BATCH

RECURSION

C AND C++

PROGRAMMING MASTERCLASS

AWESH ISLAM
BUET, CSE

C++ Class-02

SHAROARE HOSAN EMON
BME , BUET

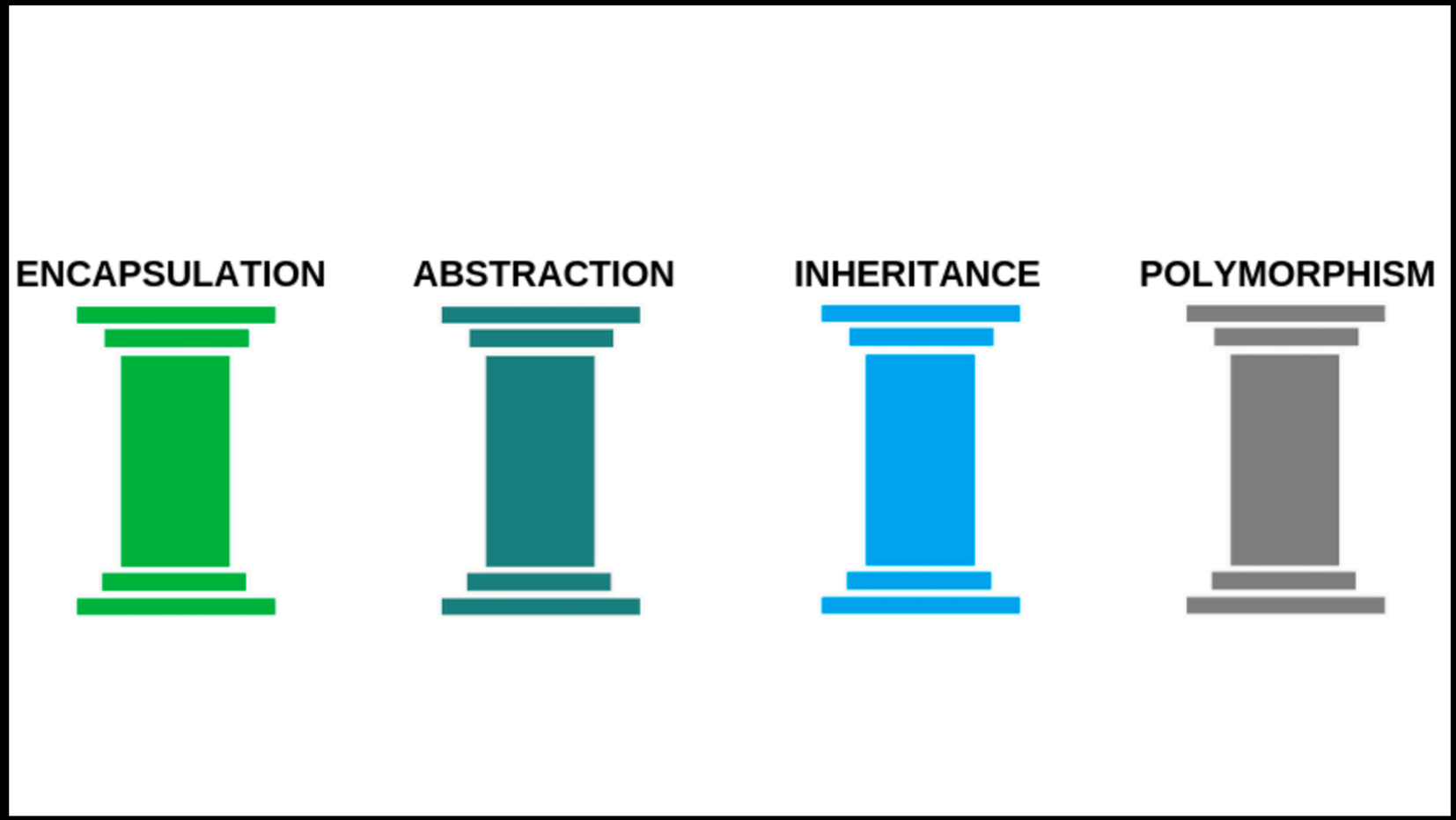
আমাদের সবগুলো ক্লাস দেখার জন্য ভিজিট করো

https://www.hsccrackers.com/

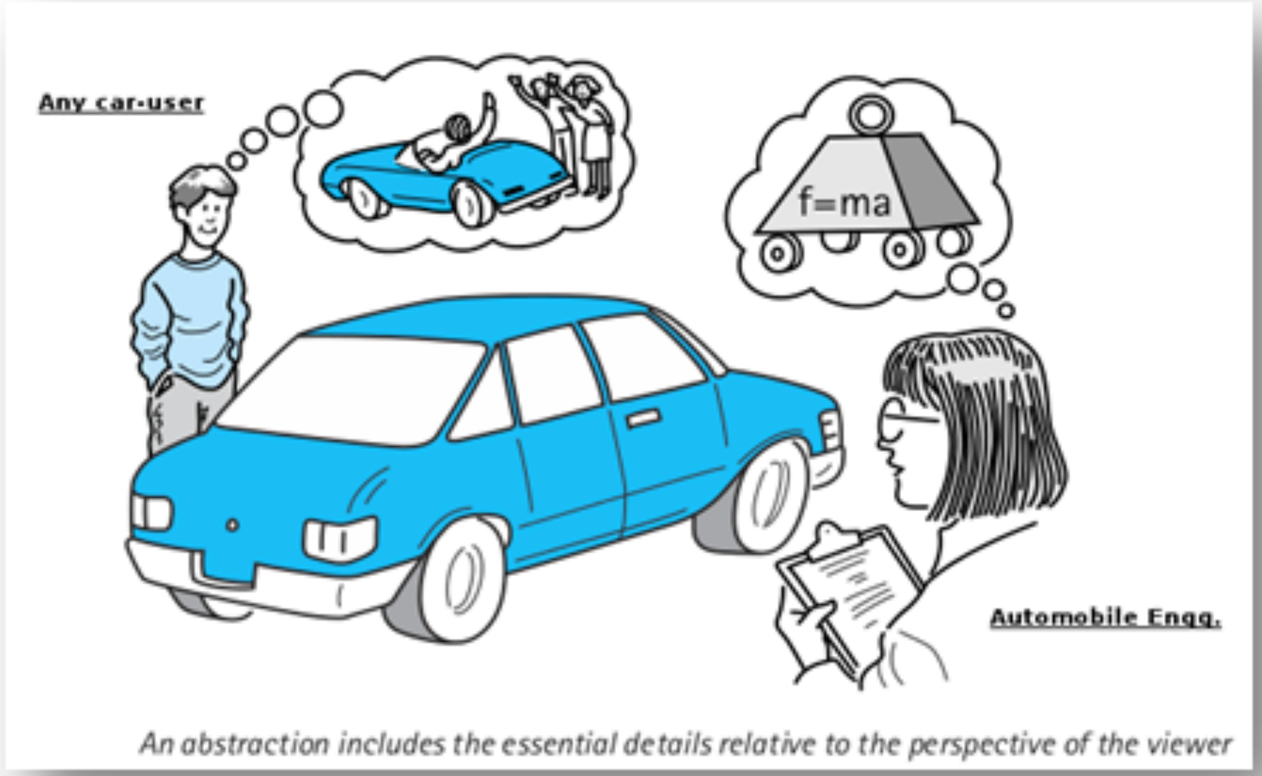
SCAN ME

4 pillars of OOP

4 pillars of OOP



Abstraction



An abstraction includes the essential details relative to the perspective of the viewer

Abstraction

```
47        cout<<"Hello World"<<endl;
```

Types of Abstraction

Types of abstraction:

1. **Data abstraction –** This type only shows the required information about the data and hides the unnecessary data.

2. **Control Abstraction –** This type only shows the required information about the implementation and hides unnecessary information.

Abstraction In C++

```
cout<<pow(2, 8)<<endl;
```

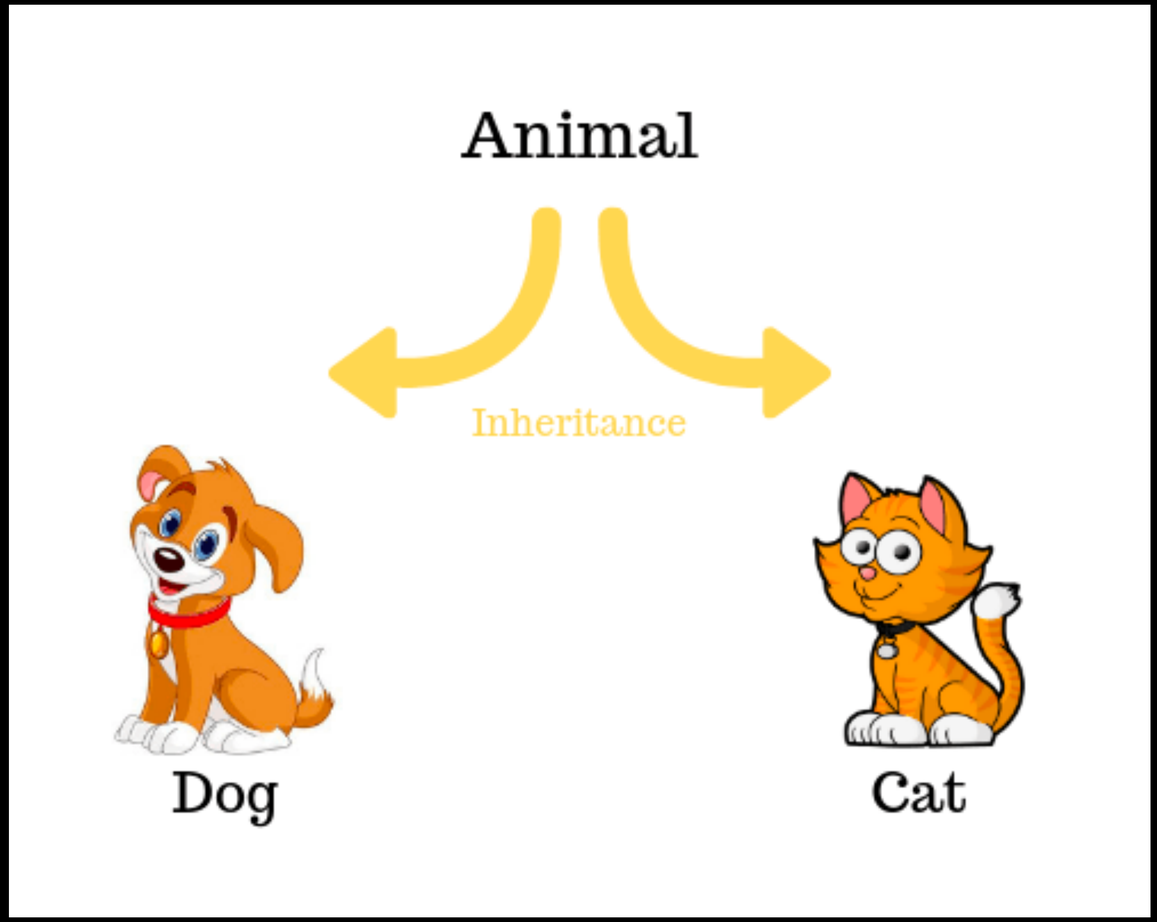
```
30      string s = "Awesh Islam";
31      cout<<s.length()<<endl;
```

Abstraction In C++

```
double calculateCgpa(double cg_term,double credit_term){
    double average = (cg_term*credit_term+credit_completed*cgpa)/(credit_term+credit_completed);
    credit_completed += credit_term;
    cgpa = average;
    return average;
}
```

```
76      a.credit_completed = 40;
77      a.calculateCgpa(3.5, 19.5);
78      cout<<a.cgpa<<endl;
```

Inheritance



Inheritance

```
class Human{
public:
    string name;
    int age;
    double height;
    double weight;

};

class Student: public Human{
```

Base Class or Super Class

Derived Class or Sub Class

```
class Vector2D{
public:
    double x,y;
};
class Vector3D: public Vector2D{
    double z;
};
```

```
a.age = 25;
a.height = 150;
```

Inheritance

```
7  class Vector2D{
8  public:
9      double x,y;
10 };
11 class Vector3D: public Vector2D{
12     double z;
13 };
```

Base Class or Super Class

Derived Class or Sub Class

Derived Class Constructor

```
7  class Vector2D{
8  public:
9      double x,y;
10     Vector2D(double X,double Y){
11         x = X;
12         y = Y;
13     }
14 };
15 class Vector3D: public Vector2D{
16 public:
17     double z;
18     Vector3D(double X,double Y,double Z):Vector2D(X, Y){
19         z = Z;
20     }
21 };
22
23
24 int main() {
25     Vector3D a = Vector3D(1,2,3);
26     cout<<"x = "<<a.x<<endl;
27     cout<<"y = "<<a.y<<endl;
28     cout<<"z = "<<a.z<<endl;
29 }
30
```

Derived Class Access to Variables

```
21      void print(){
22          cout<<"The vector is: "<<x<<"i + "<<y<<"j + "<<z<<"k"<<endl;
23      }
```

Protected Modifier

```
6
7  class Vector2D{
8  protected:
9      double x,y;
10     Vector2D(double X,double Y){
11         x = X;
12         y = Y;
13     }
14 };
15 class Vector3D: public Vector2D{
16 public:
17     double z;
18     Vector3D(double X,double Y,double Z):Vector2D(X, Y){
19         z = Z;
20     }
21     void print(){
22         cout<<"The vector is: "<<x<<"i + "<<y<<"j + "<<z<<"k"<<endl;
23     }
24 };
25
26
27 int main() {
28     Vector3D a = Vector3D(1,2,3);
29     cout<<"x = "<<a.x<<endl;
30     cout<<"y = "<<a.y<<endl;
31     cout<<"z = "<<a.z<<endl;
32     a.print();
33 }
```

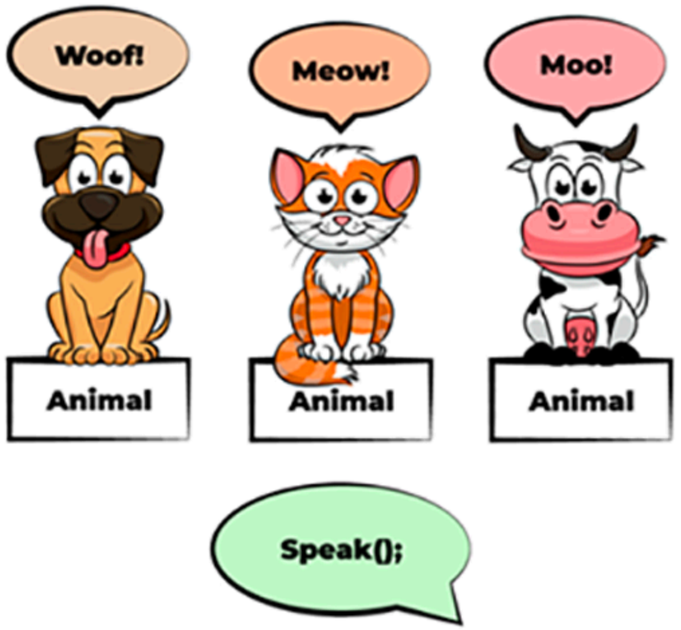
Can Access

Cannot Access

'x' is a protected member of 'Vector2D'

'y' is a protected member of 'Vector2D'

Polymorphism



Polymorphism

```c
#include<stdio.h>
#include<math.h>
int main(){
    int a = 10;
    printf("%d\n",abs(a));
    float b = - 10.5;
    printf("%lf\n",fabs(b));
    long c = -10000000000;
    printf("%ld\n",labs(c));
}
```

```cpp
#include<iostream>
using namespace  std;
int main(){
    int a = 10;
    printf("%d\n",abs(a));
    float b = - 10.5;
    printf("%lf\n",abs(b));
    long c = -10000000000;
    printf("%ld\n",abs(c));
}
```

Function Overloading

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

class Athlete{
public:
    int speed = 10;
    void run(){
        cout<<"Athlete is running "<<speed<<" meter per second"<<endl;
    }
    void run(int given_speed){
        speed = given_speed;
        cout<<"Athlete is running "<<speed<<" meter per second"<<endl;
    }
};
int main(){
    Athlete bolt;
    bolt.run();
    bolt.run(20);
}
```

```
Athlete is running 10 meter per second
Athlete is running 20 meter per second
Program ended with exit code: 0
```

Operator Overloading

```
15      Vector3D operator+(Vector3D b){
16          Vector3D temp(0,0,0);
17          temp.x = x + b.x;
18          temp.y = y + b.y;
19          temp.z = z + b.z;
20          return temp;
21      }
22      Vector3D operator*(int a){
23          Vector3D temp(0,0,0);
24          temp.x = x*a;
25          temp.y = y*a;
26          temp.z = z*a;
27          return temp;
28      }
29      Vector3D operator*(Vector3D b){
30          Vector3D temp(0,0,0);
31          temp.x = x*b.x;
32          temp.y = y*b.y;
33          temp.z = z*b.z;
34          return temp;
35      }
```

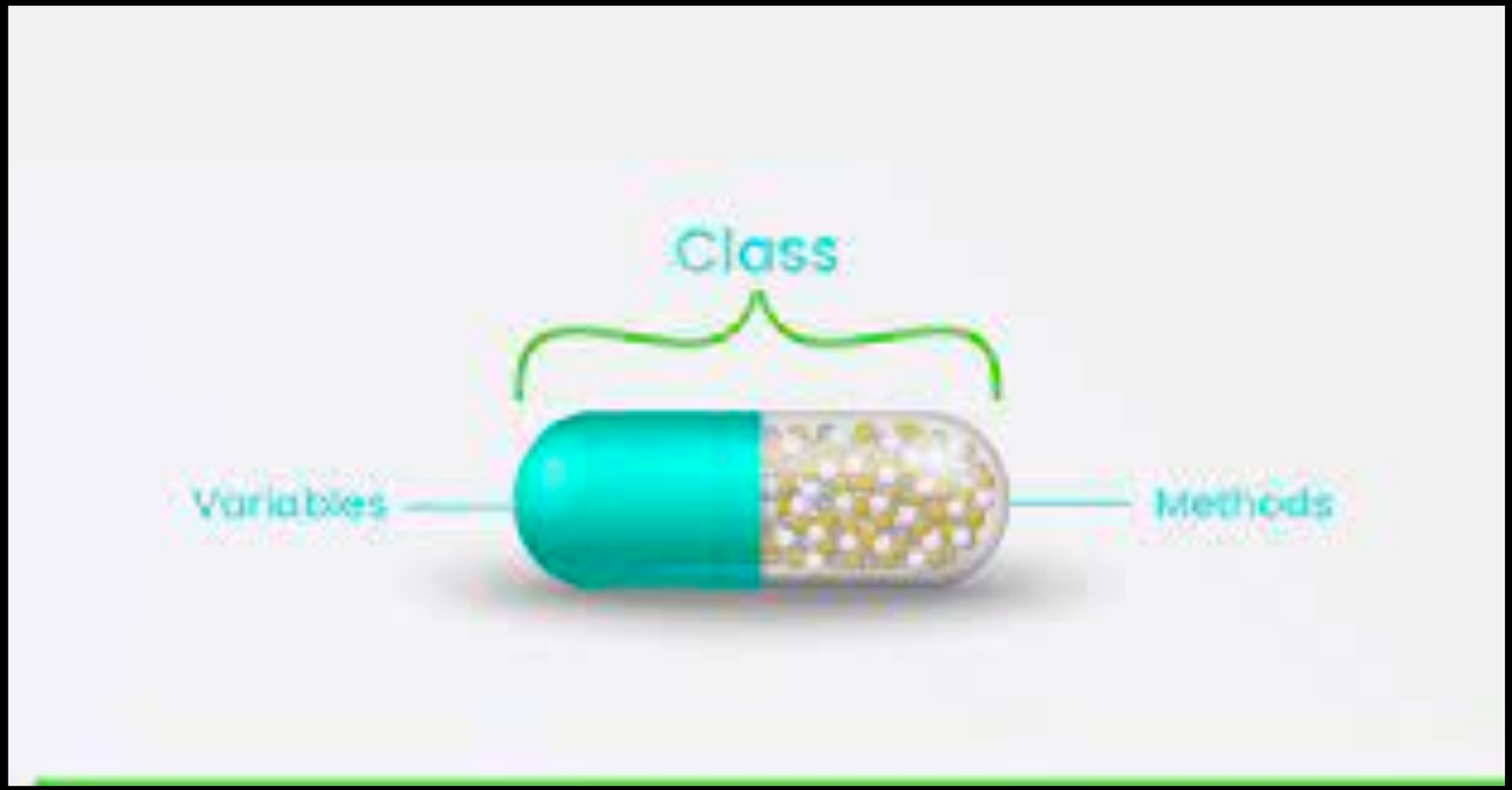
Polymorphism

```
4  class Animal{
5  public:
6      virtual void speak() = 0;
7  };
8  class Cat: public Animal{
9      void speak(){
10         cout<<"Meow Meow"<<endl;
11     }
12 };
13 class Dog: public Animal{
14     void speak(){
15         cout<<"Gheu Gheu"<<endl;
16     }
17 };
18 class Cow: public Animal{
19     void speak(){
20         cout<<"Hamba Hamba"<<endl;
21     }
22 };
```

```
24 int main(){
25     Animal *a;
26     Cat mini;
27     Dog jonny;
28     Cow hamba;
29     a = &mini;
30     a->speak();
31     a = &jonny;
32     a->speak();
33     a = &hamba;
34     a->speak();
35 }
```

```
Meow Meow
Gheu Gheu
Hamba Hamba
```

Encapsulation



Encapsulation

Important properties of encapsulation:

1. **Data Protection:** Encapsulation protects the internal state of an object by keeping its data members private. Access to and modification of these data members is restricted to the class's public methods, ensuring controlled and secure data manipulation.
2. **Information Hiding:** Encapsulation hides the internal implementation details of a class from external code. Only the public interface of the class is accessible, providing abstraction and simplifying the usage of the class while allowing the internal implementation to be modified without impacting external code.

Encapsulation

Problem:

```
71  int main(){
72
73      Student a("Awesh Islam",2005054,3.75);
74      a.cgpa = 5;
75      a.show_result();
76
77  }
78
```

```
My cgpa is : 5
Program ended with exit code: 0
```

Encapsulation

Solution:

```
6  class Student{
7  private:
8      string name;
9      int roll;
10     double cgpa;
11     double credit_completed;
```

```
21     void setCGPA(double CGPA){
22         if (CGPA <= 4.0){
23             cgpa = CGPA;
24         }
25     }
```

Encapsulation

Solution:

```
6  class Student{
7  private:
8      string name;
9      int roll;
10     double cgpa;
11     double credit_completed;
```

```
21     void setCGPA(double CGPA){
22         if (CGPA <= 4.0){
23             cgpa = CGPA;
24         }
25     }
```

Setters Getters

```
7  class Vector3D{
8  private:
9      double x,y,z;
10 public:
11     Vector3D(double X,double Y,double Z){
12         x = X;
13         y = Y;
14         z = Z;
15     }
16     void setX(double X){
17         x = X;
18     }
19     void setY(double Y){
20         y = Y;
21     }
22     void setZ(double Z){
23         z = Z;
24     }
25     double getX(){
26         return x;
27     }
28     double getY(){
29         return y;
30     }
31     double getZ(){
32         return z;
33     }
34 };
```

Encapsulation

Solution:

```
62  int main(){
63
64      Student a("Awesh Islam",2005054,3.75);
65      a.setCGPA(5);
66      a.show_result();
67
68  }
```

```
My cgpa is : 3.75
```

```
62  int main(){
63
64      Student a("Awesh Islam",2005054,3.75);
65      a.setCGPA(3.5);
66      a.show_result();
67
68  }
```

```
My cgpa is : 3.5
Program ended with exit code: 0
```

Abstraction Vs Encapsulation

Abstraction is hiding unnecessary details from the user.
Meanwhile encapsulation is hiding important data from the user for privacy concerns.