Phill-CPP-0809

```
#include <iostream>
using namespace std;
class Seat{

  private:
    int row;
    int column;
    bool isReserved;

  public:
    Seat(int r, int c){
      row=r;
      column= c;
      isReserved=false;
    }

    void reserve(){
      isReserved=true;
    }

    void cancel(){
      isReserved=false;
    }

    bool isReservedXXX(){
      return isReserved;
    }

    int getRow(){
      return row;
    }


    int getColumn(){
      return column;
    }


};

// Seat::Seat(int r, int c){
//       row=r;
//       column= c;
//       isReserved=false;
// }
int main()
{
```

```
    // Seat s[3]={
    //   Seat(1,1),
    //   Seat(2,2),
    //   Seat(3,3)
    // };
    Seat* s[9][9];

    for(int i=1; i<10; i++)
      for(int j=1; j<10; j++){
          s[i-1][j-1]= new Seat(i-1,j-1);
          // cout << i-1 <<", "<<j-1 <<endl;
      }
    cout << "Row: " <<s[1][8]->getRow()<<", Column:" <<s[1][8]->getColumn() <<endl;
    cout << "reserved:" <<s[1][8]->isReservedXXX() <<endl;

    // s.reserve();
    // cout << "reserved:" <<s.isReservedXXX() <<endl;

    // s.cancel();
    // cout << "reserved:" <<s.isReservedXXX() <<endl;

    return 0;
}
```

多重繼承 multiple inheritance

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

class Aquatic{
  public:
    void swim(){
      cout << "I can swim"<< endl;
    }
};


class Ambulatory {
  public:
    void walk(){
      cout << "I can walk" << endl;
    }

};


class Penguin: public Aquatic, public Ambulatory{
```

```
};

int main()
{
    Penguin p;
    p.swim();
    p.walk();
    return 0;
}
```

1. 多重繼承問題 : 同名時 我要繼承誰? ambiguous

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

class Base1{
  public:
    void abc(){
      cout << "I am abc from BASE1"<<endl;
    }
};

class Base2{
  public:
    void abc(){
      cout << "I am abc from BASE2"<< endl;
    }
};


class Derived: public Base1, public Base2{

};


int main()
{
    Derived d;
    d.abc();
    return 0;
}
```

解法

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

```
class Base1{
  public:
    void abc(){
      cout << "I am abc from BASE1"<<endl;
    }
};

class Base2{
  public:
    void abc(){
      cout << "I am abc from BASE2"<< endl;
    }
};


class Derived: public Base1, public Base2{
  public:
    void abcFromBase1() { Base1::abc();}
    void abcFromBase2() { Base2::abc();}
};


int main()
{
    Derived d;
    d.abcFromBase1();
    return 0;
}
```

2. 鑽石繼承(菱形繼承)

class A(aaa) → class B(aaa())→class D

class A(aaa) → class C(aaa())→class D

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

class A{
  public:


};

class B: public A{
};

class C: public A{
```

```
};

class D: public B, public C{
};


int main()
{
    D d;
    d.aaa();
    return 0;
}
```

虛擬繼承 (virtual inheritance)

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

class A{
  public:
    void aaa(){
      cout << "I am aaa()"<<endl;
    }
};

class B: public virtual A{
};

class C: public virtual A{
};

class D: public B, public C{
};


int main()
{
    D d;
    d.aaa();
    return 0;
}
```

練習

- Person

    - private → name
    - public → getName
    - Person(name)
- 繼承
    - Pitcher 投手
        - void doPitch() → I can pitch
    - Player 打擊
        - void doBat() → I can hit
    - Couch 教練
        - void doSpeak() → I can speak
- 多重繼承
    - 既是投手又是打擊 → class twoKnife