**Meta class**

A meta class is a class of a class i.e., the objects of this class can themselves act as classes. So, a user can add or remove attributes at run time.

In object-oriented computer programming, a **meta class** is one whose instances are also classes. For instance, in Python, the built-in class **type** is a meta class: instances of class **type** are themselves a class of objects.

The use of meta classes is most prevalent in object-oriented languages. In these languages, classes are "first-class" objects, meaning they can be used exactly like any other object in the language.

* They can be named variables.
* They can be passed as an argument to methods, functions, and procedures.
* They can be returned as the result of a method, function, or procedure.
* They can be included as part of a data structure.

It is possible, but uncommon, to implement meta classes in C++ and Java. More commonly, meta classes are found in programs written in Python, Ruby, Smalltalk, Perl.

**C++ Pure Virtual Functions**

Pure virtual functions are used

* if a function doesn't have any use in the base class
* but the function must be implemented by all its derived classes

Let's take an example,

Suppose, we have derived Triangle, Square and Circle classes from the Shape class, and we want to calculate the area of all these shapes.

In this case, we can create a pure virtual function named calculateArea() in the Shape. Since it's a pure virtual function, all derived classes Triangle, Square and Circle must include the calculateArea() function with implementation.

A pure virtual function doesn't have the function body and it must end with = 0. For example,

class Shape {

public:

// creating a pure virtual function

virtual void calculateArea() = 0;

};

**Note:** The = 0 syntax doesn't mean we are assigning 0 to the function. It's just the way we define pure virtual functions.

**Abstract Class**

A class that contains a pure virtual function is known as an abstract class. In the above example, the class Shape is an abstract class.

We cannot create objects of an abstract class. However, we can derive classes from them, and use their data members and member functions (except pure virtual functions).

**Example: C++ Abstract Class and Pure Virtual Function**

// C++ program to calculate the area of a square and a circle

#include <iostream>

using namespace std;

// Abstract class

class Shape {

protected:

float dimension;

public:

void getDimension()

{

cin >> dimension;

}

// pure virtual Function

virtual float calculateArea() = 0;

};

// Derived class

class Square : public Shape {

public:

float calculateArea() {

return dimension \* dimension;

}

};

// Derived class

class Circle : public Shape {

public:

float calculateArea() {

return 3.14 \* dimension \* dimension;

}

};

int main() {

Square square;

Circle circle;

cout << "Enter the length of the square: ";

square.getDimension();

cout << "Area of square: " << square.calculateArea() << endl;

cout << "\nEnter radius of the circle: ";

circle.getDimension();

cout << "Area of circle: " << circle.calculateArea() << endl;

return 0;

}

**Output**

Enter the length of the square: 4

Area of square: 16

Enter radius of the circle: 5

Area of circle: 78.5

In this program, virtual float calculateArea() = 0; inside the Shape class is a pure virtual function.

That's why we must provide the implementation of calculateArea() in both of our derived classes, or else we will get an error.