جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني

DEPARTMENT OF CYBER SECURITY

SUBJECT:

OBJECT ORIENTED PROGRAMMING (OOP)

CLASS:

SECOND

LECTURER:

DR. ABDULKADHEM A. ABDULKADHEM

LECTURE(10):

Inheritance in OOP (part 2)

Objective of the Lecture:

In this lecture, we will cover the five primary types of inheritance in object-oriented programming (OOP), explain their characteristics, and provide examples for each type.

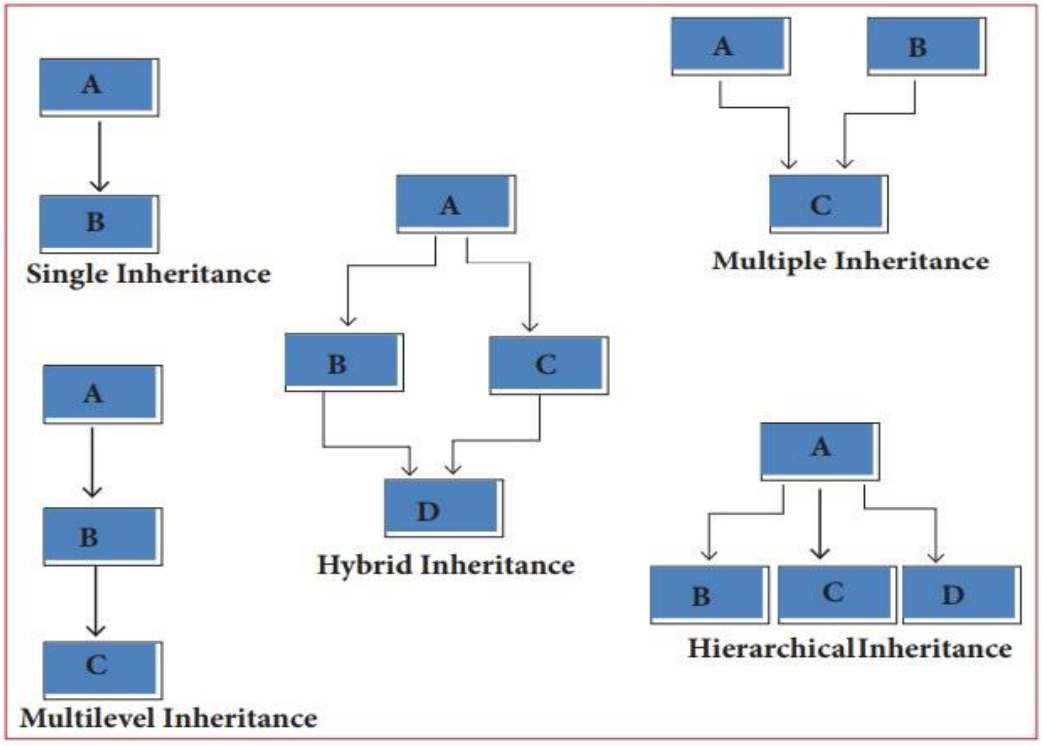
What is Inheritance?

Inheritance is a fundamental concept in OOP, allowing a class (derived class) to acquire the properties and methods of another class (base class). It enables code reuse, extends functionality, and supports a hierarchical classification.

Inheritance allows us to create new classes based on existing ones, enabling us to:

- **Reuse** code.
- **Extend** functionality.
- Establish a **hierarchical relationship** between classes.

There are different types of inheritance, Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance. The following figure represents the different types of inheritance.



1. Single Inheritance

- **Definition**: In single inheritance, a derived class inherits from a single base class. It allows the derived class to inherit the properties and methods of one parent class.
- **Characteristics**:
  - One base class.
  - One derived class.
  - Simple and straightforward inheritance.

Example of Single Inheritance:

```
#include <iostream>
using namespace std;
// Base class representing a Person
class Person {
public:
    void displayInfo() {
        cout << "This is a Person class" << endl;
    }
};
// Derived class representing a Student, inheriting from Person
class Student : public Person {
public:
    void displayStudentInfo() {
        cout << "This is a Student class" << endl;
    }
};

int main() {
    Student studentObj;
    studentObj.displayInfo();             // Inherited from Person class
    studentObj.displayStudentInfo();      // Function in Student class
    return 0;
}
```

2. Multiple Inheritance

- **Definition**: Multiple inheritance occurs when a derived class inherits from **more than one** base class. The derived class inherits attributes and methods from all the base classes.
- **Characteristics**:
  - One derived class inherits from multiple base classes.
  - Can result in ambiguity if there are overlapping member functions or attributes in the base classes.

Example of Multiple Inheritance:

```
#include <iostream>
using namespace std;
// Base class representing a Person
class Person {
```

```
public:
    string name;
    int age;

    void setInfo(string n, int a) {
        name = n;
        age = a;
    }
    void displayPersonInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};
// Base class representing Academic information
class Academic {
public:
    string Dept;
    float GPA;

    void setAcademicInfo(string m, float g) {
        Dept = m;
        GPA = g;
    }
    void displayAcademicInfo() {
        cout << "Dept: " << Dept << endl;
        cout << "GPA: " << GPA << endl;
    }
};

// Derived class representing a Student, inheriting from Person and Academic
class Student : public Person, public Academic {
public:
    void displayStudentInfo() {
        cout << "Student Information: " << endl;
        displayPersonInfo();                    // From Person class
        displayAcademicInfo();                  // From Academic class
    }
};

int main() {
    Student studentObj;

    // Setting information from both base classes
    studentObj.setInfo("Ali Mohammad", 20);         // From Person class
    studentObj.setAcademicInfo("Computer Science", 32.75);  // From
Academic class

    // Displaying student information
    studentObj.displayStudentInfo();              // From Student class
    return 0;
}
```

Output:

```
Student Information:
Name: John Doe
Age: 20
Dept: Computer Science
GPA: 3.75
```

3. Hierarchical Inheritance

- **Definition**: In hierarchical inheritance, **multiple derived classes** inherit from a **single base class**. All derived classes share the properties and methods of the base class.
- **Characteristics**:
  - One base class.
  - Multiple derived classes.
  - Common functionality can be reused across derived classes.

Example of Hierarchical Inheritance:

```
#include <iostream>
using namespace std;

// Base class representing a Player
class Player {
public:
    string name;
    int age;

    // Constructor to initialize player details
    Player(string n, int a) {
        name = n;
        age = a;
    }

    // Function to display general player information
    void displayInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};
// Derived class representing a Football Player
class FootballPlayer : public Player {
public:
    string team;
    int goalsScored;

    // Constructor to initialize football player details
    FootballPlayer(string n, int a, string t, int g) : Player(n, a) {
        team = t;
        goalsScored = g;
    }

    // Function to display football player-specific information
    void displayFootballInfo() {
        displayInfo();  // Calling base class function to display general player info
        cout << "Team: " << team << endl;
        cout << "Goals Scored: " << goalsScored << endl;
    }
};

// Derived class representing a Basketball Player
class BasketballPlayer : public Player {
```

```
public:
    string team;
    int pointsScored;

    // Constructor to initialize basketball player details
    BasketballPlayer(string n, int a, string t, int p) : Player(n, a) {
        team = t;
        pointsScored = p;
    }

    // Function to display basketball player-specific information
    void displayBasketballInfo() {
        displayInfo();  // Calling base class function to display general player info
        cout << "Team: " << team << endl;
        cout << "Points Scored: " << pointsScored << endl;
    }
};

int main() {
    // Creating a FootballPlayer object
    FootballPlayer footballPlayer("Messi", 35, "Red Team", 15);
    cout << "Football Player Information: " << endl;
    footballPlayer.displayFootballInfo();  // Display FootballPlayer details

    cout << endl;

    // Creating a BasketballPlayer object
    BasketballPlayer basketballPlayer("Mike", 28, "Blue Team", 200);
    cout << "Basketball Player Information: " << endl;
    basketballPlayer.displayBasketballInfo();  // Display BasketballPlayer details
    return 0;
}
```

```
Output:
Football Player Information:
Name: Messi
Age: 35
Team: Red Team
Goals Scored: 15

Basketball Player Information:
Name: Mike
Age: 28
Team: Blue Team
Points Scored: 200
```

Explanation:

- `footballPlayer` and `basketballPlayer` are objects of the derived classes `FootballPlayer` and `BasketballPlayer`, respectively.
- Both objects can access the common functionality of the `Player` base class (like `name` and `age`), as well as their own specific data (like `team` and `goalsScored` for football, or `team` and `pointsScored` for basketball).

---

4. Multilevel Inheritance

- **Definition**: In multilevel inheritance, a derived class becomes the base class for another derived class. Essentially, one class inherits from another, which in turn inherits from another.
- **Characteristics**:
  - A class inherits from another derived class.
  - Forms a chain of inheritance.

Example of Multilevel Inheritance:

```
#include <iostream>
using namespace std;

// Base class representing a Person
class Person {
public:
    string name;
    int age;
    // Constructor to initialize person details
    Person(string n, int a) {
        name = n;
        age = a;
    }
    // Function to display person information
    void displayPersonInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};
// Derived class representing a Student (inherits from Person)
class Student : public Person {
public:
    string studentID;
    // Constructor to initialize student details
    Student(string n, int a, string id) : Person(n, a) {
        studentID = id;
    }
    // Function to display student-specific information
    void displayStudentInfo() {
        displayPersonInfo();  // Calling base class function to display
person info
        cout << "Student ID: " << studentID << endl;
    }
};
// Further derived class representing a GraduateStudent (inherits from Student)
class GraduateStudent : public Student {
public:
    string thesisTitle;
    // Constructor to initialize graduate student details
    GraduateStudent(string n, int a, string id, string thesis) :
Student(n, a, id) {
        thesisTitle = thesis;
    }
```

```
    // Function to display graduate student-specific information
    void displayGraduateStudentInfo() {
        displayStudentInfo();      // Calling base class function to display student info
        cout << "Thesis Title: " << thesisTitle << endl;
    }
};
int main() {
    // Creating an object of GraduateStudent
    GraduateStudent gradStudent("Gaith Amer", 25, "S-23381511", "AI in
Cyber security");

    cout << "Graduate Student Information: " << endl;
    gradStudent.displayGraduateStudentInfo();   // Display all information of GraduateStudent
    return 0;
}
```

```
Output:
Graduate Student Information:
Name: Gaith Amer
Age: 25
Student ID: S-23381511
Thesis Title: AI in Healthcare
```

Explanation:

- The `GraduateStudent` class is derived from the `Student` class, which in turn is derived from the `Person` class. This creates a multilevel inheritance hierarchy.
- **Code Reusability**: The derived classes can reuse the functionality of the base class and intermediate class, allowing for efficient code structure and reducing redundancy.
- **Specific Attributes**: Each derived class adds more specific attributes, such as `studentID` in `Student` and `thesisTitle` in `GraduateStudent`, demonstrating how inheritance can extend base class functionality.

5. Hybrid Inheritance

- **Definition**: Hybrid inheritance is a combination of two or more types of inheritance. It can include combinations like **multiple** and **multilevel** inheritance.
- **Characteristics**:
  - It combines various types of inheritance, often leading to more complex structures.
  - Requires careful design to avoid ambiguities and issues like the **diamond problem**.

Example of Hybrid Inheritance:

```
#include <iostream>
using namespace std;
// Base class representing a Person
class Person {
public:
    string name;
    int age;
```

```
    // Constructor to initialize person details
    Person(string n, int a) {
        name = n;
        age = a;
    }
    // Function to display person information
    void displayPersonInfo() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};
// Derived class representing a Student (inherits from Person)
class Student : public Person {
public:
    string studentID;
    // Constructor to initialize student details
    Student(string n, int a, string id) : Person(n, a) {
        studentID = id;
    }
    // Function to display student-specific information
    void displayStudentInfo() {
        displayPersonInfo();    // Calling base class function to display person info
        cout << "Student ID: " << studentID << endl;
    }
};
// Derived class representing an Employee (inherits from Person)
class Employee : public Person {
public:
    string employeeID;
    // Constructor to initialize employee details
    Employee(string n, int a, string id) : Person(n, a) {
        employeeID = id;
    }

    // Function to display employee-specific information
    void displayEmployeeInfo() {
        displayPersonInfo();    // Calling base class function to display person info
        cout << "Employee ID: " << employeeID << endl;
    }
};
// Hybrid class representing a GraduateEmployee (inherits from both Student and Employee)
class GraduateEmployee : public Student, public Employee {
public:
    string department;
    // Constructor to initialize graduate employee details
    GraduateEmployee(string n, int a, string studentID, string employeeID,
string dept): Person(n, a), Student(n, a, studentID), Employee(n, a,
employeeID) {
        department = dept;
    }

    // Function to display graduate employee-specific information
    void displayGraduateEmployeeInfo() {
        displayStudentInfo();        // Calling base class function to display student info
        displayEmployeeInfo();       // Calling base class function to display employee info
        cout << "Department: " << department << endl;
    }
};
int main() {
    // Creating an object of GraduateEmployee
    GraduateEmployee gradEmp("Alice", 25, "S12345", "E67890", "Research and
Development");
```

```
    cout << "Graduate Employee Information: " << endl;
    gradEmp.displayGraduateEmployeeInfo();  // Display all information of GraduateEmployee
    return 0;
}
```

```
Output:
Graduate Employee Information:
Name: Alice
Age: 25
Student ID: S12345
Name: Alice
Age: 25
Employee ID: E67890
Department: Research and Development
```

Explanation:

- **Hybrid Inheritance**: This example combines **Multiple Inheritance** (with the `GraduateEmployee` class inheriting from both `Student` and `Employee`) and **Multilevel Inheritance** (since both `Student` and `Employee` are derived from the `Person` class).
- **Code Reusability**: Both `Student` and `Employee` inherit functionality from the base `Person` class, and the `GraduateEmployee` class combines features from both derived classes.
- **Function Overriding**: The `GraduateEmployee` class does not override any functions in this case, but it demonstrates how you can access and combine member functions from multiple base classes.

Conclusion

Inheritance is a powerful feature in object-oriented programming that allows us to reuse code, create hierarchical relationships, and extend functionality. Understanding the different types of inheritance (single, multiple, hierarchical, multilevel, and hybrid) is crucial for designing flexible and maintainable code.

Key Takeaways:

- **Single Inheritance**: One base class, one derived class.
- **Multiple Inheritance**: One derived class inherits from multiple base classes.
- **Hierarchical Inheritance**: Multiple derived classes inherit from one base class.
- **Multilevel Inheritance**: A class inherits from another derived class.
- **Hybrid Inheritance**: A combination of two or more types of inheritance.