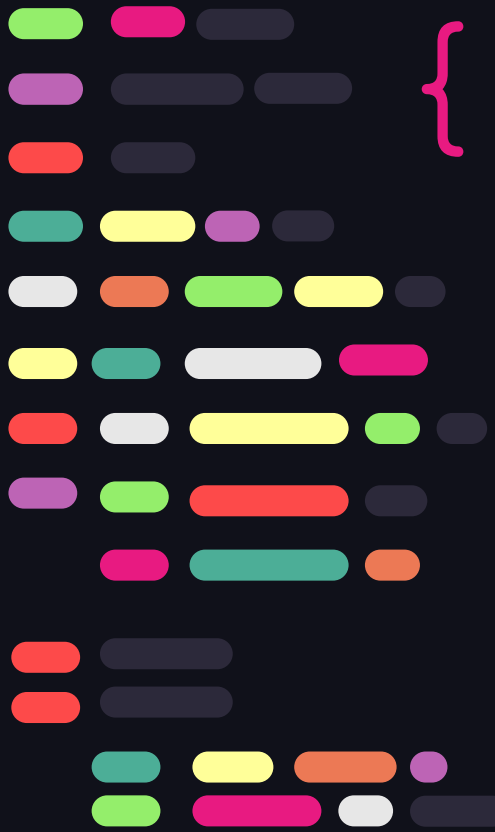


Object Oriented Programming

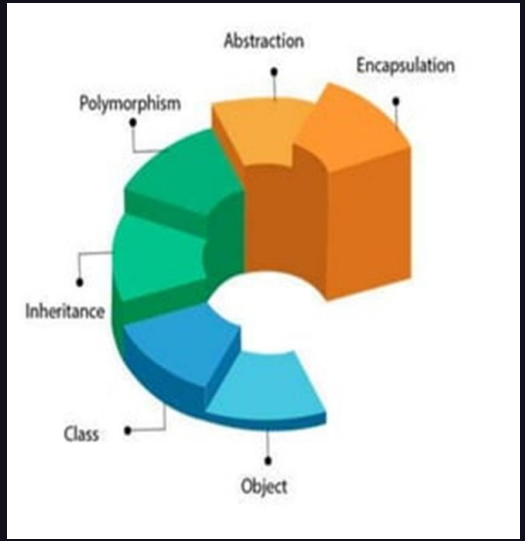
```
< Shreerang Mhatre == 52
  Sarvesh Gurav == 44
  Tanvi Kariyappa == 56 >
```

Introduction

The object-oriented paradigm is a programming methodology that promotes the efficient design and development of software systems using reusable components that can be quickly and safely assembled into larger systems.

The main aim of object-oriented programming is to implement real- world concepts like

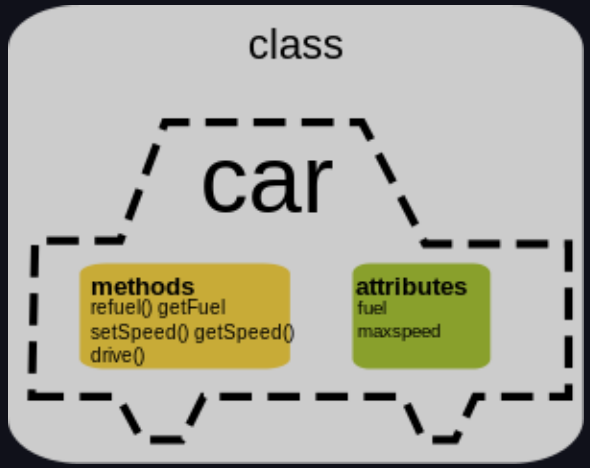
Objects	Real world entity
Classes	Templates/ Blueprints
Encapsulation	Bundling of Data
Abstraction	Visibility Controls
Inheritance	Backward Compatibility , parent child relation
Polymorphism	Many forms



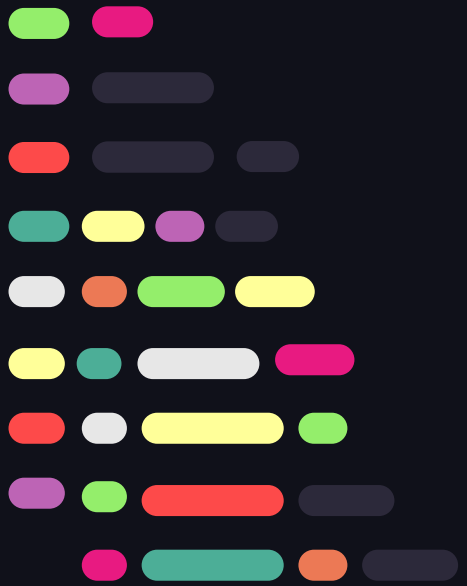
Class

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity.

-It is non primitive data type.
-It cant be physical(no memory space)
-Class members are access modifiers, objects, Methods, Instance variable and constructors.



Example: Class

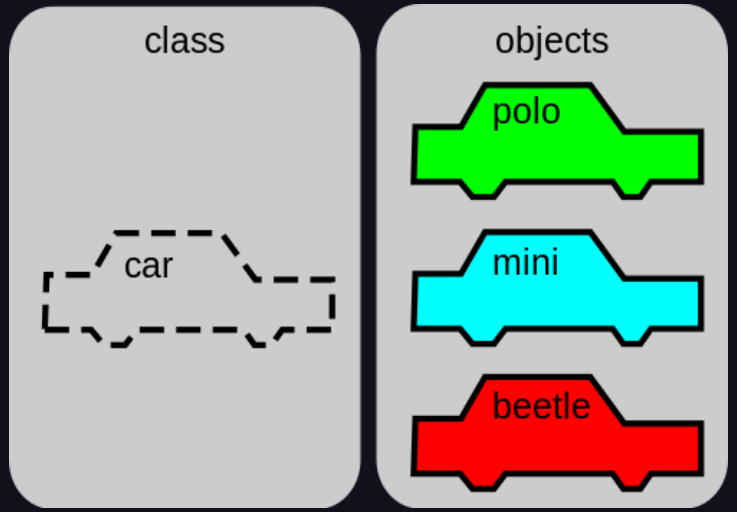


Created a class called "MyClass":

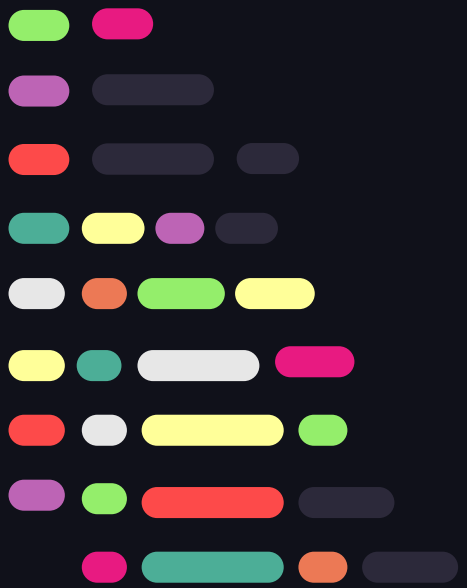
```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};
```

Object

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.



Example: Object

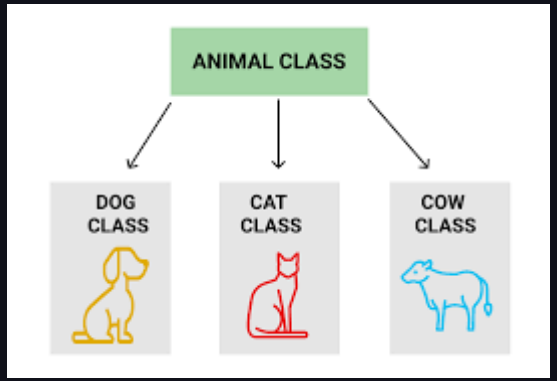


```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Inheritance

Inheritance is a mechanism in which one Object acquires all the properties and behaviors of a parent object.
Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
Like Animal is a Mammals , Reptiles or Birds.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.



Types Of inheritance:
- → Single
- → Multilevel
- → Hierarchical
- → Multiple
- → Hybrid

Example: Inheritance

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};
```

```
int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
}
```

Polymorphism

If one task is performed by different ways, it is known as polymorphism.

For example: To convince the customer differently, to draw something, like shape, triangle, rectangle, a cat speaks meow. dog barks woof, etc.

Polymorphism present a method that can have many definitions.
Polymorphism is related to
Overloading:- Compile time polymorphism/run time polymorphism
Overriding:- Run time polymorphism/ Dynamic polymorphism


```
Syntax:
getPrice()
getPrice(string name)
```

Example: Polymorphism

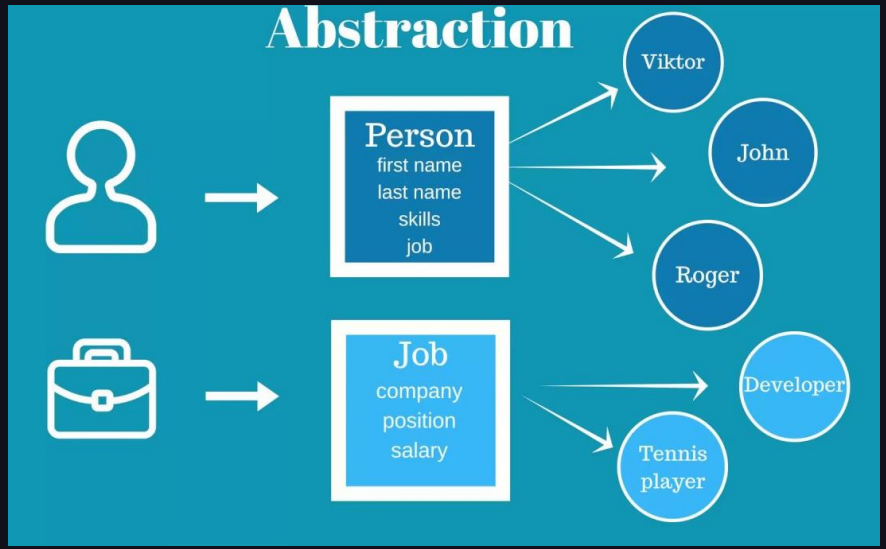
```
// Base class
class Animal {
  public:
    void animalSound() {
      cout << "The animal makes
a sound \n";
    }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
      cout << "The pig says: wee
wee \n";
    }
};
```

```
// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
      cout << "The dog says: bow wow \n";
    }
};
```

Abstraction

Abstraction is basically hiding the implementation and gain access to there functionality by exposing by extend keyword.

-An abstract class is a class that may not have any direct instances.
-An abstract operation is an operation that it is incomplete and requires a child to supply an implementation of the operation.



Example: Abstraction

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
};
```

NOTE:

We can use access specifiers to enforce restrictions on class members. For example:

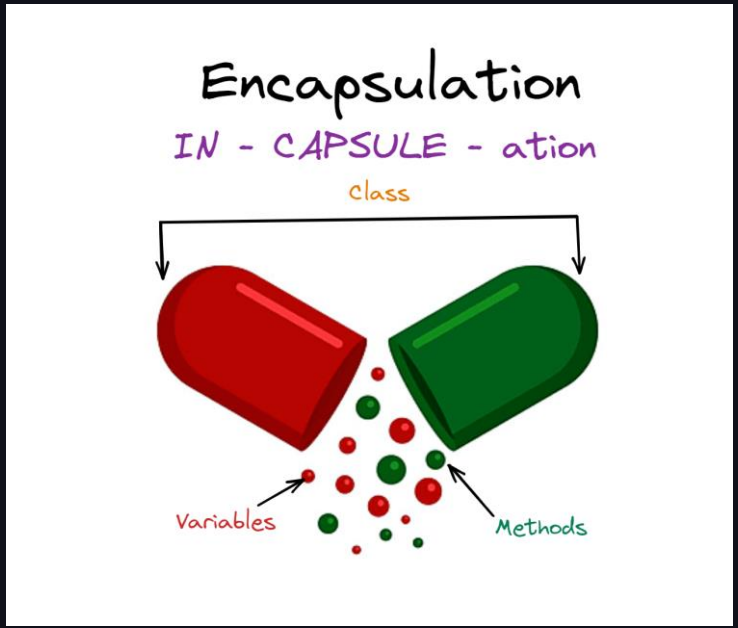
- Members declared as **public** in a class can be accessed from anywhere in the program.
- Members declared as **private** in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of the code outside the class.

Encapsulation

Encapsulation in C++ is defined as the wrapping up of data and information in a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. Now,

- The finance section handles all the financial transactions and keeps records of all the data related to finance.
- Similarly, the sales section handles all the sales-related activities and keeps records of all the sales.



Example: Encapsulation

```
class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};
```

```
int main() {
  Employee myObj;
  myObj.setSalary(50000
);
  cout <<
myObj.getSalary();
  return 0;
}
```

To access a private attribute, use public "get" and "set" methods:

{ Thankyou }