class user-define new() 호출 후 생성자에서 예외가 발생했을 때 메모리를 회수하는 방법

작성자: 이정모

일반적인 new()와 delete()를 사용하는 경우 운영체제에게 힙 영역 주소를 할당 받고 반납하는 것이 속도 측면에서 좋지 못하다고 생각하여

ObjectPool을 만들어서 하나의 큰 메모리 chunk를 한번에 할당 받고

object class 내부에서 new()와 delete()를 오버로딩 하는 중이었습니다.

```
void* Grid::operator new(size_t size, ObjectPool<Grid>* pObjectPool)
{
  return pObjectPool->Allocate();
}
```

new() 연산자를 호출할 때 매개변수로 ObjectPool을 넘겨주고

연산자 내부에서는 ObjectPool이 관리하고 있는 사용 가능한 주소를 반환하도록 했습니다.

class user-define new()를 오버로딩 하였다면 이와 짝이 맞는

class user-define delete()를 오버로딩 해야합니다.

그 이유는

https://learn.microsoft.com/en-us/cpp/error-messages/compiler-warnings/compiler-warning-level-1-c4291?view=msvc-170을 보면 알 수 있는데

placement new()를 통해 임의의 주소를 할당 받았고 그 주소에 해당하는 메모리에 생성자를 호출해서 초기화를 진행할텐데

만약 초기화 과정에서 예외가 발생한다면 new()와 짝이 맞는 delete()를 컴파일러가 호출해서 메모리를 회수할 수 있는 기회를 준다는 것입니다.

```
void Grid::operator delete(void* returnAddress, ObjectPool<Grid>* pObjectPool)
{
  pObjectPool->DeAllocate(returnAddress);

  return;
}
```

그래서 delete() 역시 new()와 같은 signature로 오버로딩 했습니다.

class user-define new()와 class user-define delete()의 짝을 맞춰주었기 때문에

예외가 발생하더라도 문제 없이 메모리를 회수할 수 있다고 생각했고

이후 코드를 작성했습니다.

```
IObject* pObj = new(mObjectPool) Grid(); // ok

delete(mObjectPool) pObj; // error
```

new()를 호출할 때 인자로 ObjectPool을 넣었기에

delete()를 호출할 때도 인자로 ObjectPool을 넣었는데 에러가 발생했습니다.

나는 분명 delete()를 오버로딩 했는데 왜 호출이 안되는 거지? 하고 찾아보니

https://stackoverflow.com/questions/11431685/overriding-delete-with-parameters

new()와 동일한 signature의 delete()를 오버로딩 한다는 것은
new() 호출 후 주소 할당 이후에 생성자에서 예외가 발생한 경우에
할당한 주소의 회수를 목적으로 “오직 컴파일러만이 호출해주는 것”이지
프로그래머가 명시적으로 호출하는 것은 불가능한 것이었습니다.

결국 프로그래머가 직접 호출할 수 있는 delete()는
오직 void operator delete(void* addr); 뿐인 것입니다.

그래서 일반적인 delete()를 호출했으나
이번에는 해당 delete 연산자를 찾을 수 없다는 오류가 발생했습니다.

그 이유는 무엇인가 찾아보니
https://en.cppreference.com/w/cpp/memory/new/operator_delete

class 내부에서 delete() 연산자를 하나라도 오버로딩하면
global 범위에 있는 모든 delete() 연산자가 전부 가려진다는 것이었습니다.

즉, ObjectPool을 인자로 받는 delete() 연산자를 하나 오버로딩 했기 때문에
global 범위에 선언되어있는 기본 delete() 연산자가 가려져서 찾지 못한 것이었습니다.

그래서 기본 delete() 연산자도 오버로딩 했습니다.

```
void Grid::operator delete(void* returnAddress)
{
  return;
}
```

new() 호출 후 주소를 할당 받은 뒤 생성자에서 예외가 발생한 경우

컴파일러가 ObjectPool을 인자로 받는 delete()를 호출해주면 주소를 다시 회수할 수 있고

만약 정상적으로 초기화가 진행되고 엔진 사용자로부터 정상적으로 반납이 되면

```
template<typename ObjectType>
inline void Factory<ObjectType>::ReturnObject(IObject* pObj)
{
  pObj->Finalize();

  mObjectPool->DeAllocate(pObj);

  delete pObj;
}
```

factory의 ReturnObject() 함수에서 주소를 회수하도록 하였습니다.

마무리하자면

default new()/delete()를 호출하여 운영체제에게 매번 주소 할당/반납을 요청하는 시간을 줄이기 위해

class user-define new()와 class user-define delete()를 오버로딩 했고

class user-define delete()의 경우 생성자에서 예외가 발생한 경우에

오직 컴파일러만이 호출 가능한 연산자였고

그렇다면 기본 delete()를 사용해야 하는데

class 내부에서 프로그래머가 직접 delete() 연산자를 단 하나라도 오버로딩하는 순간

global 영역에 있는 모든 delete() 가 가려지기 때문에

기본 delete() 또한 class 내부에 오버로딩 했습니다.

이상입니다.