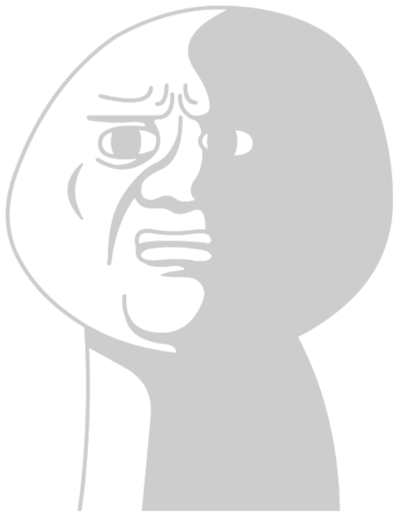
Object-oriented Programming

Inheritance

Object Relationship

- **is-a** relationship
- **has-a** relationship



Object Relationship

- **is-a** relationship
- **has-a** relationship

PRO TIP Use common sense

has-a Relationship

- In a *has-a* relationship, an object contains one or more objects of other classes as members

A car ***has a*** steering wheel ✓
An office ***has a*** department ✓

A car ***is a*** steering wheel ✗

(doesn’t make any sense so it must not be a correct relationship)

has-a Relationship

```
class Office
{
  Department d;
  // other members
};
```

is-a Relationship

- Sometimes, one class ***is*** an extension of another class

A car ***is a*** vehicle ✓
Cricket ***is a*** sport ✓

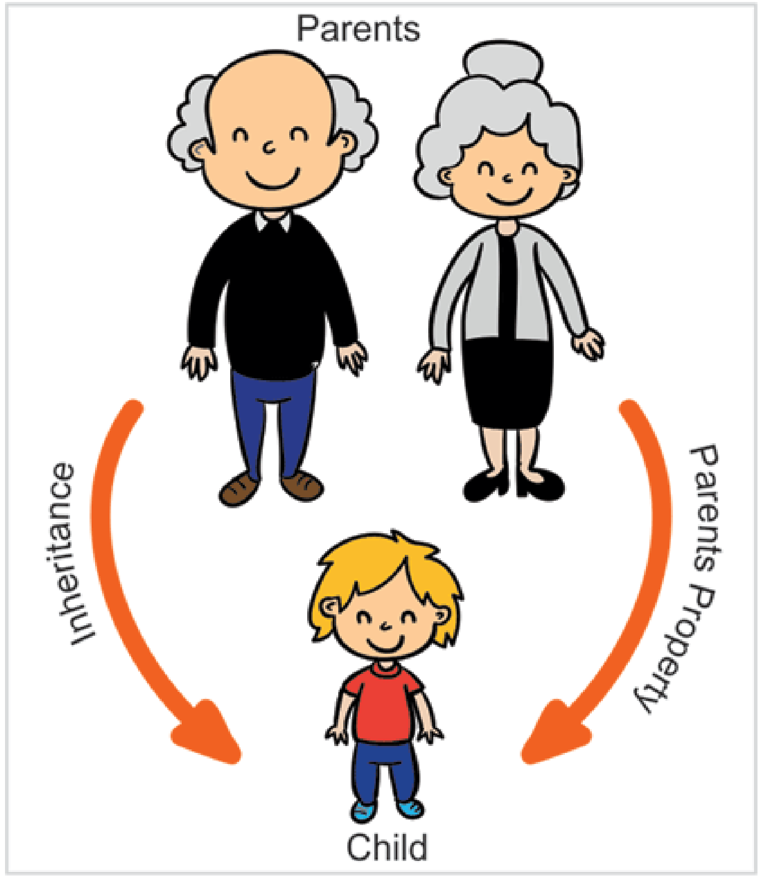
A car ***has a*** vehicle ✗
(doesn't make any sense so it must not be a correct relationship)

is-a Relationship

- The extended (or child) class contains all the features of its base (or parent) class, and may additionally have some unique features of its own
- The key idea behind **Inheritance**

Next Lecture

- Inheritance!



Inheritance

- A form of software reusability where a class *inherits* an existing class' behavior and enhances it by adding more functionalities

- The existing class is called base class (or sometimes super class) and the new class is referred to as derived class (or sometimes sub class)

Example

- The *is-a* relationship represents inheritance

- The *car* is a vehicle, so any attributes and behaviors of a *vehicle* are also attributes and behaviors of a *car*

Example

```
class Vehicle
{
        // data members of base class
}

class Car: public Vehicle
{
        //data members of derived class
}
```

Base & Derived Classes

- Every derived-class object is also an object of its base class, and one base class can have many derived classes
- A derived class can access all non-private members of its base class

Visibility of Base Class Members

- A derived class can use the access modifiers public, protected or private to restrict access to its base class members

- In all situations, a derived class can never access private members of its base class

Public Inheritance

- The use of access modifier public in derived class header

Example:

```
class myDerived: public myBase
{
        // derived class members
}
```

Public Inheritance

- In public inheritance:
  - The **public** members of a base class are treated as **public** members of the derived class by other classes further down the hierarchy
  - The **protected** members of a base class are treated as **protected** members of the derived class by other classes further down the hierarchy

Public Inheritance

```
class Parent
{
        private:        int a;
        public:int b;
        protected:      int c;
}


class Child: public Parent
{
   // can never access a directly
   // can access b & c directly
}
```

```
class GrandChild: public Child
{
   // can never access a directly
   // can access b directly
    // can access c directly
}
```

Protected Inheritance

- The use of access modifier protected in derived class header

Example:

```
class myDerived: protected myBase
{
        // derived class members
}
```

Protected Inheritance

- In protected inheritance:
    - The **public** members of a base class are treated as **protected** members of the derived class by other classes further down the hierarchy

    - The **protected** members of a base class are treated as **protected** members of the derived class by other classes further down the hierarchy

Protected Inheritance

```
class Parent
{
        private:        int a;
        public:int b;
        protected:      int c;
}


class Child: protected Parent
{
   // can never access a directly
   // can access b & c directly
}
```

```
class GrandChild: public Child
{
   // can never access a directly
   // can access b directly
    // can access c directly
}
```

Private Inheritance

- The use of access modifier private in derived class header

Example:

```
class myDerived: private myBase
{
        // derived class members
}
```

Private Inheritance

- In private inheritance:
  - All **public** & **protected** members of a base class are treated as **private** members of the derived class by other classes further down the hierarchy

  - In other words, these inherited members can be seen as *locked* and cannot be inherited further down the hierarchy

Private Inheritance

```
class Parent
{
        private:        int a;
        public:int b;
        protected:      int c;
}


class Child: private Parent
{
    // can never access a directly
    // can access b & c directly
}
```

```
class GrandChild: public Child
{
    // can never access a directly
    // cannot access b directly
    // cannot access c directly
}
```