CS217 – Object Oriented Programming (OOP)

Week – 02

Feb 15-19, 2021

Instructor: **Basit Ali**

Structure v/s Class

`struct` versus `class`

- In C++ struct and class can be used interchangeably to create a class with one exception
- What if we forget to put an access modifier before the first field?

```
struct Robot {        OR        class Robot {
   float locX;                     float locX;
```

In a class, until an access modifier is supplied, the fields are assumed to be `private`

In a struct, the fields are assumed to be `public`

INFORMATION HIDING

- Information is stored within the object
- It is hidden from the outside world
- It can only be manipulated by the object itself

Example

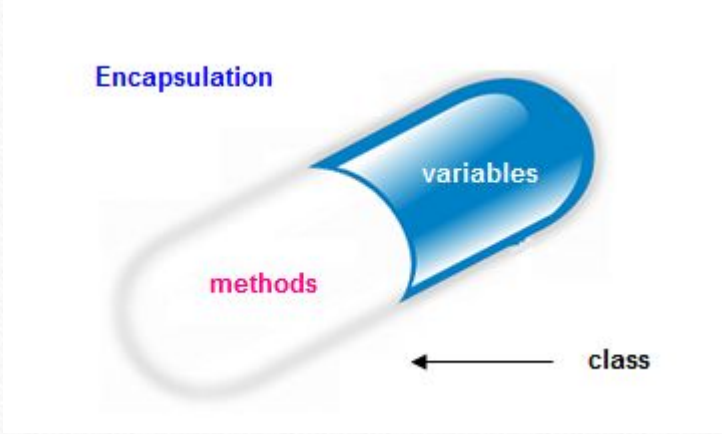
Encapsulation (1st Principle of OOP)

- **Encapsulation** is a process of wrapping of data and methods in a single unit

- The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real Life Example of Encapsulation

- The common example of encapsulation is **Capsule**. In capsule all medicine are encapsulated inside capsule.



Real Life Example of Encapsulation

- A Phone stores phone numbers in digital format and knows how to convert it into human-readable characters

- We don't know
  - How the data is stored
  - How it is converted to human-readable characters

ENCAPSULATION – ADVANTAGES

- Simplicity and clarity
- Low complexity
- Better understanding

Abstraction in OOP (2nd Principle)

- Abstraction is selecting data from a larger pool to show only the relevant details to the object.
- Abstraction is a way to cope with complexity.
- Principle of abstraction:

"Capture only those details about an object that are relevant to current perspective"

Example – Abstraction

- Ali is a PhD student and teaches BS students
- **Attributes**

- Name	- Employee ID
- Student Roll No	- Designation
- Year of Study	- Salary
- CGPA	- Age

Example – Abstraction

- Ali is a PhD student and teaches BS students
- **Behavior**

  - Study
  - GiveExam
  - PlaySports
  - DeliverLecture
  - DevelopExam
  - TakeExam
  - Eat
  - Walk

Example – Abstraction

Student's Perspective

- **Attributes**
  - Name
  - Student Roll No
  - Year of Study
  - CGPA
  - ~~Employee ID~~
  - ~~Designation~~
  - ~~Salary~~
  - ~~Age~~

Example – Abstraction

Student's Perspective

- **Behavior**
  - Study
  - GiveExam
  - PlaySports
  - ~~DeliverLecture~~
  - ~~DevelopExam~~
  - ~~TakeExam~~
  - ~~Eat~~
  - ~~Walk~~

Example – Abstraction

Teacher's Perspective

- **Attributes**
  - Name
  - ~~Student Roll No~~
  - ~~Year of Study~~
  - ~~CGPA~~
  - Employee ID
  - Designation
  - Salary
  - ~~Age~~

Example – Abstraction

Teacher's Perspective

- **Behavior**
  - ~~Study~~
  - ~~GiveExam~~
  - ~~PlaySports~~
  - DeliverLecture
  - DevelopExam
  - TakeExam
  - ~~Eat~~
  - ~~Walk~~

Example – Abstraction

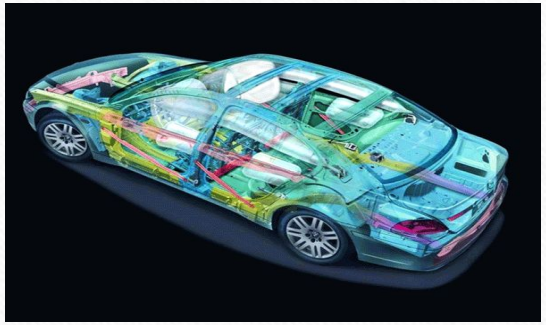
- A cat can be viewed with different perspectives
  - **Ordinary Perspective**

    A pet animal with
    - Four Legs
    - A Tail
    - Two Ears
    - Sharp Teeth
  - **Surgeon's Perspective**

    A being with
    - A Skeleton
    - Heart
    - Kidney
    - Stomach

Example – Abstraction

Driver's View



Engineer's View

Abstraction – Advantages

- Simplifies the model by hiding irrelevant details
- Abstraction provides the freedom to defer implementation decisions by avoiding commitment to details

Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. **Abstraction**- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. **Encapsulation**- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.

Syntax

- class
- {
- private:
- // private members and function
- public:
- // public members and function
- protected:
- // protected members and function
- };

A Simple Class

```
class Robot {
  public:
    float getX() { return locX; }
    float getY() { return locY; }
    float getFacing() { return facing; }
    void setFacing(float f) { facing = f; }
    void setLocation(float x, float y);
  private:
    float locX;
    float locY;
    float facing;
};
```

Access Specifiers

- Public
- Private
- Protected

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

`private` Access Modifier

- Fields marked as `private` can only be accessed by functions that are part of that class
- In the Robot class, locX, locY, and facing are private float fields, these fields can only be accessed by functions that are in class Robot (getX, getY, getFacing, setFacing, setLocation)
- Example:

```
void useRobot() {
  Robot r1;
  r1.locX = -5; // Error
```

`public` Access Modifier

- Fields marked as `public` can be accessed by anyone
- In the Robot class, the methods getX, getY, etc. are public
  - these functions can be called by anyone
- Example:

```
void useRobot() {
  Robot r1;
  r1.setLocation(-5,-5); // Legal to call
```

Class Methods

- Functions associated with a class are declared in one of two ways:

  `ReturnType FuncName(params) { code }`

  - function is both declared and defined (code provided)

  `ReturnType FuncName(params);`

  - function is merely declared, we must still define the body of the function separately

- To call a method we use the . form:

  classinstance.*FuncName*(*args*);

  FuncName is a field just like any other field in the structured variable *classinstance*

Defined Methods

```
class Robot {
  public:
    float getX() { return locX; }
};
Robot r1;
```

- The function getX is defined as part of class Robot
- To call this method:

```
cout << r1.getX() << endl; // prints r1's locX
```

Defining Methods Separately

- For methods that are declared but not defined in the class we need to provide a separate definition
- To define the method, you define it as any other function, except that the name of the function is *ClassName*::*FuncName*

  :: is the scope resolution operator, it allows us to refer to parts of a class or structure

Getter/Setter Functions

- Getter functions (or accessor functions) are used to read value of a private member of some class

- Setter functions (or mutator functions) are used to modify the value of a private member of some class

Example

```
class BankAccount
{
      int PIN;       //private variable

   Public:
    int get_PIN()
    {
      return PIN;
    }
}
```

Example

```
class BankAccount
{
      int accountNo;   //private variable

    Public:
    void set_accountNo(int num)
    {
      accountNo = num;
    }
}
```

Example: A Simple Class

```
class Robot {
  public:
    void setLocation(float x, float y);
  private:
    float locX;
    float locY;
    float facing;
};

void Robot::setLocation(float x, float y) {
  if ((x < 0.0) || (y < 0.0))
    cout << "Illegal location!!" << endl;
  else {
    locX = x;
    locY = y;
  }
}
```

Case Study

- A man who manages a scoreboard wants a simple application module to manage the history of a batsman. For every new batsman, the app must let us fill the details including the Id, Name, Age, Runs, avg, etc. These details may be modified later except for the ID of a batsman. At anytime a batsman can check his runs and his average.

Exercise!

```
class Batsman
{
    int ID;
    string name;
    int age;
    int runs;
    int avg;

    public:

    set_batsman(int m, string a, int b)
    {
        ID = m;
        name = a;
        age = b;
        runs = 0;
        avg = 0;
    }

    display()
    {
        cout<<name<<endl;
        cout<<age<<endl;
        cout<<runs<<endl;
        cout<<avg<<endl;
    }

};
```

```
int main()
{
    Batsman b1;
    b1.set_batsman(1, "Basit", 18);
    b1.display();
    getchar();
    return 1;
}
```

Questions?

- Find the name of the batsman(out of three batsmen) who has the highest runs?
- Find the name of the batsman with highest average runs?
- Find the batsman who has played most matches?