Polymorphism

Modern cpp Programming lecture 5

Polymorphism

- Last *Theoretical concept* of the course
- After this lecture…
  - remaining concepts -> much more practical
    - GUI
    - STL
    - Template
    - Operator overloading
    - Multi-thread

Polymorphism

- Definition (form Wikipedia)
  - The provision of a single interface to entities of different types
  - The use of a single symbol to represent multiple different types
  - ***Polymorphism means existing in multiple forms***
- We already learned polymorphism before!!
  - Overloading
  - Overriding

Polymorphism

- Polymorphism in this lecture
  - Actually, you had already learned the practical ways to use basic polymorphism (overloading, overriding)
  - In this lecture, we focus on *the better way* to apply them
  - using ***virtual function!!***

Polymorphism

- Polymorphism application in a broad sense
  1. Function template
     - We'll see!!
  2. Function / operator overload
     - We'll see operator overloading soon!!
  3. **Function override**
     - In this lecture, we'll learn the better way to use overriding!!

Polymorphism in cpp

- Polymorphism using virtual function
  - In cpp…
    - Allowed to allocate *the address of subclass* to

      the pointer variable of it's superclass

      ```
      class Human {};
      class Undergraduate : public Human {};
      Human* human = new Undergraduate();    // allowed!!
      ```

    - *Pointer* can point various classes
    - *Superclass Pointer* can execute *the overrode function of subclasses*
    - Object runtime type differentiates the *functionality!!*

Polymorphism in cpp

- Virtual class
  - class in which it's method(s) is(are) not yet defined
  - or will be redefined
  - Undefined methods should be defined (overrided) in subclasses
  - Many OOP languages supports the functionality
    - Java, python, javascript…

Polymorphism in cpp

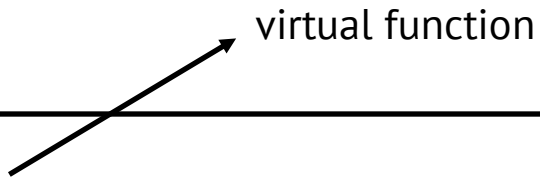
- Why virtual class?
  - Sometimes it isn't natural to implement…
    - Every functionality in the superclass
    - All sometimes evidently impossible!!
    - For example…

```
class Shape { void drawShape(); };
class Circle : public Shape { void drawShape(); };
class Rect : public Shape { void drawShape(); };
```

    - Hard to implement *drawShape* of Shape class
      - How can you draw an undefined shape??

Polymorphism in cpp

- Simple example




```
class Human {
public:
   virtual void Hi() { cout << "Hi!!" << endl; }
};

class Alice : public Human { };

int main() {
   Human* human = new Alice();
   human->Hi();                    // "Hi!!"
}
```

```
class Human {
public:
   virtual void Hi() { cout << "Hi!!" << endl; }
};

class Alice : public Human {
public:
   void Hi() { cout << "Hi!! My name is Alice!!" << endl; }
};

int main() {
   Human* human = new Alice();
   human->Hi();                    // "Hi!! My name is Alice!!" (Overrided)
}
```

Polymorphism in cpp

- Virtual function
  - Okay… why virtual class?
  - supports *runtime polymorphism* by *address type casting*

```cpp
class Human {
public:
   virtual void Hi() { cout << "Hi!!" << endl; }
};

class Alice : public Human {
public:
   void Hi() {
      cout << "Hi!! My name is Alice!!" << endl;
   }
};

int main() {
   Human* human = new Alice();
   human->Hi();
}

OUTPUT: Hi!! My name is Alice!!
```

```cpp
class Human {
public:
   void Hi() { cout << "Hi!!" << endl; }
};

class Alice : public Human {
public:
   void Hi() {
      cout << "Hi!! My name is Alice!!" << endl;
   }
};

int main() {
   Human* human = new Alice();
   human->Hi();
}

OUTPUT: Hi!!
```

without virtual keyword, runtime polymorphism is impossible!!

Virtual class

- Basic rule for virtual methods
  1. Virtual functions will be overridden
  2. Virtual functions should be accessed using pointer or reference
  3. Always defined in base class and overridden in subclass
  4. Not mandatory for subclass to re-define
     - In the case, base class version is used

Virtual class

- Pure virtual function
  - A virtual function that **should be** implemented by a subclass
    - (if the subclass is not abstract)
    - *abstract??*

```
class Human {
public:
   virtual void Hi()=0;
};

class Alice : public Human { };

Compile error!!
unimplemented pure virtual method 'Hi' in 'Alice'
virtual void Hi()=0;
```

`=0` denotes the pure virtual function

Virtual class

- Pure virtual function

```
class Human {
public:
   virtual void Hi()=0;
};

class Alice : public Human { };

Compile error!!
unimplemented pure virtual method 'Hi' in 'Alice'
virtual void Hi()=0;
```

```
class Human {
public:
   virtual void Hi()=0;
};

class Alice : public Human {
public:
   void Hi() { cout << "Hi!! My name is Alice!!" << endl; }
};

Successful compilation!!
```

Virtual class

- Abstract class
  - the class which has more than one pure virtual function
  - *Abstract*..
    - Which means *not yet specified*
    - Should be specified == should reimplement pure virtual functions
    - Class which *promises* its functionality, but no specific *implementation*

Polymorphism in cpp

- Example

```
class Ape {
public:
   virtual string introduce()=0;
};

class Chimpanzee : public Ape {
public:
   string introduce() { return "chimpanzee"; }
};

class Bonobo : public Ape {
public:
   string introduce() { return "bonobo"; }
};

class Orangutan : public Ape {
public:
   string introduce() { return "orangutan"; }
};
```

Polymorphism in cpp

- Example

```
class Ape {
public:
   virtual string introduce()=0;
};

class Chimpanzee : public Ape {
public:
   string introduce() { return "chimpanzee"; }
};

class Bonobo : public Ape {
public:
   string introduce() { return "bonobo"; }
};

class Orangutan : public Ape {
public:
   string introduce() { return "orangutan"; }
};
```

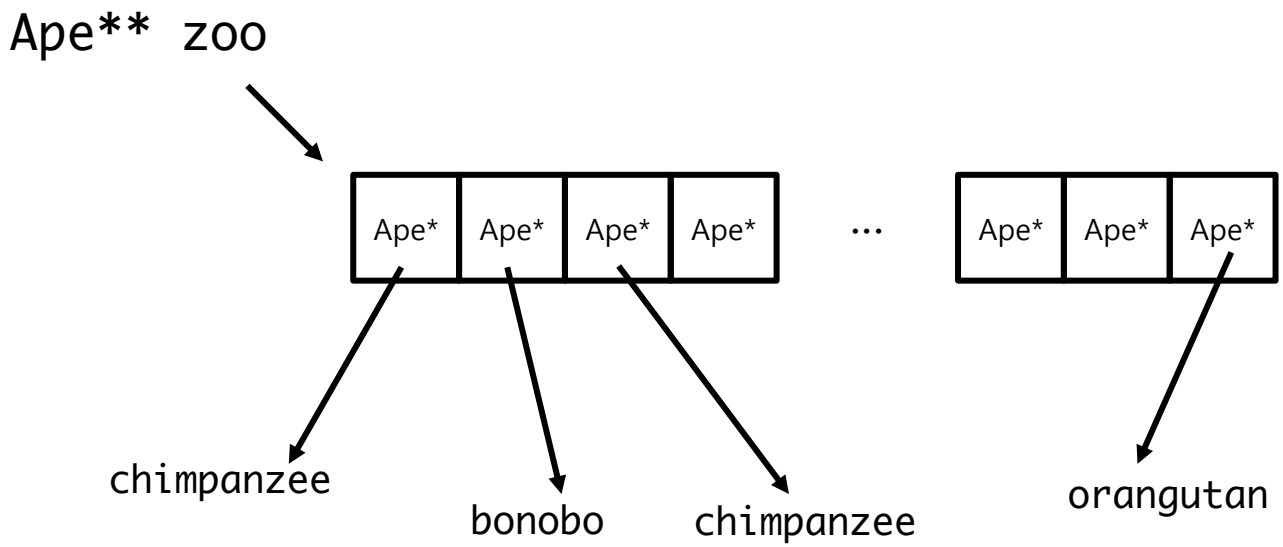
```
int main() {
   Ape** zoo = Zoo();  // returns the array consists of 100 apes
   int chim = 0; int bo = 0;  int orang = 0;
   for(int i =0; i < 100; i++) {
      if (zoo[i]->introduce() == "chimpanzee")  chim++;
      else if (zoo[i]->introduce() == "bonobo") bo++;
      else orang++;
   }
   cout << "Our zoo raises ";
   cout << chim << " chimpanzees, ";
   cout << bo << " bonobos, and ";
   cout << orang << " orangutans. ";
}
```

Polymorphism in cpp

```
class Ape {
public:
   virtual string introduce()=0;
};

class Chimpanzee : public Ape {
public:
   string introduce() { return "chimpanzee"; }
};

class Bonobo : public Ape {
public:
   string introduce() { return "bonobo"; }
};

class Orangutan : public Ape {
public:
   string introduce() { return "orangutan"; }
};
```

- Example

```
Ape** Zoo() {
   Ape** zoo = new Ape*[100];
   for(int i = 0; i < 100; i++) {
      if(i < 40) zoo[i] = new Chimpanzee;
      else if(i < 50) zoo[i] = new Bonobo
;
      else zoo[i] = new Orangutan;
   }
   return zoo;
}

int main() {
   Ape** zoo = Zoo();  // returns the array consists of 100 apes
   int chim = 0; int bo = 0;  int orang = 0;
   for(int i =0; i < 100; i++) {
      if (zoo[i]->introduce() == "chimpanzee")  chim++;
      else if (zoo[i]->introduce() == "bonobo") bo++;
      else orang++;
   }
   cout << "Our zoo raises ";
   cout << chim << " chimpanzees, ";
   cout << bo << " bonobos, and ";
   cout << orang << " orangutans. ";
}
```


Ape** zoo
Ape*
Ape*
Ape*
Ape*
...
Ape*
Ape*
Ape*
chimpanzee
bonobo
chimpanzee
orangutan

Polymorphism in cpp

- Example

```
Ape** Zoo() {
   Ape** zoo = new Ape*[100];
   for(int i = 0; i < 100; i++) {
      if(i < 40) zoo[i] = new Chimpanzee;
      else if(i < 50) zoo[i] = new Bonobo;
      else zoo[i] = new Orangutan;
   }
   return zoo;
}
```

```
int main() {
   Ape** zoo = Zoo();  // returns the array consists of 100 apes
   int chim = 0; int bo = 0;  int orang = 0;
   for(int i =0; i < 100; i++) {
      if (zoo[i]->introduce() == "chimpanzee")  chim++;
      else if (zoo[i]->introduce() == "bonobo") bo++;
      else orang++;
   }
   cout << "Our zoo raises ";
   cout << chim << " chimpanzees, ";
   cout << bo << " bonobos, and ";
   cout << orang << " orangutans. ";
}
```

```
OUTPUT: Our zoo raises 40 chimpanzees, 10 bonobos, and 50 orangutans.
```

Polymorphism in cpp

- Library
  - Not for cpp, but in Java library…
  - There exists both *LinkedList and ArrayList*
    - Both have same interface
    - However, the implementation is different!!
    - *LinkedList Class* uses Linked list to implement list
    - *ArrayList Class* uses Array to implement list
  - *Same interface, but different functionality*
    - ***The final goal of polymorphism!!***

From now on, we'll learn…

- the practical concepts of cpp!!
  - GUI
  - STL
  - Template
  - Operator overloading
- Almost everything…
  - Implemented using *inheritance & polymorphism*

Thank you!!

contact: jeonhyun97@postech.ac.kr