What is Object in C++?

In C++, an object is an instance of a class, which serves as a blueprint or template for creating objects.

- A class in C++ is a user-defined data type that acts as a blueprint for objects.
- When a class is instantiated, it becomes an object, meaning it allocates memory and creates an instance based on the blueprint defined by the class.
- Objects allow you to model real-world entities within your C++ programs, making C++ an object-oriented language.
- While a class is static in nature (it defines properties and methods), an object is dynamic, representing a particular instance of that class with its unique data.

An object in C++ is a concrete realization of a class, allowing you to create multiple instances with distinct properties and behaviors.

What is Class in C++?

A class in C++ serves as a foundational element for object-oriented programming.

- **User-Defined Data Type:** A class is a user-defined data type in C++. It allows programmers to encapsulate data and behaviors into a single unit.
- **Blueprint for Objects**: Think of a class as a blueprint. When you create an object, you're essentially creating an instance of that blueprint, inheriting its attributes and behaviors.
- **Contains Data and Functions:** Within a class, you can define both data members (variables) and member functions (methods). These methods define the operations that can be performed on the data.
- **Access Control:** The members of a class have their access controlled, ensuring that they can be accessed, modified, or used based on specified permissions.

A class provides structure and organization to code, enabling the creation of objects with defined properties and behaviors.

Class in C++ Examples

Below are some basic examples demonstrating the use of classes in C++:

1. Simple Class Example:

```
#include <iostream>

using namespace std;

// Class definition

class Rectangle {

private:

    double length;

    double width;

public:

    // Constructor to initialize length and width

    Rectangle(double l, double w) : length(l), width(w) {}

    // Method to compute area

    double area() {

        return length * width;

    }

};

int main() {

    // Create an object of the Rectangle class

    Rectangle rect(5.0, 3.0);

    // Compute and display the area of the rectangle

    cout << “Area of the rectangle: ” << rect.area() << endl;

    return 0;
}
```

2. Class with Constructor and Destructor:

```
#include <iostream>

using namespace std;

// Class definition

class Circle {

private:

 double radius;

public:

 // Constructor to initialize radius

 Circle(double r) : radius(r) {

 cout << “Circle object is created with radius: ” << radius << endl;

 }

 // Method to compute area

 double area() {

 return 3.14 * radius * radius;

 }

 // Destructor to display a message when the object is destroyed

 ~Circle() {

 cout << “Circle object with radius ” << radius << ” is destroyed.” << endl;

 }

};

int main() {
 // Create an object of the Circle class
 Circle circle(7.0);
 // Compute and display the area of the circle
 cout << “Area of the circle: ” << circle.area() << endl;
 return 0;
}
```

3. Class with Member Functions and Encapsulation:

```
#include <iostream>
using namespace std;
// Class definition
class BankAccount {
private:
 string accountNumber;
 double balance;
public:
 // Constructor to initialize account details
 BankAccount(string accNum, double bal) : accountNumber(accNum), balance(bal) {}
 // Method to deposit money
 void deposit(double amount) {
 balance += amount;
 cout << “Deposit successful. Current balance: ” << balance << endl;
 }
 // Method to withdraw money
 void withdraw(double amount) {

 if (amount > balance) {

 cout << “Insufficient funds!” << endl;

 } else {

 balance -= amount;

 cout << “Withdrawal successful. Current balance: ” << balance << endl;

 }

 }

};

int main() {

 // Create an object of the BankAccount class

 BankAccount account(“123456789”, 5000);

 // Deposit and withdraw money

 account.deposit(2000);

 account.withdraw(3000);

 return 0;

}
```

Syntax of Class in C++

The syntax of a basic class in C++ includes the class keyword, followed by the class name and its body, which contains the class members and member functions. Here’s a simple breakdown:

```
class ClassName {

   // Access Specifiers

private:
   // Private members and member functions

protected:

   // Protected members and member functions

public:

   // Public members and member functions

   // Constructor (if any)

   ClassName() {

       // Constructor body

   }

   // Destructor (if any)

   ~ClassName() {

       // Destructor body

   }

   // Member Functions (methods)

   returnType methodName(parameters) {

       // Function body

   }

   // Member Variables (attributes or properties)

   dataType variableName;

};
```

- private, protected, and public are the access specifiers that define the visibility of the class members.
- Inside the class, you can have member functions, member variables, constructors, and destructors.
- The member functions can perform operations on the member variables.

- The constructor is a special member function that gets called when an object is created.
- The destructor is a special member function that gets called when an object is destroyed.

Here’s a simple example for better understanding:

```
// Defining a class named ‘Person’

class Person {

private:          // Access specifier

    string name;  // Private member variable

public:           // Access specifier

    // Constructor with parameters

    Person(string n) {

        name = n;  // Initializing member variable

    }

    // Public member function to display the name

    void display() {

        cout << “Name: ” << name << endl;

    }

};

int main() {

    // Creating an object of class ‘Person’

    Person person1(“John”);

    // Accessing the member function using the object

    person1.display();
```

```
   return 0;

}
```

In this example, we defined a class Person with a private member variable name, a constructor to initialize the name, and a public member function display() to display the name.

Accessing Data Members

In object-oriented programming with languages like C++, data members of a class can be accessed in various ways, depending on their access specifiers (public, private, protected). Here's how you can access data members in C++:

1) Public Data Members:

Public data members are accessible from outside the class using the object of the class.

```
class MyClass {

public:

 int publicVar;

};

int main() {

 MyClass obj;

 obj.publicVar = 10; // Accessing public data member

 return 0;

}
```

2) Private Data Members:

Private data members cannot be accessed directly from outside the class. You need to use public member functions (getters and setters) to access or modify them.

```
class MyClass {

private:
 int privateVar;
public:
 void setPrivateVar(int val) {
 privateVar = val;
 }
 int getPrivateVar() {
 return privateVar;
 }
};
int main() {
 MyClass obj;
 obj.setPrivateVar(20); // Setting private data member using public member function
 int value = obj.getPrivateVar(); // Getting private data member using public member
function
 return 0;
}
```

3) Protected Data Members:

Protected data members are accessible within the class itself and in derived classes.

```
class Base {
protected:
 int protectedVar;
};
class Derived : public Base {
```

```
public:
 void setProtectedVar(int val) {
 protectedVar = val;
 }
 int getProtectedVar() {
 return protectedVar;
 }
};
int main() {
 Derived obj;
 obj.setProtectedVar(30); // Setting protected data member in derived class
 int value = obj.getProtectedVar(); // Getting protected data member in derived class
 return 0;
}
```

Constructors and Destructors

Constructors and destructors are fundamental concepts in C++ that facilitate the initialization and cleanup of objects, respectively. Here's a detailed explanation:

Constructors:

Definition:

A constructor in C++ is a special member function of a class that gets automatically invoked whenever an object is created. Its primary purpose is to initialize the object's state or allocate resources.

Types of Constructors:

1) Default Constructor: It doesn’t take any arguments. If you don’t define one, the compiler provides a default constructor for you.

```
class MyClass {
public:
 MyClass() { // Default Constructor
 // Initialization code here
 }
};
```

2) Parameterized Constructor: It accepts parameters to initialize the object with specific values.

```
class MyClass {
public:
 int value;
 MyClass(int v) : value(v) { // Parameterized Constructor
 // Initialization code here
 }
};
```

3) Copy Constructor: It initializes an object using another object of the same class.

```
class MyClass {
public:
 int value;
 MyClass(const MyClass& obj) { // Copy Constructor
 value = obj.value;
 }
};
```

Destructors:

Definition:

A destructor is a special member function that gets invoked automatically when an object goes out of scope or is explicitly deleted. Its primary purpose is to release resources or perform cleanup tasks.

Syntax:

The destructor is identified by the tilde (~) symbol followed by the class name. It cannot have parameters or a return type.

```
class MyClass {

public:

 ~MyClass() { // Destructor

 // Cleanup code here

 }

};
```

Usage:

Destructors are crucial when an object acquires resources during its lifetime, like dynamic memory allocation. Without proper cleanup in the destructor, it can lead to memory leaks or resource leaks.

Example with Dynamic Memory:

```
class DynamicArray {

private:

 int* arr;

public:

 DynamicArray(int size) { // Constructor for dynamic memory allocation
 arr = new int[size];

 }

 ~DynamicArray() { // Destructor to release allocated memory

 delete[] arr;

 }

};
```

Note:

- If a class doesn't provide a destructor, the compiler generates a default one. However, if a class acquires resources like dynamic memory, it's essential to provide a destructor to release those resources.
- Constructors and destructors are essential for resource management, ensuring that objects are initialized correctly and resources are freed when they're no longer needed.

Class and Object in C++ Examples

Below are examples that demonstrate the concepts of classes and objects in C++:

Example 1: Simple Class and Object

```
#include <iostream>

using namespace std;

// Defining a simple class named 'Rectangle'

class Rectangle {

private:

 int length;

 int width;

public:

 // Constructor to initialize length and width
 Rectangle(int l, int w) {

 length = l;

 width = w;

 }

 // Public member function to calculate area

 int area() {

 return length * width;

 }

};

int main() {

 // Creating an object of class ‘Rectangle’

 Rectangle rect(4, 5); // Length = 4, Width = 5

 // Calculating and displaying area using the object

 cout << “Area of rectangle: ” << rect.area() << ” square units” << endl;

 return 0;

}
```

Example 2: Class with Constructor and Destructor

```
#include <iostream>

using namespace std;

// Defining a class named ‘Student’

class Student {

private:

 string name;
public:

 // Constructor to initialize name

 Student(string n) {

 name = n;

 cout << “Student ” << name << ” is created.” << endl;

 }

 // Destructor

 ~Student() {

 cout << “Student ” << name << ” is destroyed.” << endl;

 }

};

int main() {

 // Creating objects of class ‘Student’

 Student s1(“John”);

 Student s2(“Doe”);

 return 0;

}
```

In these examples:

- We defined a class using the class keyword, encapsulating data (attributes) and functions (methods) within it.
- Objects (rect, s1, s2) of these classes are created in the main() function.
- We use the dot operator (.) to access the members (methods and variables) of the objects.
- The constructor (Rectangle(int l, int w) and Student(string n)) is a special member function that gets invoked when an object is created.
- The destructor (~Student()) is a special member function that gets invoked when an object is destroyed, typically when it goes out of scope.

These examples illustrate the fundamental concepts of classes and objects in C++.

C++ Classes and Objects Exercises

Here are some exercises that you can use to practice working with classes and objects in C++:

Exercise 1: Book Class

Create a Book class with attributes such as title, author, and ISBN. Include methods to display book details and set book details.

Exercise 2: Bank Account Class

Design a BankAccount class with methods like deposit, withdraw, and getBalance. Ensure you have a constructor to set an initial balance.

Exercise 3: Student Class

Create a Student class with attributes like name, rollNumber, and marks. Implement methods to calculate the grade based on the marks.

Exercise 4: Car Class

Develop a Car class that contains attributes like make, model, year, and speed. Include methods like accelerate and brake.

Exercise 5: Complex Number Class

Design a ComplexNumber class with attributes for the real and imaginary parts. Implement methods to add, subtract, multiply, and display the complex numbers.

Exercise 6: Library System

Develop a Library class that contains a collection of books (Book objects). Implement methods to add a book, remove a book, and display all books.

Exercise 7: Employee Class

Create an Employee class with attributes like name, id, designation, and salary. Include methods to give a salary hike based on performance.

Exercise 8: Polygon Class

Design a Polygon class with attributes for the number of sides and length of each side. Implement methods to calculate the perimeter and area.

Exercise 9: Time Class

Develop a Time class to represent time in hours, minutes, and seconds. Include methods to add two times together and display the result.

Exercise 10: Mobile Phone Class

Create a MobilePhone class with attributes like brand, model, price, and batteryLife. Implement methods to check if the phone is affordable based on a given budget.

To solve these exercises:

- Start by designing the class structure with attributes and methods.
- Implement constructors to initialize the objects.
- Define member functions to perform various operations on objects.
- Test your classes by creating objects in the main() function and calling the methods.

These exercises will help you understand how to design and implement classes and objects in C++, enhancing your object-oriented programming skills.

Programming Questions on Classes and Objects in C++

Here are some programming questions on classes and objects in C++ along with their respective answers:

1. **Question:** Create a Car class with attributes brand, model, and year. Provide methods to set and display these attributes.

Answer:

```
#include <iostream>

#include <string>

using namespace std;
```

```
class Car {

private:

 string brand;

 string model;

 int year;

public:

 // Constructor

 Car(string b, string m, int y) : brand(b), model(m), year(y) {}

 // Setter method

 void setDetails(string b, string m, int y) {

 brand = b;

 model = m;

 year = y;

 }

 // Display method

 void displayDetails() {

 cout << “Brand: ” << brand << “, Model: ” << model << “, Year: ” << year << endl;

 }

};

int main() {

 Car myCar(“Toyota”, “Camry”, 2020);

 myCar.displayDetails();

 // Update details
 myCar.setDetails(“Honda”, “Civic”, 2022);

 myCar.displayDetails();

 return 0;

}
```

2. **Question:** Create a Rectangle class with attributes length and width. Include methods to calculate the area and perimeter of the rectangle.

Answer:

```
#include <iostream>

using namespace std;

class Rectangle {

private:

 double length;

 double width;

public:

 // Constructor

 Rectangle(double l, double w) : length(l), width(w) {}

 // Method to calculate area

 double calculateArea() {

 return length * width;

 }

 // Method to calculate perimeter

 double calculatePerimeter() {

 return 2 * (length + width);
 }
};
int main() {
 Rectangle rect(5.0, 3.0);
 cout << "Area of Rectangle: " << rect.calculateArea() << endl;
 cout << "Perimeter of Rectangle: " << rect.calculatePerimeter() << endl;
 return 0;
}
```

3. **Question:** Define a Book class with attributes title, author, and isbn. Provide methods to set and display these attributes.

Answer:

```
#include <iostream>
#include <string>
using namespace std;
class Book {
private:
 string title;
 string author;
 string isbn;
public:
 // Constructor
 Book(string t, string a, string i) : title(t), author(a), isbn(i) {}
 // Setter method
 void setDetails(string t, string a, string i) {

 title = t;

 author = a;

 isbn = i;

 }

 // Display method

 void displayDetails() {

 cout << “Title: ” << title << “, Author: ” << author << “, ISBN: ” << isbn << endl;

 }

};

int main() {

 Book myBook(“The Alchemist”, “Paulo Coelho”, “9780062315007”);

 myBook.displayDetails();

 // Update details

 myBook.setDetails(“1984”, “George Orwell”, “9780451524935”);

 myBook.displayDetails();

 return 0;

}
```

4. **Question:** Define a Student class with attributes name, rollNumber, and grade. Provide a method to display the student details.

Solution:

```
#include <iostream>

#include <string>
```

```
using namespace std;

class Student {

private:

 string name;

 int rollNumber;

 char grade;

public:

 Student(string n, int r, char g) : name(n), rollNumber(r), grade(g) {}

 void displayDetails() {

 cout << “Name: ” << name << “, Roll Number: ” << rollNumber << “, Grade: ” << grade
<< endl;

 }

};

int main() {

 Student student1(“John Doe”, 101, ‘A’);

 student1.displayDetails();

 return 0;

}
```

5. **Question:** Create a BankAccount class with attributes accountNumber, accountHolder, and balance. Implement methods to deposit and withdraw money. Ensure to display an appropriate message if a withdrawal amount exceeds the available balance.

Solution:

```
#include <iostream>

#include <string>
```

```
using namespace std;

class BankAccount {

private:

 string accountNumber;

 string accountHolder;

 double balance;

public:

 BankAccount(string num, string holder, double bal) : accountNumber(num),
accountHolder(holder), balance(bal) {}

 void deposit(double amount) {

 balance += amount;

 cout << “Amount deposited. Current Balance: ” << balance << endl;

 }

 void withdraw(double amount) {

 if (amount > balance) {

 cout << “Insufficient balance!” << endl;

 } else {

 balance -= amount;

 cout << “Amount withdrawn. Current Balance: ” << balance << endl;

 }

 }

 void displayBalance() {

 cout << “Account Number: ” << accountNumber << “, Holder: ” << accountHolder << “,
Balance: ” << balance << endl;
```

```
 }
};
int main() {
 BankAccount account(“123456789”, “Alice”, 5000);
 account.displayBalance();
 account.deposit(2000);
 account.withdraw(1000);
 account.withdraw(7000); // This will display an “Insufficient balance!” message.
 return 0;
}
```

6. **Question:** Design a Library class with attributes bookTitle, author, and isAvailable. Include methods to lend and return books. Display appropriate messages based on book availability.

Solution:

```
#include <iostream>
#include <string>
using namespace std;
class Library {
private:
 string bookTitle;
 string author;
 bool isAvailable;
public:
 Library(string title, string auth) : bookTitle(title), author(auth), isAvailable(true) {}
 void lendBook() {

 if (isAvailable) {

 isAvailable = false;

 cout << “Book has been lent.” << endl;

 } else {

 cout << “Sorry, the book is not available at the moment.” << endl;

 }

 }

 void returnBook() {

 if (!isAvailable) {

 isAvailable = true;

 cout << “Book has been returned.” << endl;

 } else {

 cout << “This book was not lent out.” << endl;

 }

 }

};

int main() {

 Library book(“The Great Gatsby”, “F. Scott Fitzgerald”);

 book.lendBook();

 book.returnBook();

 book.lendBook(); // This will display “Sorry, the book is not available at the moment.”

 return 0;
```

```
}
```