제4장 객체 지향 소개

이번 장에서 학습할 내용

- 객체지향이란?
- 객체
- 메시지
- 클래스
- 객체 지향의 장점

- 다형식 클래스

객체 지향 개념을 완벽하게 이해해야만 객체 지향 설계의 이점을 활용할 수 있다.

객체 지향이란?

- 실제 세계를 모델링하여 소프트웨어를 개발하는 방법

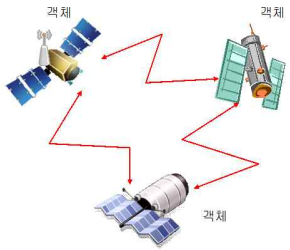
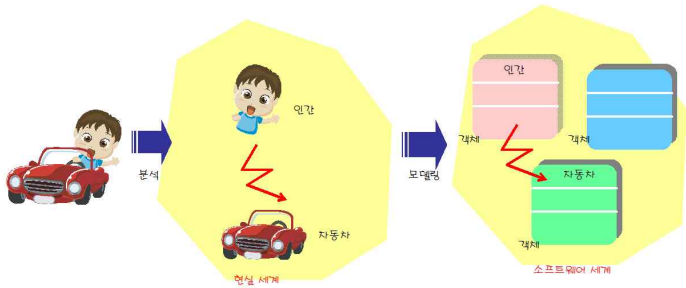


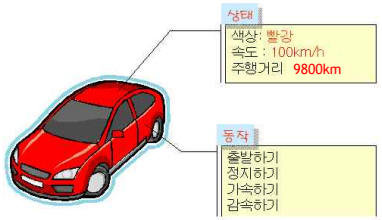
그림 4.1 실제 세계는 객체들로 이루어진다.

객체 지향의 과정

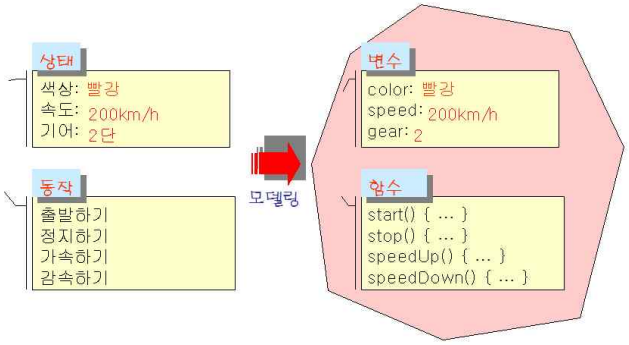


객체란?

- 객체(object)는 상태와 동작을 가지고 있다.
- 객체의 상태(state)는 객체의 특징값(속성)이다.
- 객체의 동작(behavior) 또는 행동은 객체가 취할 수 있는 동작



멤버 변수와 멤버 함수



소프트웨어 객체 = 변수 + 함수

메시지

- 소프트웨어 객체는 메시지(message)를 통해 다른 소프트웨어 객체와 통신하고 서로 상호 작용한다.

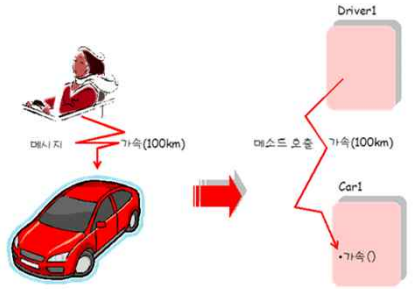


그림 7.5 메시지 전달

중간 점검 문제

1. 다음과 같은 실제 세계의 객체에서 가능한 상태와 동작을 정리하여 보자.

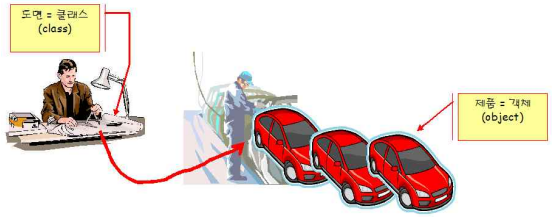
객체	상태	동작
전구		
라디오		
강아지		
자전거		
사자		

2. 객체들은 _____전달을 통해서 서로 간에 상호 작용을 한다.

3. 자동차 객체에서 생각할 수 있는 메시지와 매개 변수에 대하여 나열하여 보라.

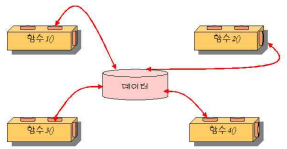
클래스

- 클래스(class): 객체를 만드는 설계도
- 클래스로부터 만들어지는 각각의 객체를 특별히 그 클래스의 **인스턴스(instance)**라고도 한다.

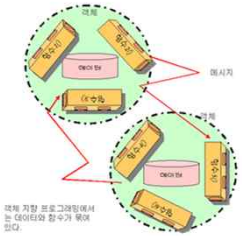


절차 지향과 객체 지향

- 절차 지향 프로그래밍
  - 문제를 해결하는 절차를 중요하게 생각하는 소프트웨어 개발 방법. 이들 절차는 모두 함수라는 단위로 묶이게 된다.
- 객체 지향 프로그래밍
  - 데이터와 함수를 하나의 덩어리로 묶어서 생각하는 방법이다. 데이터와 함수를 객체로 묶는 것을 캡슐화(encapsulation)라고 부른다.



절차 지향 프로그래밍에서는 데이터와 함수가 묶여 있지 않다.



객체 지향 프로그래밍에서는 데이터와 함수가 묶여 있다.

객체 지향의 장점

- 객체들을 조립하여서 빠르게 소프트웨어를 만들 수 있다.

부품을 조립하여 제품을 만들듯이 객체들을 조립하여 소프트웨어를 만든다.

자동차 경주 게임의 예

절차 지향

```
struct Car {
   int speed;
   int gear;
   char *pcolor;
};

void init(Car& c, char
   *color);
void start(Car& c);
void stop(Car& c);
int get_speed(Car& c);
void set_speed(Car& c, int
   speed);

int main()
{
   Car car;
   init(car, "red");
   start(car);
   set_speed(car, 60);
   stop(car);
   return 0;
}
```

객체 지향

```
class Car {
   int speed;
   int gear;
   char *pcolor;

public:
   void init(char *color);
   void start();
   void stop();
   int get_speed();
   void set_speed(int speed);
};

int main()
{
   Car car;
   car.init("red");
   car.start();
   car.set_speed(60);
   car.stop(car);
   return 0;
}
```

객체 지향의 개념들

- 캡슐화(encapsulation)
- 정보 은닉(information-hiding)
- 상속(inheritance)
- 다형성(polymorphism)

캡슐화

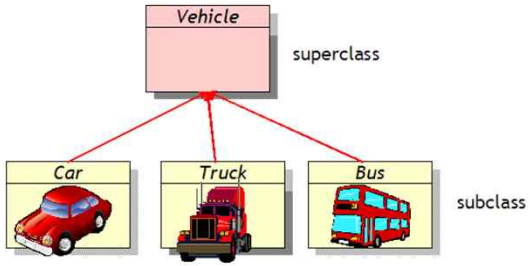
- 캡슐화(encapsulation)란 데이터와 연산들을 객체 안에 넣어서 묶는다는 의미이다.

정보 은닉

- 객체 내부의 데이터와 구현의 세부 사항을 외부 세계에게 감추는 것.
- 외부 세계에 영향을 끼치지 않으면서 쉽게 객체 내부를 업그레이드 할 수 있다.

상속

- 상속은 기존의 코드를 재활용하기 위한 기법으로 이미 작성된 클래스(부모 클래스)를 이어받아서 새로운 클래스(자식 클래스)를 생성하는 기법이다.



다형성

- 다형성이란 객체가 취하는 동작이 상황에 따라서 달라지는 것을 의미한다. -> 함수 이름의 재사용

쉬운 디버깅

- 예를 들어서 절차 지향 프로그램에서 하나의 변수를 1000개의 함수가 사용하고 있다고 가정해보자. -> 하나의 변수를 1000개의 함수에서 변경할 수 있다.
- 객체 지향 프로그램에서 100개의 클래스가 있고 클래스당 10개의 멤버 함수를 가정해보자. -> 하나의 변수를 10개의 함수에서 변경할 수 있다.
- 어떤 방법이 디버깅이 쉬울까?

객체 지향의 장점

- 신뢰성있는 소프트웨어를 쉽게 작성할 수 있다.
- 코드를 재사용하기 쉽다.
- 업그레이드가 쉽다.
- 디버깅이 쉽다.

중간 점검 문제

1. 자바에서 코드 재사용이 쉬운 이유는 관련된 _____와 _____이 하나의 덩어리로 묶여 있기 때문이다.

2. 정보 은닉이란 _____을 외부로부터 보호하는 것이다.

3. 정보를 은닉하면 발생하는 장점은 무엇인가?

클래스의 구성

- 클래스(class)는 객체의 설계도라 할 수 있다.
- 클래스는 멤버 변수와 멤버 함수로 이루어진다.
- 멤버 변수는 객체의 속성을 나타낸다.
- 멤버 함수는 객체의 동작을 나타낸다.

클래스 정의의 예

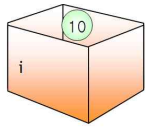
```
class Car {

public:
    // 멤버 변수 선언
    int speed; // 속도
    int gear; // 기어
    string color; // 색상

    // 멤버 함수 선언
    void speedUp() { // 속도 증가 멤버 함수
        speed += 10;
    }

    void speedDown() { // 속도 감소 멤버 함수
        speed -= 10;
    }
};
```

멤버 변수 정의!

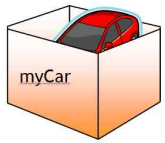
멤버 함수 정의!

객체

- int 타입의 변수를 선언하는 경우
  int i;



- 클래스도 타입으로 생각하면 된다.
- Car 타입의 변수를 선언하면 객체가 생성된다.
  Car myCar;



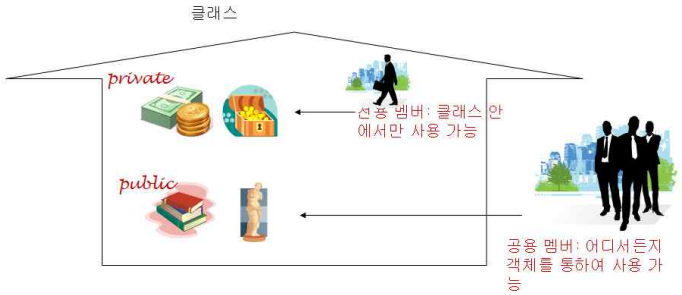
객체의 사용

- 객체를 이용하여 멤버에 접근할 수 있다.

이들 멤버 변수에 접근하기 위해서는 도트(.) 연산자를 사용한다.

```
myCar.speed = 100;
myCar.speedUp();
myCar.speedDown();
```

접근 제어



전용 멤버와 공용 멤버

- 전용 멤버(private member)
  - 클래스 내부에서만 접근이 허용
- 공용 멤버(public member)
  - 공용 멤버는 다른 모든 클래스들이 사용 가능

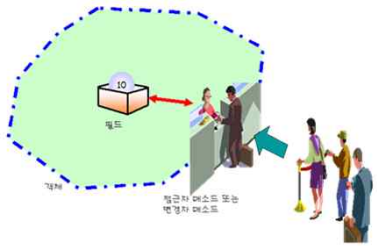
접근자와 설정자

- 접근자(accessor): 멤버 변수의 값을 반환

(예) getBalance()

- 설정자(mutator): 멤버 변수의 값을 설정

(예) setBalance();



UML

- UML(Unified Modeling Language): 애플리케이션을 구성하는 클래스들간의 관계를 그리기 위하여 사용

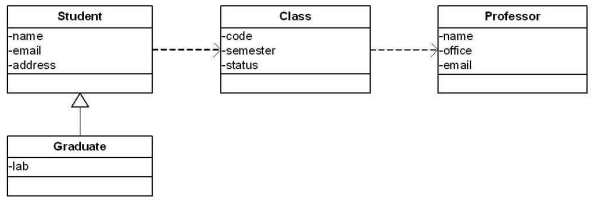
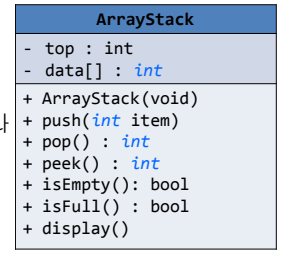


그림 8.9 UML의 예

UML 클래스 다이어그램 예

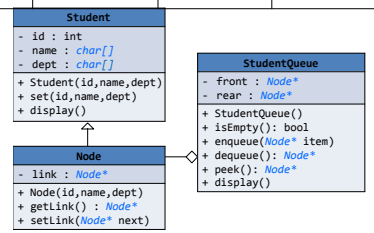
- 클래스 다이어그램
  - 첫 번째 박스는 클래스의 이름
  - 두 번째 박스는 데이터 멤버
    - 스택의 top변수
    - 항목을 저장할 배열 (정수라
    - '-': private의 접근 권한
  - 세 번째 박스는 멤버 함수
    - 생성자
    - 연산들
    - 매개 변수, 반환형



클래스와 클래스의 관계

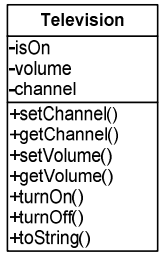
- UML, Class Diagram

관계	UML Symbol	의미	예
inheritance		is-a	A book is a printed resource.
aggregation		has-a	A book has a publisher.
composition		contains-a	A book contains a chapter.



중간 점검 문제

- TV를 나타내는 클래스를 정의하고 UML의 클래스 다이어그램으로 표현하라.



복소수 클래스

- 실습 4에서 구현한 복소수 구조체와 관련 함수들을 클래스로 구현하려고 한다.
  - 복소수 클래스의 UML 클래스 다이어그램을 작성하라.
    - 데이터 멤버는?
    - 멤버 함수는?
    - 각 멤버의 접근 권한은?
  - UML 다이어그램을 바탕으로 복소수 클래스를 작성하라.
    - 실습에서 구현했던 코드를 적절히 재배치하면 됨

Complex V1: 구조체 버젼

Complex.h

```
#pragma once
#include <cstdio>

// 복소수 구조체 선언
struct Complex
{
    double  real;
    double  imag;
};

// 복소수 내용을 설정하는 함수 : inline
inline void setComplex( Complex& c, double r, double i ) {
    c.real = r;
    c.imag = i;
}

extern Complex  readComplex( char* msg = " 복소수 입력 =" );
extern void     printComplex( Complex& c, char* msg = "복소수 =
extern Complex  addComplex( Complex a, Complex b );
```

Complex.cpp

```
#include "Complex.h"

// 복소수를 읽어들이는 함수. 표준
Complex readComplex( char* msg )
{
    Complex c;
    printf("  %s ", msg);
    scanf("%lf%lf", &c.real, &c.imag);
    return c;
}
void printComplex( const Complex& c, char* msg )
{
    printf("  %s %4.2f + %4.2fi\n", msg, c.real, c.imag
}

// 더하기 / 빼기 등의 멤버함수 추가
Complex addComplex( Complex a, Complex b )
{
    Complex c;
    c.real = a.real + b.real;
    c.imag = a.imag + b.imag;
    return c;
}
```

main.cpp

```
#include "Complex.h"
#include "QuadFn.h"

void main()
{
    Complex a, b, c;

    a = readComplex ( "A =" );
    b = readComplex ( "B =" );
    c = addComplex (a, b);

    printComplex( a, "  A  = ");
    printComplex( b, "  B  = ");
    printComplex( c, " A+B = ");
```


Complex V2: 클래스 버젼

Complex.h

```
class Complex
{
    double  real;
    double  imag;
public:
    // 복소수 내용을 설정하는 함수 : inline
    void setComplex( double r, double i ) {
        real = r;
        imag = i;
    }
    Complex readComplex( char* msg = " 복소수 입력 =" );
    void    printComplex( char* msg = "복소수 = " );
    void    addComplex ( Complex a, Complex b );
};
```

Complex.cpp

```
void Complex::readComplex( char* msg )
{
    printf("  %s ", msg);
    scanf("%lf%lf", &real, &imag);
}
void Complex::printComplex( char* msg )
{
    printf("  %s %4.2f + %4.2fi\n", msg, real, imag);
}
// 더하기 / 빼기 등의 멤버함수 추가
void Complex::addComplex( Complex a, Complex b )
{
    real = a.real + b.real;
    imag = a.imag + b.imag;
}
```

main.cpp

```
void main()
{
    Complex a, b, c;

    a.readComplex ( "A =" );
    b.readComplex ( "B =" );
    c.addComplex (a, b);

    a.printComplex( "  A  = ");
    b.printComplex( "  B  = ");
    c.printComplex( " A+B = ");
```


Complex V3: 이름 변경

Complex.h

```
class Complex
{
    double  real;
    double  imag;
public:
    // 복소수 내용을 설정하는 함수 : inline
    void set( double r, double i ) {
        real = r;
        imag = i;
    }
    void    read( char* msg = " 복소수 입력 =" );
    void    print( char* msg = "복소수 = " );
    void    add( Complex a, Complex b );
};
```

Complex.cpp

```
void Complex::read( char* msg )
{
    printf("  %s ", msg);
    scanf("%lf%lf", &real, &imag);
}

void Complex::print( char* msg )
{
    printf("  %s %4.2f + %4.2fi\n", msg, real, imag);
}

// 더하기 / 빼기 등의 멤버함수 추가
void Complex::add( Complex a, Complex b )
{
    real = a.real + b.real;
    imag = a.imag + b.imag;
}
```

```
    a.read ( "A =" );
    b.read ( "B =" );
    c.add (a, b);

    a.print ( "  A  = ");
    b.print ( "  B  = ");
    c.print ( " A+B = ");
```


Complex V4: inline 처리

Complex.h

```
class Complex
{
    double  real;
    double  imag;
public:
    // 복소수 내용을 설정하는 함수 : inline
    void set( double r, double i ) {
        real = r;
        imag = i;
    }
    void read( char* msg = " 복소수 입력 =" )
    {
        printf("  %s ", msg);
        scanf("%lf%lf", &real, &imag);
    }
    void print( char* msg = "복소수 = " )
    {
        printf("  %s %4.2f + %4.2fi\n", msg, real, imag);
    }

    // 더하기 / 빼기 등의 멤버함수 추가
    void add( Complex a, Complex b )
    {
        real = a.real + b.real;
        imag = a.imag + b.imag;
    }
};
```

main.cpp

```
void main()
{
    Complex a, b, c;

    a.read ( "A =" );
    b.read ( "B =" );
    c.add (a, b);

    a.print ( "  A  = ");
    b.print ( "  B  = ");
    c.print ( " A+B = ");
```

실습4.2 구구단 게임 클래스

- 실습 2에서 구현한 구구단 게임 프로그램을 클래스로 구현하려고 한다.
  - 구구단 게임 클래스의 UML 다이어그램을 작성하라.
    - 데이터 멤버는?
    - 멤버 함수는?
    - 각 멤버의 접근 권한은?
    - 포함이나 상속 관계는?
  - UML 다이어그램을 바탕으로 구구단 게임 클래스를 작성하라.
    - 실습에서 구현했던 코드를 적절히 재배치하면 됨

실습4.2 랭킹 관리 클래스

- 실습 3에서 구현한 랭킹 관리 프로그램을 클래스로 구현하려고 한다.
  - 랭킹 관리 클래스의 UML 다이어그램을 작성하라.
  - UML 다이어그램을 바탕으로 랭킹 관리 클래스를 작성하라.
  - 구구단 게임 클래스와 동일한 절차를 사용하면 됨
- 구구단 게임 클래스와 랭킹 관리 클래스를 이용하여 실습 3에서 구현한 랭킹 관리가 되는 구구단 게임 프로그램을 작성하라.
  - main() 함수에서 구현하면됨
  - 구구단 게임 클래스와 랭킹 클래스를 main()함수에서 사용함.

Q & A