COMP 345 Week 8

You need examples for better understanding

h_lai@encs.concordia.ca

If we don’t have polymorphism

```
void displayCShapeDynamicCastDemo(CShape& shape)
{
  CShape*     p  = &shape;
  CCircle*    p1 = dynamic_cast<CCircle*>(p);
  CRectangle* p2 = dynamic_cast<CRectangle*>(p);
  CTriangle*  p3 = dynamic_cast<CTriangle*>(p);

  if (p1 != NULL){
    cout << "Circle's area is "      << p1->getArea()   << endl;
    cout << "Circle's radius is "    << p1->getRadius() << endl;
  }
  if (p2 != NULL){
    cout << "Rectangle's area is "   << p2->getArea()   << endl;
    cout << "Rectangle's width is "  << p2->getWidth()  << endl;
    cout << "Rectangle's height is " << p2->getLength() << endl;
  }
  if (p3 != NULL){
    cout << "Triangle's area is "    << p3->getArea()   << endl;
    cout << "Triangle's width is "   << p3->getHeight() << endl;
    cout << "Triangle's height is "  << p3->getBase()   << endl;
  }
}
```

ugly code here, we need a machanism to automatically call subtype-specific behavior

Virtual Method

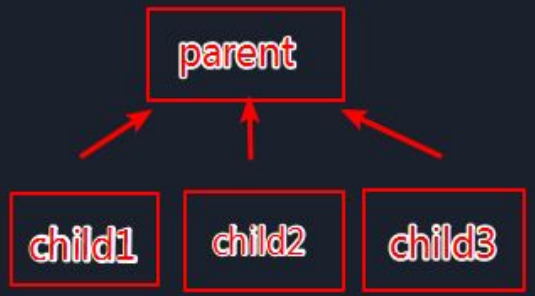
Inheritance

First we need an **inherent** relation.

super class, parent class

subclass, children class

Second we need to declare a method using virtual keyword




```
Parent *ptrParent = new Child1()
// actually executed child1 -> behaviour
ptrParent -> behaviour
```

Virtual Destructor

Always declare the destructor as virtual if a class is to be used polymorphically.

If the destructor is not declared virtual and that class is used for polymorphically, it may lead to a **memory leak**.

```
class Base {
  public:
    virtual ~Base(){
      cout << "Base::~Base()" << endl;
    }
};

class Derived : public Base {
  public:
    ~Derived(){
      cout << "Derived::~Derived()" << endl;
    }
};

int main(){
  Base *b = new Derived();
  delete b;
  int i; cin >> i;
}
```

if it is not virtual, this destructor will not be called

Pure Virtual Method and Abstract Class

Pure Virtual Method

Pure virtual method in cpp just likes abstract method in Java;

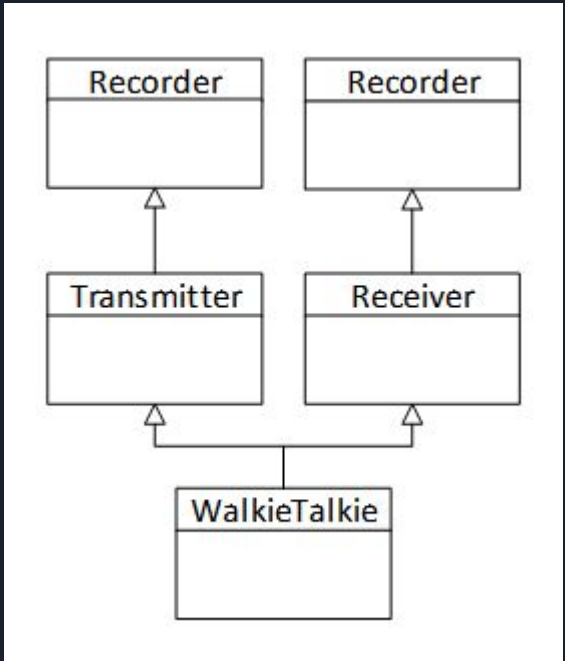
How to define a pure virtual method ?

Any class with at least one pure virtual method is an abstract class;

Can I create an object using anabstract class ?

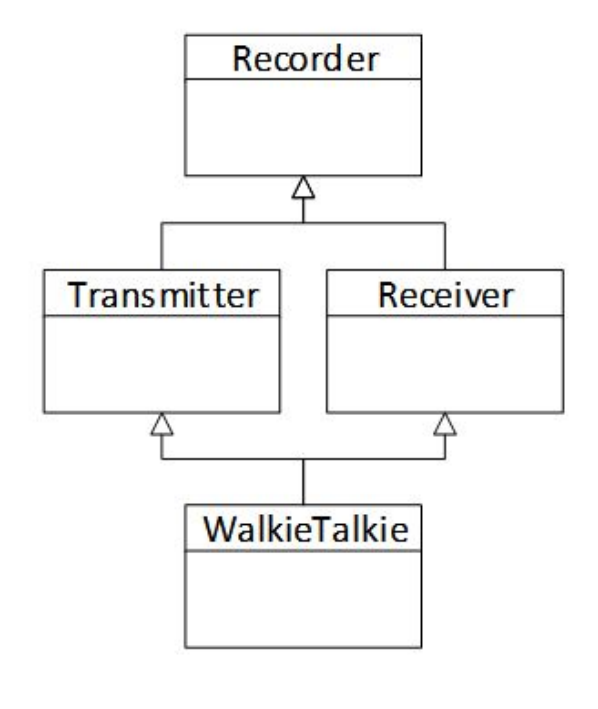
Virtual Interitance

Diamond Problem



What happen if we want to access Recorder's member through a WalkieTalkie object.

Make the diagram look like following

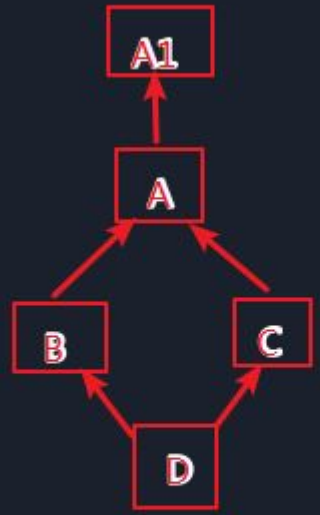


Transmitter and Receiver have the same copy of Recorder, there is no ambiguity any more !!!

The technique we used to make only have one Recorder above Transmitter and Receiver is called **virtual inheritance**

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

class A1 {
int i;
public:
A1() { cout << "in A1::A1()\n"; };
A1(int k) : i(k){ cout << "in A1::A1(int)\n"; }
};
class A : virtual public A1 {
public:
A() {cout<<"in A::A()\n";};
A(int k) : A1(k){ cout << "in A::A(int)\n";}
};

class B : public A {
public:
B(){cout<<"in B::B()\n";};
B(int i) : A1(), A(i) { cout << "in B::B(int)\n"; }
};

class C : public A {
public:
C(){ cout << "in C::C()\n"; };
C(int i) : A(i) { cout << "in C::C(int)\n"; }
};

class D : public B, C{
public:
D() { cout << "in D::D()\n"; }
D(int i) : A1(i), B(i), C(i) { cout << "in D::D(int)\n"; }
};

int main()
{
D d1(2);
D d2;
int i; cin >> i;
return 0;
}
```

```
in A1::A1(int)
in A::A(int)
in B::B(int)
in A::A(int)
in C::C(int)
in D::D(int)
in A1::A1()
in A::A()
in B::B()
in A::A()
in C::C()
in D::D()
```



```
in V::V(int)
in A::A(int)
in B::B(int)
in C::C(int)
in D::D(int)
in E::E(int)
in V::V()
in A::A()
in B::B()
in C::C()
in D::D()
in E::E()
```

```
int main()
{
  E e1(2);
  E e2;
  int i; cin >> i;
  return 0;
}
```

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

class V {
int i;
  public:
    V() { cout << "in V::V()\n"; };
    V(int i) : i(i){ cout << "in V::V(int)\n"; }
};
class A : virtual public V {
  public:
    A() { cout << "in A::A()\n"; };
    A(int i) : V(i) { cout << "in A::A(int)\n"; }
};

class B : virtual public V {
  public:
    B(){ cout << "in B::B()\n"; };
    B(int i) : V(i) { cout << "in B::B(int)\n"; }
};

class C : public A, B {
  public:
    C(){ cout << "in C::C()\n"; };
    C(int i) : A(i), B(i) { cout << "in C::C(int)\n"; }
};

class D : virtual public V {
  public:
    D() { cout << "in D::D()\n"; }
    D(int i) : V(i) { cout << "in D::D(int)\n"; }
};

class E : public C, D {
  public:
    E() { cout << "in E::E()\n"; }
    E(int i) : V(i), C(i), D(i) { cout << "in E::E(int)\n"; }
};
```