**

| **All macro names "define" their values as text before complier runs through the code, therefore using those values.** |
|---|

- **Why const int is often safer than macros**

| **Const int is safer because its type checked and stays within its scope.** |
|---|

---

**Step 4: Polymorphism in Action**

- In main():
  - Create a Player
  - Store multiple PowerUp* objects in an array or std::vector
  - Apply each power-up using a loop
  - Show that the correct apply() function runs at runtime

---

**Step 5: Static Members**

- Add to PowerUp:
  - A static int totalPowerUpsUsed
  - Increment it every time apply() is called
  - Display the total at the end of the program

---

## Concepts Practiced

- Polymorphism
- Virtual functions
- Base class pointers
- Static members
- Macros & symbols

---

## Sample Output

*Player: Alex | Health: 75 | Score: 100*

*Enemy: Goblin | Damage: 10*

*Applying HealthBoost...*

*Applying ScoreBoost...*

*Player: Alex | Health: 100 | Score: 150*

*Total power-ups used: 2*

---

# Deliverables

---

- **The complete app in VS format.**
- **Screenshot after the completion of every Task**
  - **Code**
  - **Outpu**
- **Answer the questions of the lab and submit the lab document.**

SCREENSHOTS

TASK 1
OUPUT:

Microsoft Visual Studio Debug Console

```
GameObject Constructor Invoked
Player Constructor Invoked
GameObject Constructor Invoked
Enemy Constructor Invoked

Player ID: 10, Name: Adventurer, Health: 75, score: 200
Enemy ID: 5, Name: Dragon, Damage: 20

Applying HealthBoost...
Applying ScoreBoost...

Player ID: 10, Name: Adventurer, Health: 100, score: 250
Total power-ups used: 2

GameObject Destructor Invoked
GameObject Destructor Invoked
```

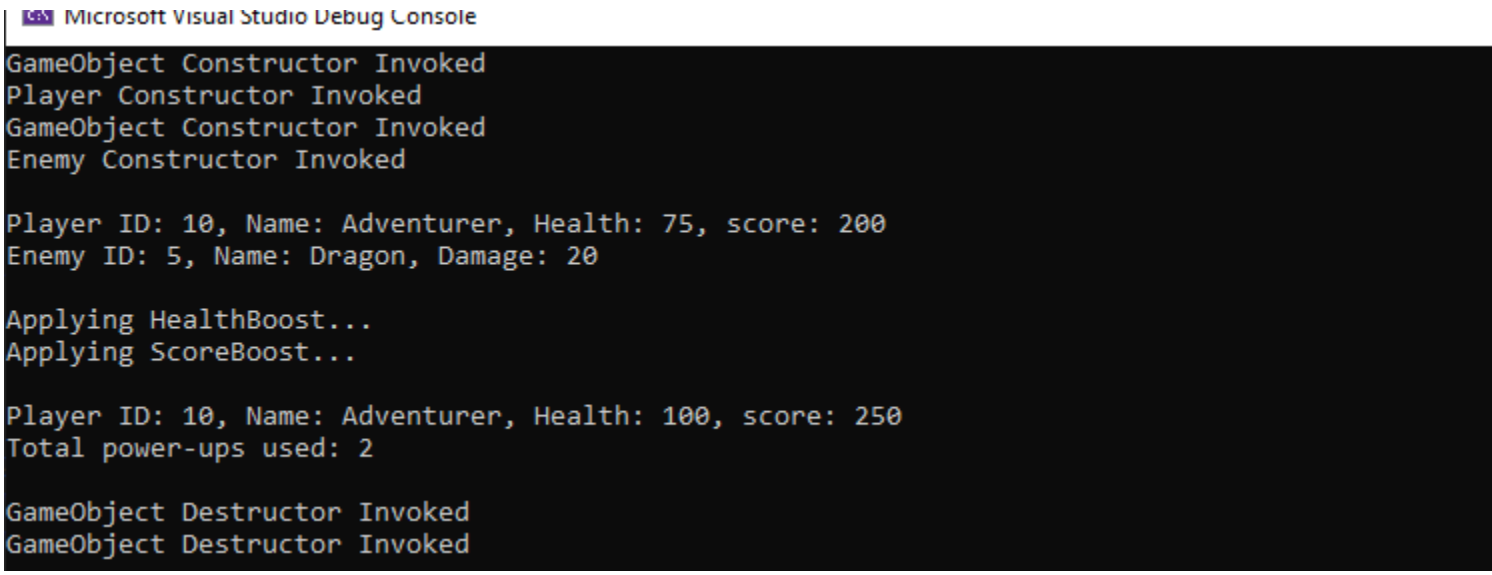
CODE:

```
public:
    //constructor
    GameObject(int id, const string& name) : id(id), name(name) {
        cout << "GameObject Constructor Invoked" << endl; //game object constructor
    }

    //virtual destructor
    virtual ~GameObject() {
        cout << "GameObject Destructor Invoked" << endl; //game object Destructor
    }

    //getter
    int getId() {
        return id;
    }


    string getName() {
        return name;
    }

    // setters
    void setId(int i) {
        id = i;
    }

    void setName(string n) {
        name = n;
    }

    //virtual function printinfo() const
    virtual void printInfo()  {
        cout << "GameObject: ID: " << id << ", Name: " << name << endl;
    }
};

//PLAYER ------------------------------------------------------------------

class Player : public GameObject {
private:
    int health;
    int score;

public:
    Player(int id, const string& name, int health, int score) : GameObject(id, name), health(health), score(score) {
        cout << "Player Constructor Invoked" << endl; //player constructor
    }

    //GETTER
    int getHealth() {
        return health;
    }

    int getScore() {
        return score;
    }

    //SETTER
    void setHealth(int h) {
        health = h;
    }
    void setScore(int s) {
        score = s;
    }

    //OVERRIDE INFO
    void printInfo() override {
        cout << "Player ID: " << getId() << ", Name: " << getName() << ", Health: " << health << ", score: " << score << endl;
    }

    //FUNCTIONS

    void takeDamage(int damage) {
        health -= damage;
    }

    void addScore(int s) {
        score += s;
    }
```

```cpp
//ENEMY ------------------------------------------------------------------------
class Enemy : public GameObject {
private:
    int damage;

public:
    Enemy(int id,  const string& name, int damage) : GameObject(id, name), damage(damage) {
        cout << "Enemy Constructor Invoked" << endl; //Enemy constructor
    }

    //GETTER
    int getDamage() {
        return damage;
    }

    //SETTER
    void setDamage(int d) {
        damage = d;
    }

    //OVERRIDE INFO
    void printInfo() override {
        cout << "Enemy ID: " << getId() << ", Name: " << getName() << ", Damage: " << damage << endl;
    }

};



//FREE FUNCTIONS ---------------------------------------------------------------
void healByValue(Player p, int amount) {
    p.setHealth(p.getHealth() + amount);
};

void healByReference(Player& p, int amount) {
    p.setHealth(p.getHealth() + amount);
};
```

```cpp
//MAIN -------------------------------------------------------------------------

int main() {
    Player player(10, "Adventurer", 75, 200);
    Enemy enemy(5, "Dragon", 20);

    GameObject* objectPlayer = &player;
    GameObject* objectEnemy = &enemy;

    cout << endl;

    objectPlayer->printInfo();
    objectEnemy->printInfo();

    cout << endl;

    return 0;
}
```

TASK 2
OUTPUT:




ADDITIONAL CODE ADDED:

```cpp
//POWERUP ------------------------------------------------------------------------
class PowerUp {
    protected:
    std::string powerName;

    public:
    static int totalPowerUpsUsed;

    //virtual destructor
    virtual ~PowerUp() {
        cout << "PowerUp Destructor Invoked" << endl; //PowerUp Destructor
    }

    virtual void apply(Player& p) = 0;
};

class HealthBoost : public PowerUp {
    public:
    void apply(Player& p) override {
        int health = HEALTH_BOOST;
        int maxHealth = MAX_HEALTH;
        healByReference(p, health);

        cout << "Applying HealthBoost..." << endl;

        if (p.getHealth() > maxHealth) {
            p.setHealth(maxHealth);
        }

        totalPowerUpsUsed++;
    }
};

class ScoreBoost : public PowerUp {
    public:
    void apply(Player& p) override {
        int score = SCORE_BOOST;
        cout << "Applying ScoreBoost..." << endl;
        p.addScore(score);

        totalPowerUpsUsed++;
    }
};

int PowerUp::totalPowerUpsUsed = 0;
```

```cpp
//MAIN ----------------------------------------------------------------------

int main() {
    Player player(10, "Adventurer", 75, 200);
    Enemy enemy(5, "Dragon", 20);

    GameObject* objectPlayer = &player;
    GameObject* objectEnemy = &enemy;

    cout << endl;

    objectPlayer->printInfo();
    objectEnemy->printInfo();

    cout << endl;

    //Applying Power Ups Loop
    PowerUp* powerUpArray[2];
    powerUpArray[0] = new HealthBoost();
    powerUpArray[1] = new ScoreBoost();

    for (int i = 0; i < 2; i++) {
        powerUpArray[i]->apply(player);
    }

    cout << endl;

    objectPlayer->printInfo();
    cout << "Total power-ups used: " << PowerUp::totalPowerUpsUsed << endl;

    cout << endl;

    return 0;
}
```