1

가상 함수와 오버라이딩 활용 사례

- 가상 함수를 가진 기본 클래스의 목적
- 가상 함수 오버라이딩
- 동적 바인딩 실행
- 기본 클래스의 포인터 활용

1. 가상 함수를 가진 기본 클래스의 목적

Shape은 상속을 위한 기본 클래스로의 역할
- 가상 함수 draw()로 파생 클래스의 인터페이스를 보여줌
- Shape 객체를 생성할 목적 아님
- 파생 클래스에서 draw() 재정의. 자신의 도형을 그리도록 유도

Shape.h

```
class Shape {
    Shape* next;
protected:
    virtual void draw();
public:
    Shape() { next = NULL; }
    virtual ~Shape() { }
    void paint();
    Shape* add(Shape* p);
    Shape* getNext() { return next;}
};
```

Shape.cpp

```
#include <iostream>
#include "Shape.h"
using namespace std;

void Shape::paint() {
    draw();
}

void Shape::draw() {
    cout << "--Shape--" << endl;
}

Shape* Shape::add(Shape *p) {
    this->next = p;
    return p;
}
```

Circle.h

```
class Circle : public Shape {
protected:
    virtual void draw();
};
```

Circle.cpp

```
#include <iostream>
#include "Shape.h"
#include "Circle.h"
using namespace std;

void Circle::draw() {
    cout << "Circle" << endl;
}
```

Rect.h

```
class Rect : public Shape {
protected:
    virtual void draw();
};
```

Rect.cpp

```
#include <iostream>
#include "Shape.h"
#include "Rect.h"
using namespace std;

void Rect::draw() {
    cout << "Rectangle" << endl;
}
```

Line.h

```
class Line : public Shape {
protected:
    virtual void draw();
};
```

Line.cpp

```
#include <iostream>
#include "Shape.h"
#include "Line.h"
using namespace std;

void Line::draw() {
    cout << "Line" << endl;
}
```

2. 가상 함수 오버라이딩

- 파생 클래스마다 다르게 구현하는 다형성

```
void Circle::draw() { cout << "Circle" << endl; }

void Rect::draw() { cout << "Rectangle" << endl; }

void Line::draw() { cout << "Line" << endl; }
```

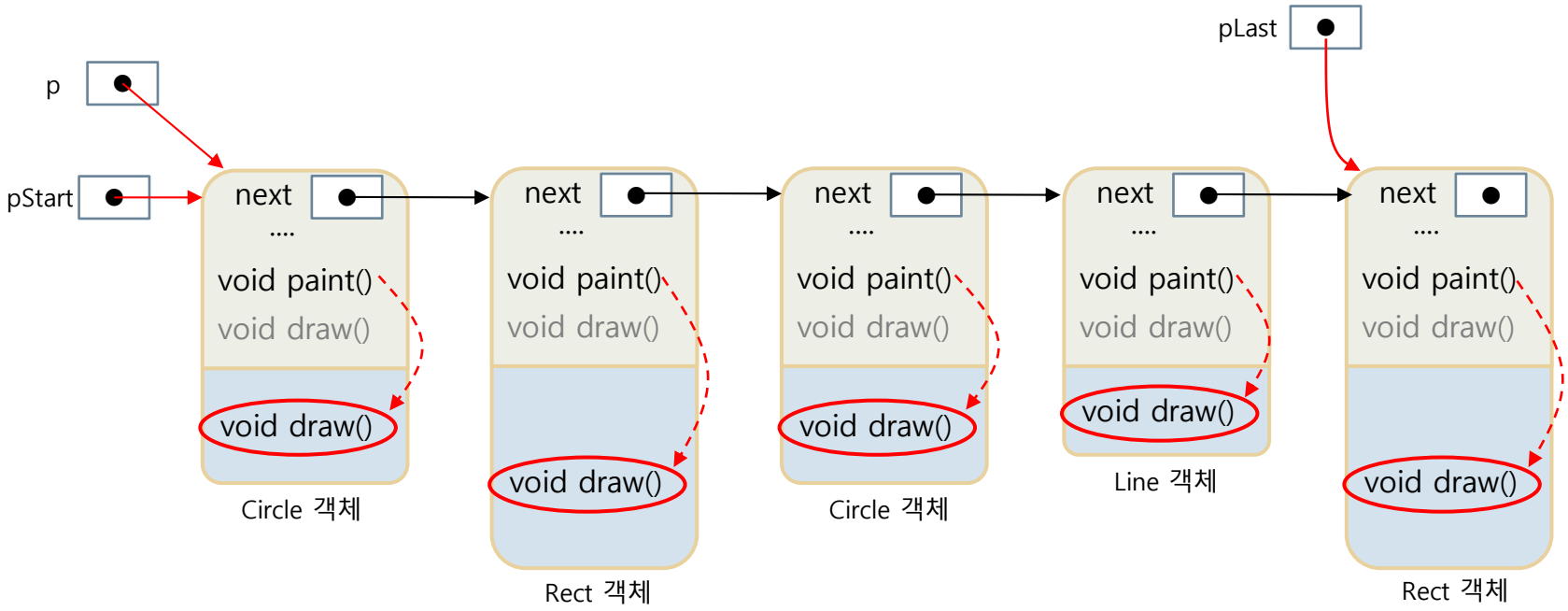
- 파생 클래스에서 가상 함수 draw()의 재정의
  - 어떤 경우에도 자신이 만든 draw()가 호출됨을 보장 받음
    - 동적 바인딩에 의해

3. 동적 바인딩 실행 : 파생 클래스의 가상 함수 실행


```
#include <iostream>
#include "Shape.h"
#include "Circle.h"
#include "Rect.h"
#include "Line.h"
using namespace std;

int main() {
    Shape *pStart=NULL;
    Shape *pLast;

    pStart = new Circle(); // 처음에 원 도형을 생성한다.
    pLast = pStart;

    pLast = pLast->add(new Rect()); // 사각형 객체 생성
    pLast = pLast->add(new Circle()); // 원 객체 생성
    pLast = pLast->add(new Line()); // 선 객체 생성
    pLast = pLast->add(new Rect()); // 사각형 객체 생성

    // 현재 연결된 모든 도형을 화면에 그린다.
    Shape* p = pStart;
    while(p != NULL) {
        p->paint();
        p = p->getNext();
    }
    // 현재 연결된 모든 도형을 삭제한다.
    p = pStart;
    while(p != NULL) {
        Shape* q = p->getNext(); // 다음 도형 주소 기억
        delete p; // 기본 클래스의 가상 소멸자 호출
        p = q; // 다음 도형 주소를 p에 저장
    }
}
```

```
Circle
Rectangle
Circle
Line
Rectangle
```

main() 함수가 실행될 때 구성된 객체의 연결



4. 기본 클래스의 포인터 활용

- 기본 클래스의 포인터로 파생 클래스 접근
  - pStart, pLast, p의 타입이 Shape*
  - 링크드 리스트를 따라 Shape을 상속받은 파생 객체들 접근
  - p->paint()의 간단한 호출로 파생 객체에 오버라이딩된 draw() 함수 호출

순수 가상 함수

- 기본 클래스의 가상 함수 목적
  - 파생 클래스에서 재정의할 함수를 알려주는 역할
    - 실행할 코드를 작성할 목적이 아님
  - *기본 클래스의 가상 함수를 굳이 구현할 필요가 있을까?*
- 순수 가상 함수
  - pure virtual function
  - 함수의 **코드가 없고** 선언만 있는 **가상 멤버** 함수
  - 선언 방법
    - 멤버 함수의 원형=0;으로 선언

```
class Shape {
public:
   virtual void draw()=0; // 순수 가상 함수 선언
};
```

추상 클래스

- 추상 클래스 : 최소한 하나의 순수 가상 함수를 가진 클래스

```
class Shape { // Shape은 추상 클래스
    Shape *next;
public:
    void paint() {
        draw();
    }
    virtual void draw() = 0; // 순수 가상 함수
};
void Shape::paint() {
    draw(); // 순수 가상 함수라도 호출은 할 수 있다.
}
```

- 추상 클래스의 특징
  - 온전한 클래스가 아니므로 객체 생성 불가능

```
Shape shape; // 컴파일 오류
Shape *p = new Shape(); // 컴파일 오류
```

error C2259: 'Shape' : 추상 클래스를 인스턴스화할 수 없습니다.

  - 추상 클래스의 포인터는 선언 가능

```
Shape *p;
```

추상 클래스의 목적

- 추상 클래스의 목적
  - 추상 클래스의 인스턴스를 생성할 목적 아님
  - 상속에서 기본 클래스의 역할을 하기 위함
    - 순수 가상 함수를 통해 파생 클래스에서 구현할 함수의 형태(원형)을 보여주는 인터페이스 역할
    - 추상 클래스의 모든 멤버 함수를 순수 가상 함수로 선언할 필요 없음

추상 클래스의 상속과 구현

- 추상 클래스의 상속
  - 추상 클래스를 단순 상속하면 자동 추상 클래스
- 추상 클래스의 구현
  - 추상 클래스를 상속받아 순수 가상 함수를 오버라이딩
    - 파생 클래스는 추상 클래스가 아님

Shape은 추상 클래스

Circle도 추상 클래스

```
class Shape {
public:
   virtual void draw() = 0;
};

class Circle : public Shape {
public:
   string toString() { return "Circle 객체"; }
};


Shape shape; // 객체 생성 오류
Circle waffle; // 객체 생성 오류
```

추상 클래스의 단순 상속

```
class Shape {
public:
   virtual void draw() = 0;
};

class Circle : public Shape {
public:
   virtual void draw() {
      cout << "Circle";
   }
   string toString() { return "Circle 객체"; }
};


Shape shape; // 객체 생성 오류
Circle waffle; // 정상적인 객체 생성
```

Shape은 추상 클래스

Circle은 추상 클래스 아님

순수 가상 함수 오버라이딩

추상 클래스의 구현

Shape을 추상 클래스로 수정

Shape.h

```
class Shape {
    Shape* next;
protected:
    virtual void draw() = 0;
public:
    Shape() { next = NULL; }
    virtual ~Shape() { }
    void paint();
    Shape* add(Shape* p);
    Shape* getNext() { return next;}
};
```

Shape은 추상 클래스

Shape.cpp

```
#include <iostream>
#include "Shape.h"
using namespace std;

void Shape::paint() {
    draw();
}

void Shape::draw() {
    cout << "--Shape--" << endl;
}

Shape* Shape::add(Shape *p) {
    this->next = p;
    return p;
}
```

Circle.h

```
class Circle : public Shape {
protected:
    virtual void draw();
};
```

Circle.cpp

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

#include "Shape.h"
#include "Circle.h"

void Circle::draw() {
    cout << "Circle" << endl;
}
```

Rect.h

```
class Rect : public Shape {
protected:
    virtual void draw();
};
```

Rect.cpp

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

#include "Shape.h"
#include "Rect.h"

void Rect::draw() {
    cout << "Rectangle" << endl;
}
```

Line.h

```
class Line : public Shape {
protected:
    virtual void draw();
};
```

Line.cpp

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

#include "Shape.h"
#include "Line.h"

void Line::draw() {
    cout << "Line" << endl;
}
```

예제 9-7(실습) 추상 클래스를 상속받는 파생 클래스 구현 연습

다음 코드와 실행 결과를 참고하여 추상 클래스 Calculator를 상속받는 Adder와 Subractor 클래스를 구현하라.

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

class Calculator {
    void input() {
        cout << "정수 2 개를 입력하세요>> ";
        cin >> a >> b;
    }
protected:
    int a, b;
    virtual int calc(int a, int b) = 0; // 두 정수의 합 리턴
public:
    void run() {
        input();
        cout << "계산된 값은 " << calc(a, b) << endl;
    }
};

int main() {
    Adder adder;
    Subtractor subtractor;
    adder.run();
    subtractor.run();
}
```

adder.run()에 의한 실행 결과

subtractor.run()에 의한 실행 결과

```
정수 2 개를 입력하세요>> 5 3
계산된 값은 8
정수 2 개를 입력하세요>> 5 3
계산된 값은 2
```