unique_ptr

Status	시작 전

> C++ 의 경우 획득한 자원을 해제해주지 않는 이상 프로그램이 종료되기 전까지 남아있게 된다.

자원 해제가 애매한 경우

```
#include <iostream>

class A {
  int *data;

 public:
  A() {
    data = new int[100];
    std::cout << "자원을 획득함!" << std::endl;
  }

  ~A() {
    std::cout << "자원을 해제함!" << std::endl;
    delete[] data;
  }
};

void thrower() {
  // 예외를 발생시킴!
  throw 1;
}

void do_something() {
  A *pa = new A();
  thrower();
```

```
  // 발생된 예외로 인해 delete pa 가 호출되지 않는다!
  delete pa;
}

int main() {
  try {
    do_something();
  } catch (int i) {
    std::cout << "예외 발생!" << std::endl;
    // 예외 처리
  }
}
```

`thrower()` 로 에서 발생된 예외로 인해, 밑에 있는 `delete pa` 가 실행 되지 않고 넘어가버렸습니다. 물론 예외는 정상적으로 처리되었지만, 이로 인해 메모리 누수는 피할 수 없게 됩니다.

Resource Acquisition Is Initialization - RAII

자원의 획득은 초기화다.

자원 관리를 스택에 할당한 객체를 통해 수행하는 것입니다.

지난 강좌에서 **예외가 발생해서 함수를 빠져나가더라도, 그 함수의 스택에 정의되어 있는 모든 객체들은 빠짐없이 소멸자가 호출된다고 하였습니다** (이를 stack unwinding 이라 한다고 했었죠)

예를 들어서 위 포인터 `pa` 의 경우 객체가 아니기 때문에 소멸자가 호출되지 않습니다. 그렇다면 그 대신에, `pa` 를 일반적인 포인터가 아닌, **포인터 '객체' 로 만들어서 자신이 소멸 될 때 자신이 가리키고 있는 데이터도 같이 `delete` 하게 하면 됩니다.** 즉, 자원 (이 경우 메모리) 관리를 스택의 객체 (포인터 객체) 를 통해 수행하게 되는 것입니다.

> 💡 포인터 객체를 스마트 포인터(smart pointer)

unique_ptr

메모리 관련 발생할 수 있는 문제들

- 메모리를 해제 하지 않아서 발생하는 memory leak
- 이미 해제한 메모리에 대한 참조 (ex: double free bug)

unique_ptr 은 특정 객체에 유일한 소유권을 부여하는 포인터 객체 이다. unique_ptr 은 복사할 수 없지만, 소유권 이전은 가능하다. 소유권이 이전된 포인터를 dangling pointer 라고 한다.

▼ 코드

```
#include <iostream>
#include <memory>

class A {
  int *data;

 public:
  A() {
    std::cout << "자원을 획득함!" << std::endl;
    data = new int[100];
  }

  void some() { std::cout << "일반 포인터와 동일하게 사용가능!" <

  ~A() {
    std::cout << "자원을 해제함!" << std::endl;
    delete[] data;
  }
};

void do_something() {
  std::unique_ptr<A> pa(new A());
  std::cout << "pa : ";
  pa->some();

  // pb 에 소유권을 이전.
```

```
  std::unique_ptr<A> pb = std::move(pa);
  std::cout << "pb : ";
  pb->some();
 }

 int main() { do_something(); }
```

정리해보자면,

- `unique_ptr` 은 어떤 객체의 유일한 소유권을 나타내는 포인터 이며, `unique_ptr` 가 소멸될 때, 가리키던 객체 역시 소멸된다.
- 만약에 다른 함수에서 `unique_ptr` 가 소유한 객체에 일시적으로 접근하고 싶다면, get 을 통해 해당 객체의 포인터를 전달하면 된다.
- 만약에 소유권을 이동하고자 한다면, `unique_ptr` 를 move 하면 된다.

unique_ptr 생성

`std::make_unique`

```
#include <iostream>
#include <memory>

class Foo {
  int a, b;

 public:
  Foo(int a, int b) : a(a), b(b) { std::cout << "생성자 호출!" <
  void print() { std::cout << "a : " << a << ", b : " << b <<
  ~Foo() { std::cout << "소멸자 호출!" << std::endl; }
};

int main() {
  auto ptr = std::make_unique<Foo>(3, 5);
  ptr->print();
}
```

`make_unique` 함수는 아예 템플릿 인자로 전달된 클래스의 생성자에 인자들에 직접 완벽한 전달 을 수행합니다.

unique_ptr 를 원소로 가지는 STL 컨테이너

복사 생성자가 없다는 특성

```
#include <iostream>
#include <memory>
#include <vector>

class A {
  int *data;

 public:
  A(int i) {
    std::cout << "자원을 획득함!" << std::endl;
    data = new int[100];
    data[0] = i;
  }

  void some() { std::cout << "일반 포인터와 동일하게 사용가능!" << s

  ~A() {
    std::cout << "자원을 해제함!" << std::endl;
    delete[] data;
  }
};

int main() {
  std::vector<std::unique_ptr<A>> vec;
  std::unique_ptr<A> pa(new A(1));

  vec.push_back(pa);  // ERROR 발생
}
```

삭제된 unique_ptr 의 복사 생성자에 접근하였기 때문에 오류 발생. `vector` 의 push_back 함수는 전달된 인자를 복사해서 집어 넣기 때문에 위와 같은 문제가 발생하게 되는 것.

이를 방지하기 위해서는 명시적으로 `pa` 를 `vector` 안으로 이동 시켜주어야만 한다.

```
vec.push_back(std::move(pa));  // 잘 실행됨
```