Object Oriented Programming Wiki

Tuesday, April 25, 2023 10:23 AM

This wiki provides information about syntax terms and usage as they apply to object oriented programming.
The programming languages are C++ and python.

Table of Contents

Tuesday, April 25, 2023 10:25 AM

OOP Wiki Terms

Abstraction

Tuesday, April 25, 2023 10:27 AM

TOC

Abstraction is the concept of taking complex functional methods and packaging them so the details of the underlying processes are hidden while providing an interface for the user to run the processes that is simple and intuitive.

The functions and methods of a class can be utilized by other programmers or users by simply instantiating a class.

Example code:

```
class Shape {
   public:
      virtual float area() = 0;
};

class Circle : public Shape {
   private:
      float radius;
   public:
      Circle(float r) {
         radius = r;
      }
      float area() {
         return 3.14 * radius * radius;
      }
};
```

In this example, the Shape class is an abstract class that defines the area() method without providing an implementation. The Circle class inherits from Shape and provides its own implementation of the area() method. This allows users to create Circle objects without knowing the details of how the area is calculated.

Attributes/properties

Tuesday, April 25, 2023 10:27 AM

TOC

An attribute, also called a data member, is a variable that holds data associated with an object of a class.

Example code:

```
class Car {
   public:
      string make;
      string model;
      int year;
};
```

In this example, make, model, and year are attributes of the Car class.

Class

Tuesday, April 25, 2023 10:27 AM

TOC

A class is a blueprint or template for creating objects. It defines the attributes and methods that objects of that class will have.

Example Code:

```
class Animal {
public:
   std::string name;
   int age;
   void speak() {
      std::cout << "Hello, my name is " << name << " and I am " << age << " years old."
<< std::endl;
   }
};
```

Class variable

Tuesday, April 25, 2023 10:27 AM

TOC

Class Variable/Static Variable

A class variable is a variable that is shared by all instances of a class. It is declared using the static keyword.

Example Code

```
class Animal {
public:
   static int count;
   std::string name;
   int age;
   Animal(std::string n, int a) : name(n), age(a) {
      count++;
   }
};
int Animal::count = 0;
```

Composition

Tuesday, April 25, 2023 10:27 AM

TOC

Composition is a way of creating complex objects by combining simpler objects. In C++, this is done by including objects of one class within another class.

Example Code:

```
class Engine {
public:
   void start() {
      std::cout << "Engine started." << std::endl;
   }
};
class Car {
public:
   Engine engine;
   void start() {
      engine.start();
      std::cout << "Car started." << std::endl;
   }
};
```

Constructor

Tuesday, April 25, 2023 10:28 AM

TOC

A constructor is a special method that is called when an object is created. It is used to initialize the object's member variables. A constructor will be the same name as the class and the function is defined without a return type.

Example Code

```
class Animal {
public:
   std::string name;
   int age;
   Animal(std::string n, int a) : name(n), age(a) {
      std::cout << "An animal has been created." << std::endl;
   }
};
```

In this example, an instance of the Animal class is created using the Animal (string n, int a) constructor function.

Encapsulation

Tuesday, April 25, 2023 10:28 AM

TOC

Encapsulation is the practice of hiding the internal details of a class from outside code. It is achieved by making member variables private and providing public methods to access and modify them.

Example code:

```
class BankAccount {
private:
   double balance;
public:
   void deposit(double amount) {
      balance += amount;
   }
   double getBalance() {
      return balance;
   }
};
```

Friend

Tuesday, April 25, 2023 10:28 AM

TOC

A friend is a function or class that is given access to the private members of another class.

Example Code

```
class Animal {
private:
   int age;
   friend void setAge(Animal& animal, int age);
public:
   std::string name;
   Animal(std::string n, int a) : name(n), age(a) {}
   void speak() {
      std::cout << "Hello, my name is " << name << " and I am " << age << " years old." << std::endl;
   }
};
void setAge(Animal& animal, int age) {
   animal.age = age;
}
```

Inheritance

Tuesday, April 25, 2023 10:28 AM

TOC

Inheritance is a way of creating a new class by deriving from an existing class. The new class, called the derived class, inherits the attributes and methods of the existing class, called the base class.

Example Code:

```
class Animal {
public:
   std::string name;
   int age;
   void speak() {
      std::cout << "Hello, my name is " << name << " and I am " << age << " years old." << std::endl;
   }
};
class Cat : public Animal {
public:
   void purr() {
      std::cout << name << " is purring." << std::endl;
   }
};
```

Instance variable

Tuesday, April 25, 2023 10:28 AM

TOC

An instance variable, also called an instance field or instance member, is an attribute that belongs to a specific instance of a class.

Example Code:

```
class Car {
   public:
      string make;
      string model;
      int year;
      Car(string m, string mdl, int y) {
         make = m;
         model = mdl;
         year = y;
      }
};

Car myCar("Toyota", "Camry", 2020);
```

In this example, make, model, and year are instance variables of the Car class. The values assigned to these variables when the myCar object is created are specific to that instance of the Car class.

Member variable

Tuesday, April 25, 2023 10:28 AM

TOC

A member variable, also called a class variable, is a variable that belongs to a class as a whole rather than to a specific instance of the class. Member variables are declared with the static keyword.

```
class Car {
   public:
      static int count;
      string make;
      string model;
      int year;
      Car(string m, string mdl, int y) {
         make = m;
         model = mdl;
         year = y;
         count++;
      }
};

int Car::count = 0;

Car car1("Toyota", "Camry", 2020);
Car car2("Honda", "Accord", 2019);

cout << Car::count; // Output: 2
```

In this example, count is a member variable of the Car class. It is declared as static, so there is only one instance of it for the entire Car class. Each time a new Car object is created, the constructor increments the count variable. The output shows that two Car objects have been created.

Method

Tuesday, April 25, 2023 10:28 AM

TOC

A method is a function that is associated with a class and can be called on objects of that class. It can access and modify the data members of the class.

Example Code:

```
class Car {
   public:
      string make;
      string model;
      int year;
      void printInfo() {
         cout << make << " " << model << " " << year << endl;
      }
};

Car myCar;
myCar.make = "Toyota";
myCar.model =
```

Multiple inheritance

Tuesday, April 25, 2023 10:28 AM

TOC

Multiple inheritance is a feature of some OOP languages that allows a class to inherit from more than one base class. This can lead to the Diamond problem where ambiguity arises due to multiple implementations of a method from different base classes.

Example Code:

```
class Vehicle {
   public:
      virtual void move() = 0;
};

class Car : public Vehicle {
   // Implementation
};

class Boat : public Vehicle {
   // Implementation
};

class AmphibiousVehicle : public Car, public Boat {
   // Implementation
};
```

Object

Tuesday, April 25, 2023 10:28 AM

TOC

An object is an instance of a class. It has its own unique state and behavior, and can access the attributes and methods of its class.

Example code:

```
Car myCar;
myCar.make = "Toyota";
myCar.model = "Corolla";
myCar.year = 2021;
```

Overloading

Tuesday, April 25, 2023 10:28 AM

TOC

Overloading is the ability to define multiple functions with the same name but different parameters. This allows for more flexibility in the use of the function.

Example Code:

```
class Math {
   public:
      int add(int a, int b) {
         return a + b;
      }
      double add(double a, double b) {
         return a + b;
      }
};
```

Overriding

Tuesday, April 25, 2023 10:37 AM

TOC

Overriding is a feature of object-oriented programming (OOP) that allows a subclass or child class to provide a specific implementation of a method that is already provided by its parent class or superclass. This means that when the method is called on an object of the subclass, the implementation provided by the subclass will be executed, rather than the implementation provided by the superclass. Overriding is an important concept in inheritance.

Example Code:

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

// Base class
class Shape {
  public:
    virtual void draw() {
      cout << "Drawing Shape" << endl;
    }
};

// Derived class
class Circle: public Shape {
  public:
    void draw() {
      cout << "Drawing Circle" << endl;
    }
};

int main(void) {
  Shape *shape;
  Circle circle;

  // calling draw method of Shape using pointer to Circle object
  shape = &circle;
  shape->draw(); // Will call the draw method of Circle class

  return 0;
}
```

Polymorphism

Tuesday, April 25, 2023 10:28 AM

TOC

Polymorphism is the ability of objects of different classes to be treated as if they were objects of the same class. This is achieved through inheritance and virtual functions.

Example Code:

```
class Vehicle {
   public:
      virtual void move() = 0;
};

class Car : public Vehicle {
   public:
      void move() {
         // Implementation
      }
};

class Boat : public Vehicle {
   public:
      void move() {
         // Implementation
      }
};

void makeMove(Vehicle& v) {
   v.move();
}

Car myCar;
Boat myBoat;
makeMove(myCar);
makeMove(myBoat);
```

Public/Private/Protected

Tuesday, April 25, 2023 10:28 AM

TOC

In object-oriented programming, access modifiers such as public, private, and protected are used to control the visibility of class members (i.e., variables and methods) from outside the class. These modifiers determine how accessible a class member is to other parts of the program.

- Public: Public members are accessible from anywhere in the program, including from outside the class. They are often used for methods and variables that need to be accessed from other classes.
- Private: Private members are only accessible within the class they are defined in. They cannot be accessed from outside the class, including from derived classes. They are often used for variables and methods that should only be used internally by the class.
- Protected: Protected members are accessible within the class they are defined in and any derived classes. They cannot be accessed from outside the class or its derived classes. They are often used for variables and methods that need to be accessed by derived classes but should not be public.

Example Code:

```
class MyClass {
public:
   int publicVar;
void publicMethod() {
      // Code for public method
   }
private:
   int privateVar;
void privateMethod() {
      // Code for private method
   }
protected:
   int protectedVar;
void protectedMethod() {
      // Code for protected method
   }
```

```
};
int main() {
    MyClass obj;
    obj.publicVar = 1;     // Accessing public member
    obj.publicMethod();    // Calling public method
// obj.privateVar = 2; // Error: Cannot access private member
    // obj.privateMethod();// Error: Cannot access private member
// obj.protectedVar = 3; // Error: Cannot access protected member
    // obj.protectedMethod();// Error: Cannot access protected member
return 0;
}
```

Static

Tuesday, April 25, 2023 10:28 AM

TOC

A static variable is a variable that is associated with the class rather than with individual objects of the class. It can be accessed without creating an instance of the class, and its value is shared across all objects of the class.
Here is an example code snippet in C++ that demonstrates the usage of a static variable:

Example Code:

```
class MyClass {
public:
   static int count;     // Declaration of static variable

   MyClass() {
      count++;         // Increment count each time an object is created
   }
};

int MyClass::count = 0;   // Definition of static variable

int main() {
   MyClass obj1;
   MyClass obj2;
   MyClass obj3;

   std::cout << "Number of objects created: " << MyClass::count << std::endl; // Accessing static variable

   return 0;
}
```

Virtual

Tuesday, April 25, 2023 10:29 AM

TOC

A virtual function is a function that is declared in a base class and can be overridden in derived classes. It allows derived classes to provide their own implementation of the function, which is called instead of the base class implementation when the function is called on an object of the derived class.

Example Code:

```
class Shape {
public:
   virtual void draw() {
      std::cout << "Drawing a shape" << std::endl;
   }
};

class Circle : public Shape {
public:
   void draw() override {
      std::cout << "Drawing a circle" << std::endl;
   }
};

int main() {
   Shape* shape = new Circle();  // Upcasting
   shape->draw();              // Calling virtual function

   delete shape;

   return 0;
}
```