Inheritance

Definition:

"Process of extending existing class into new class is known as inheritance"

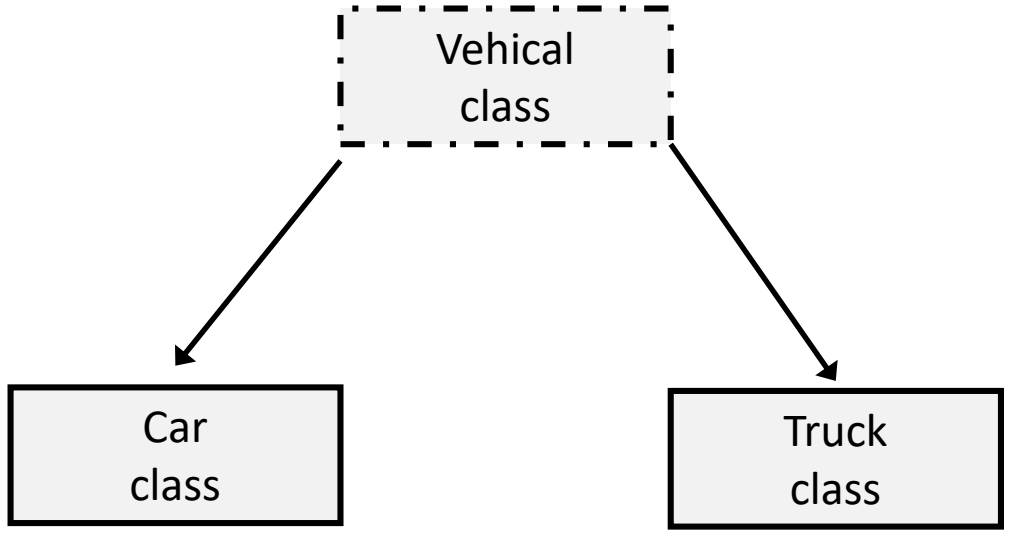
- Existing class is known as **Base Class** (or Parent Class)
- New class is known as **Derived Class** (or Child Class)

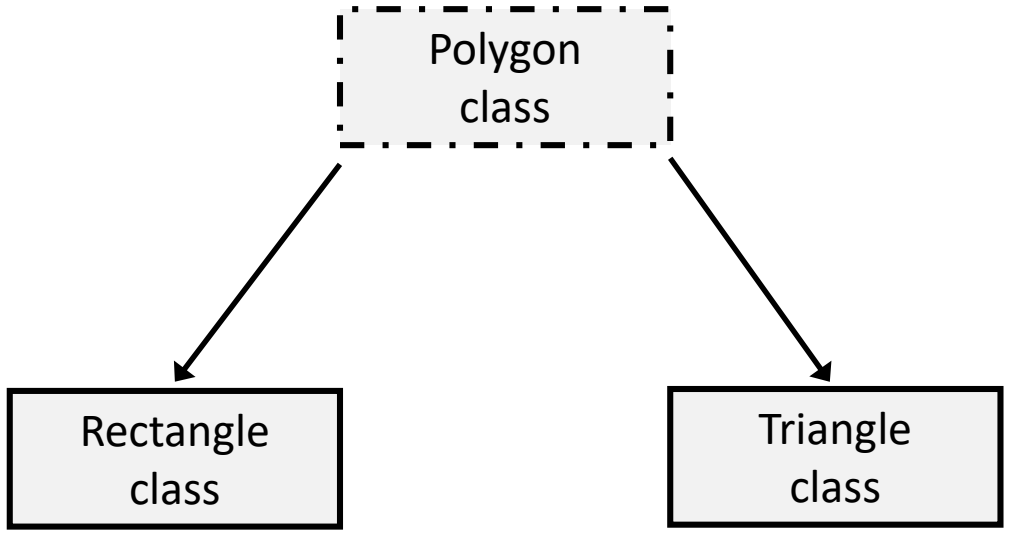
What is inherited from the base class?

Every member of a base class except:

- constructors and its destructor
- assignment operator members (operator=)
- friends
- private members

Inheritance (example)





Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

```
class Rectangle : public Polygon {
  public:
    int area()
    {
       return width * height;
    }
};
```

Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

```
class Rectangle : public Polygon {
  public:
    int area()
    {
       return width * heigh
    }
};
```

This access specifier limits the most accessible level for the members inherited from the base class

Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

Remember, we can't inherit private members

```
class Rectangle : public Polygon {
  public:
    int area()
    {
       return width * height;
    }
};
```

Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

```
class Rectangle : public Polygon {
  public:
    int area()
    {
       return width * height;
    }
};
```

```
class Triangle : public Polygon {
  public:
    int area()
    {
       return width * height / 2;
    }
};
```

Inheritance (example)

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {
 protected:
   int width, height;
 public:
   void set_values(int a, int b)
   {
      width = a; height = b;
   }
};
```

```
class Rectangle : public Polygon {
  public:
    int area()
    {
       return width * height;
    }
};


class Triangle : public Polygon {
  public:
    int area()
    {
       return width * height / 2;
    }
};
```

```
int main() {
Rectangle rect;
Triangle trgl;
rect.set_values(4, 5);
trgl.set_values(4, 5);
cout << rect.area();
cout << trgl.area();
return 0;
}
```

Multiple Inheritance

When a class is derived from more than one base class, using a comma-separated list, is called ***Multiple Inheritance***

```
class MyClass {
public:
    void myFunction() {
    cout << “Parent class“ << endl;
   }
};

// Another base class
class MyOtherClass {
public:
    void myOtherFunction() {
    cout << “Another class" << endl;
    }
};
```

```
// Derived class
class MyChildClass : public MyClass, public MyOtherClass
{
};


 int main() {
     MyChildClass myObj;
     myObj.myFunction();
     myObj.myOtherFunction();
 return 0;
 }
```

Multi-level Inheritance

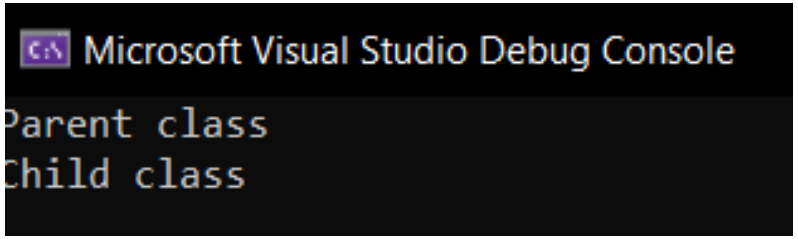
When a class is derived from a class which is also derived from another class is called ***Multilevel Inheritance***

```
class Grand_Parent {

public:
    void MyGrandParent()
    {
        cout << "Grand Parent class" << endl;
    }
};

class Parent : public Grand_Parent {

public:
    void MyParent()
    {
        cout << "Parent class" << endl;
    }
};
```

```
class Child : public Parent {
public:
    void Me()
    {
        cout << "Child class" << endl;
    }
};

int main() {
    Child myObj;
    myObj.MyGrandParent();
    myObj.MyParent();
    myObj.Me();

    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Grand Parent class
Parent class
Child class
```

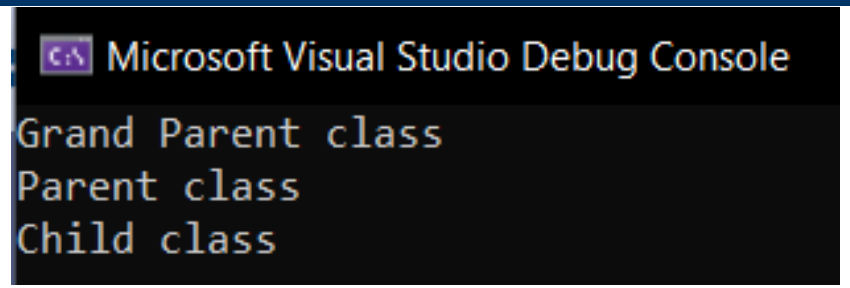
Multi-level Inheritance




```
class Grand_Parent {

public:
    void Func()
    {
        cout << "Grand Parent class" << endl;
    }
};

class Parent : public Grand_Parent {

public:
    void Func()
    {
        cout << "Parent class" << endl;
    }
};
```

```
class Child : public Parent {
public:
    void Me()
    {
        cout << "Child class" << endl;
    }
};

int main() {
    Child myObj;
    myObj.Func();
    myObj.Me();

    return 0;
}
```

Multi-level Inheritance




```
class Grand_Parent {

public:
    void Func()
    {
        cout << "Grand Parent class" << endl;
    }
};

class Parent : public Grand_Parent {

public:
    void Func()
    {
        cout << "Parent class" << endl;
    }
};
```

```
class Child : public Parent {
public:
    void Me()
    {
        cout << "Child class" << endl;
    }
};

int main() {
    Child myObj;
    myObj.Grand_Parent::Func();
    myObj.Func();
    myObj.Me();

    return 0;
}
```

Multiple Inheritance

```
class Grand_Parent {

public:
    void Func()
    {
        cout << "Grand Parent class" << endl;
    }
};

class Parent {

public:
    void Func()
    {
        cout << "Parent class" << endl;
    }
};
```

```
class Child : public Parent, public
Grand_Parent {
public:
    void Me()
    {
        cout << "Child class" << endl;
    }
};
int main() {
    Child myObj;
    myObj.Func();
    myObj.Me();

    return 0;
}
```

Error

“Pass by Value” and “Pass by Reference”

Pass by Value:

- Makes a copy in memory of the actual parameters

- Use pass by value when you are only **using** the parameter for some computation, not changing it

Pass by Reference:

- Forwards the actual parameters

- Use pass by reference when you are **changing** the parameter passed in the program

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}


int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Declaration

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}


int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Declaration

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}

int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
return 0;
}
```

Function Definition

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}

int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Definition

“Pass by Value”

Pass by Pointer ~~“Pass by Reference”~~

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Calling

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}


int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Calling

Another way for “Pass by Reference”

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

int add(int &a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Reference Variable:
Reference variable is an alias for a variable which is assigned to it.

Different from pointer:

- The reference variable can only be initialized at the time of its creation
- The reference variable returns the address of the variable preceded by the reference sign ‘&’
- The reference variable can never be reinitialized again in the program
- The reference variable can never refer to NULL

Thanks a lot

If you are taking a Nap, **wake up**.....Lecture Over