3. Classes and Objects

3.1 Understanding Classes and Objects

Definition:

- **Class**: A blueprint or template for creating objects (like a cookie cutter)
- **Object**: An instance of a class (like a cookie made from the cutter)

Real-World Example:

```
Class: Student (blueprint)
    - Attributes: name, ID, GPA
    - Methods: study(), takeExam(), getGPA()

Objects (instances):
    - student1: "Alice", "S001", 3.8
    - student2: "Bob", "S002", 3.5
    - student3: "Charlie", "S003", 3.9
```

3.2 Defining a Class

Basic Class Syntax:

```
class ClassName {
private:
    // Private members (data and functions)
    // Only accessible within the class

public:
    // Public members
    // Accessible from outside the class

protected:
    // Protected members
    // Accessible in derived classes (inheritance)
};
```

Simple Example:

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

class Student {
private:
    string name;
    int id;
    float gpa;

public:
    // Constructor
    Student(string n, int i, float g) {
        name = n;
        id = i;
        gpa = g;
    }

    // Member functions (methods)
    void displayInfo() {
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
        cout << "GPA: " << gpa << endl;
    }

    void study() {
        cout << name << " is studying..." << endl;
    }

    float getGPA() {
        return gpa;
    }

    void setGPA(float newGPA) {
        if (newGPA >= 0.0 && newGPA <= 4.0) {
            gpa = newGPA;
        } else {
            cout << "Invalid GPA!" << endl;
```

```
        }
    }
};

int main() {
    // Creating objects
    Student student1("Alice", 1001, 3.8);
    Student student2("Bob", 1002, 3.5);

    // Using objects
    student1.displayInfo();
    cout << endl;

    student2.study();
    cout << "Bob's GPA: " << student2.getGPA() << endl;

    student2.setGPA(3.7);
    cout << "Updated GPA: " << student2.getGPA() << endl;

    return 0;
}
```

3.3 Procedural vs OOP: A Practical Comparison

Procedural Approach (C):

```
#include <stdio.h>
#include <string.h>

// Separate data structure
struct Student {
    char name[50];
    int id;
    float gpa;
};

// Separate functions
void displayStudent(struct Student s) {
    printf("Name: %s\n", s.name);
    printf("ID: %d\n", s.id);
```

```
    printf("GPA: %.2f\n", s.gpa);
}

void studyStudent(struct Student s) {
    printf("%s is studying...\n", s.name);
}

float getGPA(struct Student s) {
    return s.gpa;
}

int main() {
    struct Student student1;
    strcpy(student1.name, "Alice");
    student1.id = 1001;
    student1.gpa = 3.8;

    displayStudent(student1);
    studyStudent(student1);

    return 0;
}
```

OOP Approach (C++):

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

class Student {
private:
    string name;
    int id;
    float gpa;

public:
    Student(string n, int i, float g) : name(n), id(i), gpa(g) {}

    void display() {
        cout << "Name: " << name << endl;
```

```
        cout << "ID: " << id << endl;
        cout << "GPA: " << gpa << endl;
    }

    void study() {
        cout << name << " is studying..." << endl;
    }

    float getGPA() { return gpa; }
};

int main() {
    Student student1("Alice", 1001, 3.8);

    student1.display();
    student1.study();

    return 0;
}
```

Key Advantages of OOP:

1. **Encapsulation**: Data and functions are bundled together
2. **Data Protection**: Private members prevent unauthorized access
3. **Cleaner Syntax**: Methods are called directly on objects
4. **Better Organization**: Related functionality is grouped together

---

Revision #2
Created 2025-11-09 13:14:39 UTC by DS
Updated 2025-11-09 13:15:22 UTC by DS