# 객체 지향 프로그래밍
# (Object Oriented Programming)

10장

강사 – 강대기

# 차례 (Agenda)

- 객체지향 프로그래밍
- 클래스의 개념 정의와 구현
- 데이터 멤버 / 클래스 메소드
- private / protected / public
- 생성과 사용, 사멸 – 생성자, 파괴자
- const 멤버 함수
- this 포인터
- 객체 배열
- 사용 범위

# 절차적 프로그래밍과 객체지향 프로그래밍

- 절차적 프로그래밍 – 절차에 초점을 맞추어 문제를 분석 (562쪽)
- 객체지향 프로그래밍 – 데이터에 초점을 맞추어 문제를 분석 (563쪽)

- 추상화: 플라톤 – 이데아의 고양이
- 기본형 : 연산자 = 사용자 정의형 : 연산자

# 클래스의 개념 정의와 구현

- 클래스 = 데이터 + 인터페이스 (567,583쪽)
- 데이터는 private, 인터페이스는 public
  - cin 객체와 getline 인터페이스를 통해 통신
- 그림 10.1 (569쪽)
  - 클래스 선언 안에선 인라인 함수
- 참고 – cin, cout, cerr, clog

# 데이터 멤버 / 클래스 메소드

- 데이터 멤버는 헤더 파일 (567쪽), 메소드는 소스 파일 (573쪽)

  char * Bozo::Retort()
- 클래스 안에서 메소드는 인라인, 밖에서는 함수이나 inline을 붙일 수 있음 (576쪽)
- 객체는 멤버 연산자로 메소드 사용 (577쪽)

  Bozo bozetta;

  Bozetta.Retort();
- 클라이언트 서버 모델로서의 OOP (582쪽)

# private / protected / public

- 클래스는 자동으로 private, 구조체는 자동으로 public (571쪽 노트)
- private – 객체 밖에서 그 멤버에 접근 불가
- public – 객체 밖에서 그 멤버에 접근 가능
- protected – 객체 내부나 상속된 객체에서 접근 가능

# 생성과 사용, 사멸 – 생성자, 파괴자

- *모든 객체는 그것을 생성할 때 초기화한다!*
- struct하고 같은 방법으로 초기화할 수 없음
  - private 멤버들 때문 → 생성자 사용 (585쪽)
- 생성자는 클래스 이름과 같으며, 리턴값이 없고, 오버로딩이 가능함

# 생성자 사용하기 (587쪽)

- 명시적 호출
  - Stock food = Stock(“Cabbage”, 250, 1.25);
- 암시적 호출
  - Stock garment(“Mason”, 50, 2.5);
- 동적 메모리 할당
  - Stock* pS = new Stock(“Games”, 18, 19.0);

# 컴파일러가 은근슬쩍 만드는 네 가지

- 빈 클래스라도 컴파일러는 네 가지를 은근슬쩍 만들어 놓고 있음
  - 디폴트 생성자, 소멸자, 복사 생성자, 복사 대입 연산자

```
class Empty {};
```

➔

```
class Empty {
public:
  Empty() {...} // 다른 생성자가 없을 경우에만 만듦
  ~Empty() {...}
  Empty (const Empty& rhs) {}
  Empty& operator=(const Empty& rhs) {}
};
```

# 디폴트 생성자 (589쪽)

- 초기값을 제공하지 않을 때 사용되는 생성자
  - Stock stock1;
    - Stock:: Stock() {} 의 형태로 들어있음
    - 기본형으로 int x; 라고만 선언하는 것과 동일
- 자신만의 디폴트 생성자
  - 디폴트 전달 인자
    - Stock (const char* co="Error", int n=0, double pr = 0.0);
  - 함수 오버로딩
    - Stock ();
- 디폴트 생성자를 통한 객체 생성
  - Stock first; // 디폴트 생성자 암시적 호출
  - Stock first = Stock(); // 디폴트 생성자 명시적 호출
  - Stock* p = new Stock; // 디폴트 생성자 암시적 호출
  - Stock first("Concrete"); // 일반 생성자 호출
  - Stock second(); // 이건 생성자 호출이 아니라, 함수 선언!!!

# 파괴자(590쪽)

- 생성자나 파괴자는 사용자가 호출하는 게 아님 (예외 : 위치지정 new, 760쪽)
  - 해당 객체가 사라질 때 호출됨
- 생성자가 포인터인 멤버에 대해 new를 했다면, 파괴자는 delete
- 파괴자는 클래스 이름 앞에 ~를 붙이며 전달인자가 없음
  - ~Stock();
- 참고 – Java의 finalize()

# Effective C++의 처음 다섯 항목

1. C++는 여러 언어들의 집합체
2. #define 대신 const, enum, inline 사용
3. 낌새만 보이면 const를 들이댄다
4. 객체를 만들면 사용하기 전, 반드시 초기화
5. C++가 은근슬쩍 만드는 함수들에 조심

# 객체 생성 예 (596쪽, 598쪽)

- Stock s1(“NanoSmart”,12,20.0);
- Stock s2 = Stock(“Boffo”,2,2.0);
  - 임시 객체를 사용할 수도 있고 안할 수도 있음
- s1 = s2; (599쪽)
- s1 = Stock(“Nifty”,2,2.0);
  - 객체 대입 시에 임시 객체 생성/사멸

# const 멤버 함수

- 멤버 함수가 해당 객체를 변경하지 않는다는 것을 보장하는 방법
  - void show() const;
  - void Stock::show() const;
  - const double Stock::func(const int x) const;

# 생성자가 한 개의 전달인자만 가질 때 (601쪽)

- Bozo(int age);
  - Bozo dribble = Bozo(44);
  - Bozo roon(66); // int x(20);
  - Bozo tubby = 32; // int x=20;

# this 포인터 (603쪽)

- 클래스 객체 안에서 바로 객체 자기 자신을 가리키는 포인터
- total_val → this->total_val
- total() → this->total()
- *this

# 객체 배열

```
Stock myStuff[4]; // 디폴트 생성자 필요함
myStuff[0].update();
myStuff[3].show();
Stock tops = myStuff[2].topval(myStuff[1]);

const int STKS = 10;
Stock stocks[STKS] = {// 10 개 중 4 개만... 디폴트 생성자 필요함
  Stock("Nano", 12.5, 20),
  Stock("Boffo", 200, 2.0),
  Stock("Mono", 130, 3.25),
  Stock(), // 서로 다른 생성자 사용 가능
};
```

# 사용 범위 (616쪽)

```
Stock sleeper("Ore", 100, 0.25);
sleeper.show();
show(); // 틀렸음. 메서드는 직접 호출 안됨.

class Stock {
private:
  const int LENS = 30; // 실패, 메모리 할당 불가능하므로
  char company[LENS];

class Stock {
private:
  enum {LENS = 30}; // 성공, 클래스용 상수
  char company[LENS];

class Stock {
private:
  static const int LENS = 30; // 성공, 정적 공간으로 설정
  char company[LENS];
```

# 추상 데이터 형 (ADT)

- 객체 지향 기법으로 추상 데이터 형을 기술하기에 매우 적합함
- 스택(620쪽, 943쪽)의 구현과 클래스 템플릿(942쪽)으로 구현된 스택 (945쪽)을 참조할 것

- createEmptyStack → Stack()
- push → bool push(const T&);
- pop → bool pop(const T&);
- isFull → bool isFull() const;
- isEmpty → bool isEmpty() const;