- Is-a vs. has-a relationships and how it relates to inheritance vs. composition as programming concepts
    - Is-a
        - relationship where one class derives from another
        - Indicates inheritance; a subclass is a specialized version of the superclass
    - Has-a
        - relationship where one class owns or uses another
        - Indicates composition; a class contains an object of another class
- Inheriting multiple classes (with an example)
    - class inherits from more than one base class

```
class Engine {
   // Engine class details
};
class Wheels {
   // Wheels class details
};
class Car : public Engine, public Wheels {
   // Car inherits from Engine and Wheels
};
```

- compile-time vs. runtime polymorphism
    - Compile time
        - Function overloading, operator overloading
    - Runtime
        - Function overriding with base class pointers
- compile-time polymorphism: function and operator overloading (with examples)
    - Function overloading
        - Multiple functions with the same name but different parameters

```
void print(int i) { cout << i; }
void print(double d) { cout << d; }
void print(string s) { cout << s; }
```

- Operator overloading
  - Define custom behavior for operators

```
class Complex {
   double real, imag;
public:
   Complex operator + (const Complex& obj) {
      return Complex(real + obj.real, imag + obj.imag);
   }
};
```

- runtime polymorphism:
  - derived/base class pointer conversion (with an example)
    - Base class pointers can point to derived class objects

```
class Base { public: virtual void display() { cout << "Base"; } };
class Derived : public Base { public: void display() override { cout << "Derived"; } };
Base* b = new Derived();
b->display(); // Outputs "Derived"
```

  - virtual functions (with an example)
    - Functions declared in the base class and overridden in derived classes

```
class Animal { public: virtual void sound() { cout << "Generic Sound"; } };
class Dog : public Animal { public: void sound() override { cout << "Bark"; } };
```

  - override keyword (with an example)
    - Ensures the function overrides a base class function

```
class Base { public: virtual void foo() { ... } };
class Derived : public Base { public: void foo() override { ... } };
```

  - virtual class (with an example)
    - Used to solve ambiguity in inheritance

```
class A { ... };
class B : virtual public A { ... };
class C : virtual public A { ... };
class D : public B, public C { ... };
```

- abstract vs. concrete classes (with examples)
  - Abstract
    - Contains at least one pure virtual function

```
class Shape {
   virtual void draw() = 0; // Pure virtual
};
```

  - Concrete
    - Fully implemented class

```
class Circle : public Shape {
   void draw() override { cout << "Drawing Circle"; }
};
```