MemoryManager

0.21

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

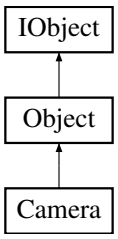
Class Documentation

4.1 Camera Class Reference

크기가 48인 테스트용 클래스

`#include <Camera.h>`

Inheritance diagram for Camera:



Public Member Functions

- **Camera** (const char *pName="Camera")
  Camera *(p. 7) 생성자 @datails 부모 클래스 생성자에 이름("Camera") 전달 *.
- virtual ~**Camera** ()
  Camera *(p. 7) 소멸자 *.
- void **run** ()
  Camera *(p. 7) 실행 함수 @datails data배열에 0~4까지(NUM_CAMERAS만큼)의 값을 저장하고 합계 출력 *.

Public Member Functions inherited from Object

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Additional Inherited Members

Static Public Attributes inherited from Object

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.1.1 Detailed Description

크기가 48인 테스트용 클래스

* *

생성자로 이름 저장, 배열에 0~4까지의 값을 저장하고 출력 *

Date

2024-05-19 *

Version

0.21 *

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Camera()

```
Camera::Camera (
            const char * pName = "Camera" )  [inline]
```

Camera (p. 7) 생성자 @datails 부모 클래스 생성자에 이름("Camera") 전달 *.

*

Date

2024-05-21 *

Version

0.21 *

4.1.2.2 ~Camera()

```
virtual Camera::~Camera ( )  [inline], [virtual]
```

Camera (p. 7) 소멸자 *.

*

Date

2024-05-21 *

Version

0.21 *

4.1.3 Member Function Documentation

4.1.3.1 run()

```
void Camera::run ( )  [inline]
```

Camera (p. 7) 실행 함수 @datails data배열에 0~4까지(NUM_CAMERAS만큼)의 값을 저장하고 합계 출력 *.

* *

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following file:

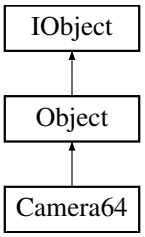
- Camera.h

4.2 Camera64 Class Reference

크기가 64인 테스트용 클래스

```
#include <Camera64.h>
```

Inheritance diagram for Camera64:



Public Member Functions

- **Camera64** (const char *pName="Camera64")
  Camera64 *(p. 9) 생성자*
- virtual ~**Camera64** ()
  Camera64 *(p. 9) 소멸자* *.
- void **run** ()
  Camera64 *(p. 9) 실행 함수*

Public Member Functions inherited from Object

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Additional Inherited Members

Static Public Attributes inherited from Object

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.2.1 Detailed Description

크기가 64인 테스트용 클래스

* *

생성자로 이름 저장, 배열 0~4까지의 값을 저장하고 출력 *

Date

2024-05-19 *

Version

0.21 *

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Camera64()

```
Camera64::Camera64 (
            const char * pName = "Camera64" )  [inline]
```

Camera64 (p. 9) 생성자

* *

부모 클래스 생성자에 이름("Camera64") 전달, id 값 입력 *

Date

2024-05-21 *

Version

0.21 *

4.2.2.2 ~Camera64()

```
virtual Camera64::~Camera64 ( )  [inline], [virtual]
```

Camera64 (p. 9) 소멸자 *.

*

Date

2024-05-21 *

Version

0.21 *

4.2.3 Member Function Documentation

4.2.3.1 run()

```
void Camera64::run ( )  [inline]
```

Camera64 (p. 9) 실행 함수

* *

data배열에 0~4까지(NUM_CAMERAS만큼)의 값을 저장하고 합계 출력 *

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following file:

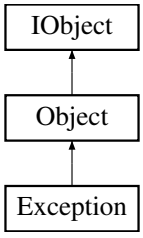
- Camera64.h

4.3 Exception Class Reference

예외 클래스

`#include <Exception.h>`

Inheritance diagram for Exception:



Public Types

- enum class **ECode** { **eOutOfMemory** , **eLengthECode** }
  예외 코드 집합

Public Member Functions

- **Exception** (**ECode** eCode, const char *pNameObject, const char *pNameClass="Exception")
  예외 생성자
- virtual ~**Exception** ()
  소멸자
- **ECode** **getECode** () const
  예외 코드 반환
- const char * **getNameObject** () const
  오류가 발생한 객체 이름 반환
- const char * **getNameClass** () const
  예외 클래스 이름 반환
- const char * **getECodeAsString** () const
  예외 코드를 문자열로 변환
- void **println** ()
  *예외 정보를 콘솔에 출력 **

Public Member Functions inherited from Object

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Additional Inherited Members

Static Public Attributes inherited from Object

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.3.1 Detailed Description

예외 클래스

* *

예외 정보 저장 및 출력 클래스 *

Date

2024-05-19 *

Version

0.21 *

4.3.2 Member Enumeration Documentation

4.3.2.1 ECode

```
enum class  Exception::ECode  [strong]
```

예외 코드 집합

* *

예외 종류를 모아놓은 상수 집합 *

Date

2024-05-19 *

Version

0.21 *

4.3.3 Constructor & Destructor Documentation

4.3.3.1 Exception()

```
Exception::Exception (
             ECode eCode,
            const char * pNameObject,
            const char * pNameClass = "Exception" )  [inline]
```

예외 생성자

* *

예외를 발생 시 예외 코드, 객체 이름, 클래스 이름을 저장

Parameters

eCode	예외 코드
pNameObject	예외가 발생한 객체 이름
pNameClass	예외 클래스명 (기본값: "Exception") *

Date

2024-05-21 *

Version

0.21 *

4.3.3.2 ~Exception()

```
virtual Exception::~Exception ( )  [inline], [virtual]
```

소멸자

* *

문자열 배열 삭제 *

Date

2024-05-19 *

Version

0.21 *

4.3.4 Member Function Documentation

4.3.4.1 getECode()

```
ECode Exception::getECode ( ) const  [inline]
```

예외 코드 반환

* *

Returns

예외 코드 *

Date

2024-05-19 *

Version

0.21 *

4.3.4.2 getECodeAsString()

```
const char * Exception::getECodeAsString ( ) const  [inline]
```

예외 코드를 문자열로 변환

* *

예외 코드를 switch문을 이용해 문자열로 변환

Returns

예외 메시지 문자열 *

Date

2024-05-19 *

Version

0.21 *

4.3.4.3 getNameClass()

```
const char * Exception::getNameClass ( ) const  [inline]
```

예외 클래스 이름 반환

* *

Returns

클래스 이름 문자열 *

Date

2024-05-19 *

Version

0.21 *

4.3.4.4 getNameObject()

```
const char * Exception::getNameObject ( ) const  [inline]
```

오류가 발생한 객체 이름 반환

* *

Returns

객체 이름 문자열 *

Date

2024-05-19 *

Version

0.21 *

4.3.4.5 println()

```
void Exception::println ( )  [inline]
```

예외 정보를 콘솔에 출력 *

* *

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following file:

- Exception.h

4.4 ExceptionManager Class Reference

예외처리 매니저 클래스

```
#include <ExceptionManager.h>
```

Static Public Member Functions

- static void **logException** (const **Exception** &ex)
  예외를 배열에 저장 및 출력
- static void **printAllExceptions** ()
  배열에 저장된 예외 출력

4.4.1 Detailed Description

예외처리 매니저 클래스

* *

예외를 배열에 저장하고 출력 *

Date

2024-05-19 *

Version

0.21 *

4.4.2 Member Function Documentation

4.4.2.1 logException()

```
static void ExceptionManager::logException (
            const  Exception & ex )  [inline], [static]
```

예외를 배열에 저장 및 출력

* *

예외 객체의 정보를 문자열로 만들어 배열에 저장 및 콘솔에 출력

Parameters

ex	예외 객체 주소 *

Date

2024-05-19 *

Version

0.21 *

4.4.2.2 printAllExceptions()

```
static void ExceptionManager::printAllExceptions ( )  [inline], [static]
```

배열에 저장된 예외 출력

* *

배열에 저장된 예외를 하나씩 꺼내서 출력 *

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following files:

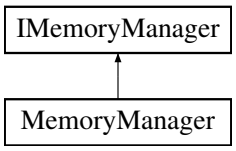
- ExceptionManager.h
- ExceptionManager.cpp

4.5 IMemoryManager Class Reference

메모리 매니저 인터페이스

```
#include <IMemoryManager.h>
```

Inheritance diagram for IMemoryManager:



Public Member Functions

- virtual void * **allocate** (size_t sizeSlot, const char *pName)=0
  메모리 할당
- virtual void **dellocate** (void *pObject)=0
  메모리 해제
- virtual **ISlotIndex** * **findASlotIndex** (void *pObject)=0
  SlotIndex *(p. 46) 검색*
- virtual **PageIndex** **findAPageIndex** (void *pObject)=0
  PageIndex *(p. 35) 검색*
- virtual void **showStatus** ()=0
  메모리 할당 상태 출력

4.5.1 Detailed Description

메모리 매니저 인터페이스

* *

메모리 할당/해제, slot 및 pageIndex 검색, 상태 출력을 위한 인터페이스 *

Date

2024-05-02 *

Version

0.21 *

4.5.2 Member Function Documentation

4.5.2.1 allocate()

```
virtual void * IMemoryManager::allocate (
            size_t sizeSlot,
            const char * pName )  [pure virtual]
```

메모리 할당

* *

슬롯크기, 이름을 받아 메모리 할당

Parameters

sizeSlot	슬롯 크기
pName	할당할 객체 이름 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **MemoryManager** (p. 27).

4.5.2.2 dellocate()

```
virtual void IMemoryManager::dellocate (
            void * pObject )  [pure virtual]
```

메모리 해제

* *

할당 해제할 객체의 포인터를 받아 메모리 해제

Parameters

pObject	할당 해제할 객체의 포인터 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **MemoryManager** (p. 27).

4.5.2.3 findAPageIndex()

```
virtual  PageIndex IMemoryManager::findAPageIndex (
            void * pObject )  [pure virtual]
```

PageIndex (p. 35) 검색

* *

검색할 객체의 포인터를 받아 **PageIndex** (p. 35) 검색

Parameters

pObject	검색할 객체의 포인터 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **MemoryManager** (p. 28).

4.5.2.4 findASlotIndex()

```
virtual ISlotIndex * IMemoryManager::findASlotIndex (
            void * pObject )  [pure virtual]
```

SlotIndex (p. 46) 검색

* *

검색할 객체의 주소를 받아 **SlotIndex** (p. 46) 검색

Parameters

pObject	검색할 객체의 주소 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **MemoryManager** (p. 28).

4.5.2.5 showStatus()

```
virtual void IMemoryManager::showStatus ( )  [pure virtual]
```

메모리 할당 상태 출력

* *

현재 메모리 할당 상태를 콘솔에 출력 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **MemoryManager** (p. 30).

The documentation for this class was generated from the following file:

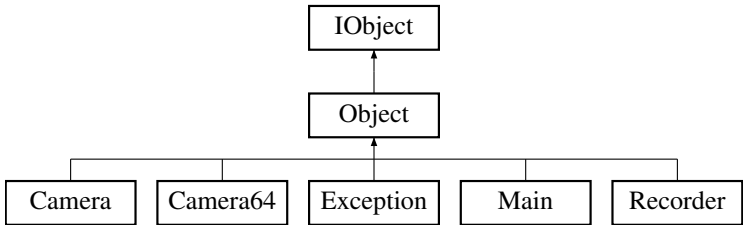
- IMemoryManager.h

4.6 IObject Class Reference

Object (p. 31) 클래스의 인터페이스 클래스

`#include <IObject.h>`

Inheritance diagram for IObject:



Public Member Functions

- virtual int **getId** ()=0
  객체의 인터페이스 클래스 *
- virtual char * **getNameClass** ()=0
  객체의 인터페이스 클래스 *
- virtual char * **getNameObject** ()=0
  객체의 인터페이스 클래스 *

4.6.1 Detailed Description

Object (p. 31) 클래스의 인터페이스 클래스

* *

객체의 아이디, 이름, 객체 이름을 반환한다. *

Date

2024-05-02 *

Version

0.21 *

4.6.2 Member Function Documentation

4.6.2.1 getId()

```
virtual int IObject::getId ( )  [pure virtual]
```

객체의 인터페이스 클래스 *

* *

Date

2024-05-02 *

Version

0.21 *

Implemented in **Object** (p. 32).

4.6.2.2 getNameClass()

```
virtual char * IObject::getNameClass ( )  [pure virtual]
```

객체의 인터페이스 클래스 *

* *

Date

2024-05-02 *

Version

0.21 *

Implemented in **Object** (p. 32).

4.6.2.3 getNameObject()

```
virtual char * IObject::getNameObject ( )  [pure virtual]
```

객체의 인터페이스 클래스 *

* *

Date

2024-05-02 *

Version

0.21 *

Implemented in **Object** (p. 33).

The documentation for this class was generated from the following file:

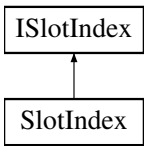
- IObject.h

4.7 ISlotIndex Class Reference

SlotIndex (p. 46) 클래스의 인터페이스 클래스 *.

`#include <ISlotIndex.h>`

Inheritance diagram for ISlotIndex:



Public Member Functions

- virtual char * **getNameObject** ()=0
  슬롯에 할당된 객체의 이름을 반환하는 함수

4.7.1 Detailed Description

SlotIndex (p. 46) 클래스의 인터페이스 클래스 *.

* *

Date

2024-05-19 *

Version

0.21 *

4.7.2 Member Function Documentation

4.7.2.1 getNameObject()

`virtual char * ISlotIndex::getNameObject ( ) [pure virtual]`

슬롯에 할당된 객체의 이름을 반환하는 함수

* *

Returns

슬롯에 할당된 객체의 이름 *

Date

2024-05-19 *

Version

0.21 *

Implemented in **SlotIndex** (p. 48).

The documentation for this class was generated from the following file:

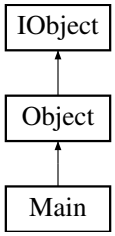
- ISlotIndex.h

4.8 Main Class Reference

메인 클래스 *

`#include <Main.h>`

Inheritance diagram for Main:



Public Member Functions

- **Main** (const char *pName="Main")
- void **initialize** ()
- void **finalize** ()
- void **run** ()

Public Member Functions inherited from Object

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Additional Inherited Members

Static Public Attributes inherited from Object

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.8.1 Detailed Description

메인 클래스 *

* *

Date

2024-05-02 *

Version

0.21 *

The documentation for this class was generated from the following file:

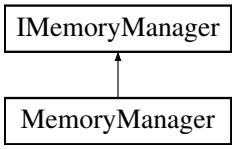
- Main.h

4.9 MemoryManager Class Reference

메모리 할당, 해제 관리 클래스

`#include <MemoryManager.h>`

Inheritance diagram for MemoryManager:



Public Member Functions

- void * **operator new** (size_t size)
  *malloc함수를 통해 **MemoryManager** (p. 25) 생성*
- void **operator delete** (void *pObject)
  *free함수를 통해 **MemoryManager** (p. 25) 할당 해제*
- **MemoryManager** (char *pBuffer, size_t sizeBuffer, size_t sizePage)
  *메모리 관리 객체 생성자 *
- virtual ~**MemoryManager** ()
  *메모리 관리 객체 소멸자 *
- void * **allocate** (size_t sizeMemory, const char *pName)
  *메모리에 객체를 할당하는 함수 *
- void **dellocate** (void *pObject)
  *객체를 할당 해제하는 함수 *
- **SlotIndex** * **findASlotIndex** (void *pObject)
  객체 주소에 해당하는 SlotIndex를 찾는 함수
- **PageIndex** **findAPageIndex** (void *pObject)
  객체 주소에 해당하는 PageIndex를 찾는 함수
- void **showStatus** ()
  메모리 할당 상태 출력 함수

4.9.1 Detailed Description

메모리 할당, 해제 관리 클래스

* *

메모리 공간을 페이지 단위로 나누어 PageIndex배열을 생성하고 객체를 할당, 해제한다. *

Date

2024-05-19 *

Version

0.21 *

4.9.2 Constructor & Destructor Documentation

4.9.2.1 MemoryManager()

```
MemoryManager::MemoryManager (
            char * pBuffer,
            size_t sizeBuffer,
            size_t sizePage )  [inline]
```

메모리 관리 객체 생성자 *

* *

메모리 공간을 페이지 크기 만큼 나누어 PageIndex배열을 생성하고 초기화한다.

Parameters

pBuffer	메모리 시작주소
sizeBuffer	메모리 크기
sizePage	한 페이지 크기 *

Date

2024-05-19 *

Version

0.21 *

4.9.2.2 ~MemoryManager()

```
virtual MemoryManager::~MemoryManager ( )  [inline], [virtual]
```

메모리 관리 객체 소멸자 *

* *

Date

2024-05-19 *

Version

0.21 *

4.9.3 Member Function Documentation

4.9.3.1 allocate()

```
void * MemoryManager::allocate (
            size_t sizeMemory,
            const char * pName )  [inline], [virtual]
```

메모리에 객체를 할당하는 함수 *

* *

객체 크기를 16배수의 slot으로 일반화하고 할당할 PageIndex를 뒤에서부터 찾아 할당한다.

Parameters

sizeMemory	할당할 객체 크기
pName	할당할 객체 이름

Returns

할당된 객체의 메모리 주소 *

Date

2024-05-19 *

Version

0.21 *

Implements **IMemoryManager** (p. 18).

4.9.3.2 dellocate()

```
void MemoryManager::dellocate (
            void * pObject )  [inline], [virtual]
```

객체를 할당 해제하는 함수 *

* *

객체 주소에 해당하는 PageIndex를 찾아 객체를 할당 해제 후 빈 페이지를 정리한다.

Parameters

pObject	할당 해제할 객체 주소 *

Date

2024-05-19 *

Version

0.21 *

Implements **IMemoryManager** (p. 19).

4.9.3.3 findAPageIndex()

```
PageIndex MemoryManager::findAPageIndex (
            void * pObject )  [inline], [virtual]
```

객체 주소에 해당하는 PageIndex를 찾는 함수

* *

객체 주소를 받아 페이지 번호를 찾고 해당하는 PageIndex를 반환한다.

Parameters

pObject	객체 주소

Returns

객체 주소에 해당하는 **PageIndex** (p. 35) *

Date

2024-05-19 *

Version

0.21 *

Implements **IMemoryManager** (p. 19).

4.9.3.4 findASlotIndex()

```
SlotIndex * MemoryManager::findASlotIndex (
            void * pObject )  [inline], [virtual]
```

객체 주소에 해당하는 SlotIndex를 찾는 함수

* *

객체 주소에 해당하는 PageIndex를 찾고 PageIndex의 검색함수를 호출해 SlotIndex를 찾고 반환한다.

Parameters

pObject	객체 주소

Returns

객체 주소에 해당하는 **SlotIndex** (p. 46) *

Date

2024-05-19 *

Version

0.21 *

Implements **IMemoryManager** (p. 20).

4.9.3.5 operator delete()

```
void MemoryManager::operator delete (
            void * pObject )  [inline]
```

free함수를 통해 **MemoryManager** (p. 25) 할당 해제

* *

Parameters

pObject	**MemoryManager** (p. 25) 메모리 주소 *

Date

2024-05-21 *

Version

0.21 *

4.9.3.6 operator new()

```
void * MemoryManager::operator new (
            size_t size )  [inline]
```

malloc함수를 통해 **MemoryManager** (p. 25) 생성

* *

Parameters

size	**MemoryManager** (p. 25) 크기

Returns

MemoryManager (p. 25) 메모리 주소 *

Date

2024-05-21 *

Version

0.21 *

4.9.3.7 showStatus()

```
void MemoryManager::showStatus ( )  [inline], [virtual]
```

메모리 할당 상태 출력 함수

* *

반복문으로 PageIndex배열의 할당 상태(슬롯 크기, 연결된 페이지)를 출력, 페이지 할당 상태 출력함수 호출. *

Date

2024-05-02 *

Version

0.21 *

Implements **IMemoryManager** (p. 20).

The documentation for this class was generated from the following file:

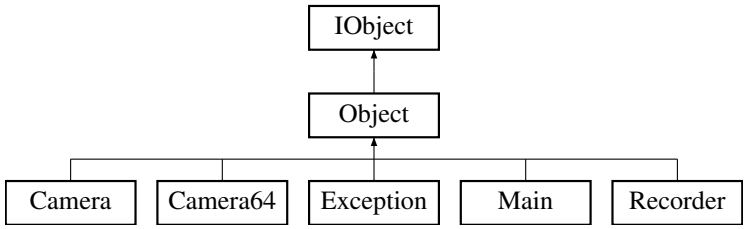
- MemoryManager.h

4.10 Object Class Reference

메모리에 할당되는 객체를 만드는 클래스의 부모 클래스

`#include <Object.h>`

Inheritance diagram for Object:



Public Member Functions

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Static Public Attributes

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.10.1 Detailed Description

메모리에 할당되는 객체를 만드는 클래스의 부모 클래스

* *

객체의 생성과 소멸, 객체의 정보를 반환하는 함수를 가지고 있다. *

Date

2024-05-19 *

Version

0.21 *

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Object()

```
Object::Object (
            const char * pClassName = "Object" )  [inline]
```

```
  @brief 객체 생성자
  @details 객체 이름을 받아서 저장하고, 아이디를 부여한다.
  @param pClassName 객체 이름(기본값: "Object")
```

*

Date

2024-05-19 *

Version

0.21 *

4.10.2.2 ~Object()

```
virtual Object::~Object ( )  [inline], [virtual]
```

```
  @brief 객체 소멸자
```

*

Date

2024-05-19 *

Version

0.21 *

4.10.3 Member Function Documentation

4.10.3.1 getId()

```
int Object::getId ( )  [inline], [virtual]
```

```
  @brief 객체 아이디를 반환하는 함수
  @return 객체 아이디
```

*

Date

2024-05-19 *

Version

0.21 *

Implements **IObject** (p. 22).

4.10.3.2 getNameClass()

```
char * Object::getNameClass ( )  [inline], [virtual]
```

@brief 객체를 생성할 때 사용한 클래스 이름을 반환하는 함수
@return 객체를 생성할 때 사용한 클래스 이름

*

Date

2024-05-19 *

Version

0.21 *

Implements **IObject** (p. 22).

4.10.3.3 getNameObject()

```
char * Object::getNameObject ( )  [inline], [virtual]
```

@brief 객체 이름을 반환하는 함수
@return 객체 이름

*

Date

2024-05-19 *

Version

0.21 *

Implements **IObject** (p. 22).

4.10.3.4 getSizeSlot()

```
size_t Object::getSizeSlot ( )  [inline]
```

@brief 객체가 할당된 Slot의 크기를 반환하는 함수
@return 객체가 할당된 Slot의 크기

*

Date

2024-05-19 *

Version

0.21 *

4.10.3.5 operator delete() [1/2]

```
void Object::operator delete (
            void * pAllocated )  [inline]
```

@brief 객체 삭제시 할당 해제하는 함수
@details 객체 주소를 메모리매니저에게 주면서 할당 해제한다.
@param pAllocated 할당 해제할 객체의 주소

*

Date

2024-05-19 *

Version

0.21 *

4.10.3.6 operator delete() [2/2]

```
void Object::operator delete (
            void * pObject,
            const char * pName )  [inline]
```

@brief 객체 삭제시 할당 해제하는 함수
@details 객체 주소를 메모리매니저에게 주면서 할당 해제한다.
@param pAllocated 할당 해제할 객체의 주소
@param pName 할당 해제할 객체의 이름

*

Date

2024-05-19 *

Version

0.21 *

4.10.3.7 operator new()

```
void * Object::operator new (
            size_t size,
            const char * pName )  [inline]
```

```
@brief 객체 생성시 메모리에 할당하는 함수
@details 객체를 생성할 때 메모리매니저에게 이름과 크기를 주면서 할당한다.
@param size 할당할 객체의 크기
@param pName 할당할 객체의 이름
@return 할당된 메모리의 포인터
```

*

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following files:

- Object.h
- Object.cpp

4.11 PageIndex Class Reference

페이지 내 Slot 할당, 해제 클래스

`#include <PageIndex.h>`

Public Member Functions

- void * **operator new** (size_t size)
  malloc함수를 통해 ***PageIndex*** *(p. 35) 생성*
- void **operator delete** (void *pObject)
  free함수를 통해 ***PageIndex*** *(p. 35) 할당 해제*
- int **getId** ()
  페이지 아이디를 반환하는 함수
- void **setPPage** (char *pPage)
  PageIndex의 시작주소를 반환하는 함수
- void **setSizePage** (size_t sizePage)
  페이지 크기를 지정하는 함수
- void **setNumConsecutivePages** (int numConsecutivePages)
  페이지 뒤에 연속된 비어있는 페이지 갯수 값을 지정하는 함수
- int **getNumConsecutivePages** ()
  페이지 뒤에 연속된 비어있는 페이지 갯수 값을 반환하는 함수

- bool **isAllocated** ()
  페이지 할당 상태를 반환하는 함수
- void **setBAllocated** (bool bAllocated)
  페이지 할당 상태를 지정하는 함수
- size_t **getSizeSlot** ()
  페이지의 슬롯 크기를 반환하는 함수
- **PageIndex** ()
  PageIndex *(p. 35) 생성자*
- ~**PageIndex** ()
  페이지 소멸자 ∗
- void **initialize** (size_t sizeSlot)
  페이지 초기화 함수
- void **finalize** ()
  페이지 할당 해제 함수 ∗
- void ∗ **allocate** (size_t size, const char ∗pNameObject)
  슬롯 할당 함수 ∗
- bool **dellocate** (void ∗pMemory)
  슬롯 할당 해제 함수 ∗
- **SlotIndex** ∗ **findASlotIndex** (void ∗pMemory)
  메모리 주소에 해당하는 SlotIndex를 찾는 함수
- void **showStatus** ()
  페이지 내부 할당 상태 출력 함수

4.11.1 Detailed Description

페이지 내 Slot 할당, 해제 클래스

∗ ∗

SlotIndex배열에 SlotIndex객체를 할당, 해제 및 할당 상태를 출력할 수 있다. ∗

Date

2024-05-02 ∗

Version

0.21 ∗

4.11.2 Constructor & Destructor Documentation

4.11.2.1 PageIndex()

```
PageIndex::PageIndex ( )  [inline]
```

PageIndex (p. 35) 생성자

∗ ∗

객체 아이디를 설정하고 나머지 값은 정의되지 않은 값으로 초기화한다. ∗

Date

2024-05-19 ∗

Version

0.21 ∗

4.11.2.2 ~PageIndex()

```
PageIndex::~PageIndex ( )  [inline]
```

페이지 소멸자 *

* *

Date

2024-05-19 *

Version

0.21 *

4.11.3 Member Function Documentation

4.11.3.1 allocate()

```
void * PageIndex::allocate (
            size_t size,
            const char * pNameObject )  [inline]
```

슬롯 할당 함수 *

* *

할당 가능한 슬롯을 찾아 객체 이름을 설정하고 할당 상태를 변경한다.

Parameters

size	할당할 슬롯 크기
pNameObject	할당할 객체 이름

Returns

할당된 슬롯의 메모리 주소 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.2 dellocate()

```
bool PageIndex::dellocate (
            void * pMemory )  [inline]
```

슬롯 할당 해제 함수 *

* *

Slot메모리 주소를 받아 해당 **SlotIndex** (p. 46) 할당 상태를 변경하고 할당 가능한 슬롯의 개수를 증가시킨다.

Parameters

pMemory	할당 해제할 Slot의 메모리 주소

Returns

페이지 내 모든 슬롯이 비어있는지 여부 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.3 finalize()

```
void PageIndex::finalize ( )  [inline]
```

페이지 할당 해제 함수 *

* *

Date

2024-05-19 *

Version

0.21 *

4.11.3.4 findASlotIndex()

```
SlotIndex * PageIndex::findASlotIndex (
            void * pMemory )  [inline]
```

메모리 주소에 해당하는 SlotIndex를 찾는 함수

* *

메모리 주소를 받아 Slot 번호를 찾고 해당하는 SlotIndex를 반환한다.

Parameters

pMemory	메모리 주소

Returns

메모리 주소에 해당하는 **SlotIndex** (p. 46) *

Date

2024-05-19 *

Version

0.21 *

4.11.3.5 getId()

```
int PageIndex::getId ( )  [inline]
```

페이지 아이디를 반환하는 함수

* *

Returns

PageIndex (p. 35) 아이디 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.6 getNumConsecutivePages()

```
int PageIndex::getNumConsecutivePages ( )  [inline]
```

페이지 뒤에 연속된 비어있는 페이지 갯수 값을 반환하는 함수

* *

Returns

페이지 뒤에 연속된 비어있는 페이지 갯수 값 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.7 getSizeSlot()

```
size_t PageIndex::getSizeSlot ( )  [inline]
```

페이지의 슬롯 크기를 반환하는 함수

* *

Returns

페이지의 슬롯 크기 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.8 initialize()

```
void PageIndex::initialize (
            size_t sizeSlot )  [inline]
```

페이지 초기화 함수

* *

페이지 내 슬롯의 크기를 설정하고 슬롯 인덱스 배열을 할당한다.

Parameters

sizeSlot	슬롯 크기 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.9 isAllocated()

```
bool PageIndex::isAllocated ( )  [inline]
```

페이지 할당 상태를 반환하는 함수

* *

Returns

페이지 할당 상태 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.10 operator delete()

```
void PageIndex::operator delete (
            void * pObject )  [inline]
```

free함수를 통해 **PageIndex** (p. 35) 할당 해제

* *

Parameters

pObject	**PageIndex** (p. 35) 메모리 주소 *

Date

2024-05-21 *

Version

0.21 *

4.11.3.11 operator new()

```
void * PageIndex::operator new (
            size_t size )  [inline]
```

malloc함수를 통해 **PageIndex** (p. 35) 생성

* *

Parameters

size	**PageIndex** (p. 35) 크기

Returns

PageIndex (p. 35) 메모리 주소 *

Date

2024-05-21 *

Version

0.21 *

4.11.3.12 setBAllocated()

```
void PageIndex::setBAllocated (
            bool bAllocated )  [inline]
```

페이지 할당 상태를 지정하는 함수

* *

Parameters

bAllocated	페이지 할당 상태 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.13 setNumConsecutivePages()

```
void PageIndex::setNumConsecutivePages (
            int numConsecutivePages )  [inline]
```

페이지 뒤에 연속된 비어있는 페이지 갯수 값을 지정하는 함수

* *

Parameters

numConsecutivePages	페이지 뒤에 연속된 비어있는 페이지 갯수 값 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.14 setPPage()

```
void PageIndex::setPPage (
            char * pPage )  [inline]
```

PageIndex의 시작주소를 반환하는 함수

* *

Returns

PageIndex (p. 35) 시작주소 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.15 setSizePage()

```
void PageIndex::setSizePage (
            size_t sizePage )  [inline]
```

페이지 크기를 지정하는 함수

* *

Parameters

sizePage	페이지 크기 *

Date

2024-05-19 *

Version

0.21 *

4.11.3.16 showStatus()

```
void PageIndex::showStatus ( )  [inline]
```

페이지 내부 할당 상태 출력 함수

* *

반복문으로 SlotIndex배열의 할당 상태(클래스 및 객체 이름, 객체 아이디)를 출력 *

Date

2024-05-02 *

Version

0.21 *

The documentation for this class was generated from the following files:

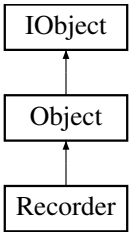
- PageIndex.h
- PageIndex.cpp

4.12 Recorder Class Reference

크기가 104인 테스트용 클래스

```
#include <Recorder.h>
```

Inheritance diagram for Recorder:



Public Member Functions

- **Recorder** (const char *pName="Recorder")

  Recorder *(p. 44) 생성자 @datails 부모 클래스 생성자에 이름("Recorder") 전달 *.*
- virtual ~**Recorder** ()

  Recorder *(p. 44) 소멸자 *.*
- void **run** ()

  Recorder *(p. 44) 실행 함수 @datails data배열에 0~19까지(NUM_RECORDERS만큼)의 값을 저장하고 합계 출력 *.*

Public Member Functions inherited from Object

- void * **operator new** (size_t size, const char *pName)
- void **operator delete** (void *pAllocated)
- void **operator delete** (void *pObject, const char *pName)
- **Object** (const char *pClassName="Object")
- virtual ~**Object** ()
- int **getId** ()
- char * **getNameClass** ()
- char * **getNameObject** ()
- size_t **getSizeSlot** ()

Additional Inherited Members

Static Public Attributes inherited from Object

- static int **s_counterId** = 0
- static **IMemoryManager** * **s_pMemoryManager** = nullptr

4.12.1 Detailed Description

크기가 104인 테스트용 클래스

* *

생성자로 이름 저장, 배열에 0~20까지의 값을 저장하고 출력 *

Date

2024-05-19 *

Version

0.21 *

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Recorder()

```
Recorder::Recorder (
            const char * pName = "Recorder" )  [inline]
```

Recorder (p. 44) 생성자 @datails 부모 클래스 생성자에 이름("Recorder") 전달 *.

*

Date

2024-05-21 *

Version

0.21 *

4.12.2.2 ~Recorder()

```
virtual Recorder::~Recorder ( )  [inline], [virtual]
```

Recorder (p. 44) 소멸자 *.

*

Date

2024-05-21 *

Version

0.21 *

4.12.3 Member Function Documentation

4.12.3.1 run()

```
void Recorder::run ( )  [inline]
```

Recorder (p. 44) 실행 함수 @datails data배열에 0~19까지(NUM_RECORDERS만큼)의 값을 저장하고 합계 출력 *.

*

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following file:

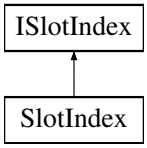
- Recorder.h

4.13 SlotIndex Class Reference

페이지 내부에 할당되는 슬롯의 정보를 저장하는 클래스

`#include <SlotIndex.h>`

Inheritance diagram for SlotIndex:



Public Member Functions

- **SlotIndex** ()
  슬롯 생성자
- ∼**SlotIndex** ()
  슬롯 소멸자 ∗
- void **setPMemory** (void ∗pMemory)
  슬롯의 시작 주소를 설정하는 함수
- void ∗ **getPMemory** ()
  슬롯의 시작 주소를 반환하는 함수
- void **setNameObject** (const char ∗pNameObject)
  슬롯의 할당된 객체의 이름을 설정하는 함수
- char ∗ **getNameObject** ()
  슬롯에 할당된 객체의 이름을 반환하는 함수
- void **setBAllocated** (bool bAllocated)
  슬롯의 할당여부를 설정하는 함수
- bool **isAllocated** ()
  슬롯의 할당여부를 반환하는 함수

4.13.1 Detailed Description

페이지 내부에 할당되는 슬롯의 정보를 저장하는 클래스

∗ ∗

슬롯의 주소, 이름, 할당 상태를 저장하고 반환한다. ∗

Date

2024-05-19 ∗

Version

0.21 ∗

4.13.2 Constructor & Destructor Documentation

4.13.2.1 SlotIndex()

```
SlotIndex::SlotIndex ( )  [inline]
```

슬롯 생성자

* *

슬롯의 주소, 이름, 할당 상태를 값이 없는 상태로 초기화한다. *

Date

2024-05-19 *

Version

0.21 *

4.13.2.2 ~SlotIndex()

```
SlotIndex::~SlotIndex ( )  [inline]
```

슬롯 소멸자 *

* *

Date

2024-05-19 *

Version

0.21 *

4.13.3 Member Function Documentation

4.13.3.1 getNameObject()

```
char * SlotIndex::getNameObject ( )  [inline], [virtual]
```

슬롯에 할당된 객체의 이름을 반환하는 함수

* *

Returns

슬롯에 할당된 객체의 이름 *

Date

2024-05-19 *

Version

0.21 *

Implements **ISlotIndex** (p. 23).

4.13.3.2 getPMemory()

```
void * SlotIndex::getPMemory ( )  [inline]
```

슬롯의 시작 주소를 반환하는 함수

* *

Returns

슬롯의 시작 주소 *

Date

2024-05-19 *

Version

0.21 *

4.13.3.3 isAllocated()

```
bool SlotIndex::isAllocated ( )  [inline]
```

슬롯의 할당여부를 반환하는 함수

* *

Returns

슬롯의 할당여부 *

Date

2024-05-19 *

Version

0.21 *

4.13.3.4 setBAllocated()

```
void SlotIndex::setBAllocated (
            bool bAllocated )  [inline]
```

슬롯의 할당여부를 설정하는 함수

* *

Parameters

bAllocated	슬롯의 할당여부 *

Date

2024-05-19 *

Version

0.21 *

4.13.3.5 setNameObject()

```
void SlotIndex::setNameObject (
            const char * pNameObject )  [inline]
```

슬롯의 할당된 객체의 이름을 설정하는 함수

* *

Parameters

pNameObject	슬롯의 할당된 객체의 이름 *

Date

2024-05-19 *

Version

0.21 *

4.13.3.6 setPMemory()

```
void SlotIndex::setPMemory (
            void * pMemory )  [inline]
```

슬롯의 시작 주소를 설정하는 함수

* *

Parameters

pMemory	슬롯의 시작 주소 *

Date

2024-05-19 *

Version

0.21 *

The documentation for this class was generated from the following file:

- SlotIndex.h

Chapter 5

File Documentation

5.1 Camera.h

```
00001 #pragma once
00002 #include "Object.h"
00003
00004 #define NUM_CAMERAS 5
00005
00014 class Camera : public Object {
00015 private:
00016     int data[NUM_CAMERAS] = {};
00017 public:
00026     Camera(const char* pName = "Camera") :
00027         Object(pName)
00028     {
00029     }
00030
00038     virtual ~Camera() {
00039     }
00040
00049     void run() {
00050         for (int i = 0; i < NUM_CAMERAS; i++) {
00051             this->data[i] = i;
00052         }
00053         int sum = 0;
00054         for (int i = 0; i < NUM_CAMERAS; i++) {
00055             sum = sum + this->data[i];
00056         }
00057         printf("%s::%s(%d)::run()=%d\n", this->getNameClass(), this->getNameObject(), this->getId(),
      sum);
00058     }
00059 };
00060
```

5.2 Camera64.h

```
00001 #pragma once
00002 #include "Object.h"
00003
00004 #define NUM_CAMERAS 5
00005
00014 class Camera64 : public Object {
00015 private:
00016     int data[NUM_CAMERAS] = {};
00017     int id1;
00018     int id2;
00019     int id3;
00020     int id4;
00021 public:
00030     Camera64(const char* pName = "Camera64") :
00031         Object(pName),
00032         id1(1),
00033         id2(2),
00034         id3(3),
00035         id4(4)
00036     {
00037     }
```

```
00038
00046     virtual ~Camera64() {
00047     }
00048
00057     void run() {
00058         for (int i = 0; i < NUM_CAMERAS; i++) {
00059             this->data[i] = i;
00060         }
00061         int sum = 0;
00062         for (int i = 0; i < NUM_CAMERAS; i++) {
00063             sum = sum + this->data[i];
00064         }
00065         printf("%s::%s(%d)::run()=%d\n", this->getNameClass(), this->getNameObject(), this->getId(),
      sum);
00066     }
00067 };
00068
```

5.3 Exception.h

```
00001 #pragma once
00002
00003 #include "Object.h"
00004 #include <cstring>
00005 #include <string>
00006
00015 class Exception : public Object {
00016 public:
00025     enum class ECode {
00026         eOutOfMemory,
00027         eLengthECode
00028     };
00029
00030 private:
00031     ECode eCode;
00032     char* pNameObject;
00033     char* pNameClass;
00034
00035 public:
00047     Exception(ECode eCode, const char* pNameObject, const char* pNameClass = "Exception")
00048         : eCode(eCode), pNameObject(_strdup(pNameObject)), pNameClass(_strdup(pNameClass)) {}
00049
00058     virtual ~Exception() {
00059         delete[] pNameObject;
00060         delete[] pNameClass;
00061     }
00062
00071     ECode getECode() const { return eCode; }
00072
00081     const char* getNameObject() const { return pNameObject; }
00090     const char* getNameClass() const { return pNameClass; }
00091
00101     const char* getECodeAsString() const {
00102         switch (eCode) {
00103         case ECode::eOutOfMemory: return "Out Of Memory";
00104         default: return "Unknown Error";
00105         }
00106     }
00107
00115     void println() {
00116         printf("Exception.h : %s(%s, %s)\n", getECodeAsString(), pNameObject, pNameClass);
00117     }
00118 };
```

5.4 ExceptionManager.h

```
00001 #pragma once
00002
00003 #include "Exception.h"
00004 #include <vector>
00005 #include <iostream>
00006
00015 class ExceptionManager {
00016 private:
00017     static std::vector<std::string> exceptionLogs;
00018
00019 public:
00029     static void logException(const Exception& ex) {
00030         std::string log = "ExceptionManager-Exception: " + std::string(ex.getECodeAsString()) +
```

```
00031             "(" + std::string(ex.getNameObject()) + ", " + std::string(ex.getNameClass()) + ")";
00032         exceptionLogs.push_back(log);
00033         std::cout « log « std::endl;
00034     }
00043     static void printAllExceptions() {
00044         std::cout « "ExceptionManager-Logged Exceptions:" « std::endl;
00045         for (const auto& log : exceptionLogs) {
00046             std::cout « log « std::endl;
00047         }
00048     }
00049 };
00050
```

5.5 IMemoryManager.h

```
00001 #pragma once
00002
00003 #include "ISlotIndex.h"
00004 #include "PageIndex.h"
00005
00014 class IMemoryManager
00015 {
00016 public:
00027     virtual void* allocate(size_t sizeSlot, const char* pName) = 0;
00028
00038     virtual void dellocate(void* pObject) = 0;
00039
00049     virtual ISlotIndex* findASlotIndex(void* pObject) = 0;
00050
00060     virtual PageIndex findAPageIndex(void* pObject) = 0;
00061
00070     virtual void showStatus() = 0;
00071 };
00072
```

5.6 IObject.h

```
00001 #pragma once
00002
00011 class IObject
00012 {
00013 public:
00021     virtual int getId() = 0;
00029     virtual char* getNameClass() = 0;
00037     virtual char* getNameObject() = 0;
00038 };
00039
```

5.7 ISlotIndex.h

```
00001 #pragma once
00002
00010 class ISlotIndex
00011 {
00012 public:
00021     virtual char* getNameObject() = 0;
00022 };
00023
```

5.8 Main.h

```
00001 #pragma once
00002 #include "Object.h"
00003
00004 #include "Camera.h"
00005 #include "Recorder.h"
00006
00014 class Main : public Object
00015 {
00016 public:
00017     Main(const char* pName = "Main") : Object(pName)  {
```

```
00018     }
00019     void initialize() {
00020     }
00021     void finalize() {
00022     }
00023     ~Main() {
00024     }
00025
00026     void run() {
00027     }
00028 };
```

5.9 MemoryManager.h

```
00001 #pragma once
00002
00003 #include "IMemoryManager.h"
00004 #include "PageIndex.h"
00005 #include "Exception.h"
00006 #include "ExceptionManager.h"
00007
00016 class MemoryManager : public IMemoryManager {
00017 private:
00018     char* pBuffer;
00019     size_t sizeBuffer;
00020
00021     int numPages;
00022     size_t sizePage;
00023     PageIndex* aPageIndex;
00024
00025     size_t normalizeSize(size_t size) {
00026         size_t size1 = size » 4;
00027         size_t size2 = size1 « 4;
00028         size = ((size == size2) ? size1 : ++size1) « 4;
00029         return size;
00030     }
00031     PageIndex* allocateNewPages(size_t sizeSlot) {
00032         int numPagesRequired = (sizeSlot % sizePage) == 0 ? (int)(sizeSlot / sizePage) :
      (int)(sizeSlot / sizePage) + 1;
00033
00034         PageIndex* pPageIndexAllocated = nullptr;
00035         bool bFound = false;
00036         int numConsequtivePages = 0;
00037
00038         for (int i = numPages - 1; i >= 0; i--) {
00039             if (bFound) {
00040                 if (this->aPageIndex[i].isAllocated()) {
00041                     break;
00042                 }
00043                 else {
00044                     numConsequtivePages++;
00045                     this->aPageIndex[i].setNumConsecutivePages(numConsequtivePages);
00046                 }
00047             }
00048             else {
00049                 if (!this->aPageIndex[i].isAllocated()) {
00050                     if (this->aPageIndex[i].getNumConsecutivePages() == numPagesRequired) {
00051                         pPageIndexAllocated = &(this->aPageIndex[i]);
00052                         for (int k = i; k < i + numPagesRequired; k++) {
00053                             this->aPageIndex[k].initialize(sizeSlot);
00054                             this->aPageIndex[k].setBAllocated(true);
00055                         }
00056                         bFound = true;
00057                     }
00058                 }
00059             }
00060         }
00061         return pPageIndexAllocated;
00062     }
00063
00064     void collectGarbage() {
00065
00066     }
00067
00068 public:
00078     void* operator new(size_t size) {
00079         return malloc(size);
00080     }
00081
00090     void operator delete(void* pObject) {
00091         free(pObject);
00092     }
00093
```

```
00105     MemoryManager(char* pBuffer, size_t sizeBuffer, size_t sizePage) :
00106         pBuffer(pBuffer),
00107         sizeBuffer(sizeBuffer),
00108         sizePage(sizePage)
00109     {
00110         numPages = (int)(sizeBuffer / sizePage);
00111         aPageIndex = new PageIndex[numPages];
00112 
00113         char* pBufferCurrent = pBuffer;
00114         for (int i = 0; i < numPages; i++) {
00115             aPageIndex[i].setPPage(pBufferCurrent);
00116             aPageIndex[i].setSizePage(sizePage);
00117             aPageIndex[i].setNumConsecutivePages(numPages - i);
00118             pBufferCurrent = pBufferCurrent + sizePage;
00119         }
00120     }
00121 
00129     virtual ~MemoryManager() {
00130     }
00131 
00143     void* allocate(size_t sizeMemory, const char* pName) {
00144         size_t sizeSlot = normalizeSize(sizeMemory);
00145 
00146         PageIndex* pPageIndexFound = nullptr;
00147         for (int i = 0; i < numPages; i++) {
00148             if (this->aPageIndex[i].isAllocated()) {
00149                 if (this->aPageIndex[i].getSizeSlot() == sizeSlot) {
00150                     pPageIndexFound = &(this->aPageIndex[i]);
00151                     break;
00152                 }
00153             }
00154         }
00155 
00156         if (pPageIndexFound == nullptr) {
00157             pPageIndexFound = allocateNewPages(sizeSlot);
00158             if (pPageIndexFound == nullptr) {
00159                 throw Exception(Exception::ECode::eOutOfMemory, "allocateNewPages1");
00160             }
00161         }
00162 
00163         void* pSlotAllocated = pPageIndexFound->allocate(sizeSlot, pName);
00164         if (pSlotAllocated == nullptr) {
00165             pPageIndexFound = allocateNewPages(sizeSlot);
00166             if (pPageIndexFound == nullptr) {
00167                 throw Exception(Exception::ECode::eOutOfMemory, "allocateNewPages2");
00168             }
00169             else {
00170                 pSlotAllocated = pPageIndexFound->allocate(sizeSlot, pName);
00171             }
00172         }
00173         return pSlotAllocated;
00174     }
00175 
00185     void dellocate(void* pObject) {
00186         size_t offset = (size_t)pObject - (size_t)(this->pBuffer);
00187         int pageIndex = (int)(offset / sizePage);
00188         bool isEmpty = this->aPageIndex[pageIndex].dellocate(pObject);
00189         if (isEmpty) {
00190             // this->aPageIndex[pageIndex].finalize();
00191             for (int i = 0; i < this->aPageIndex[pageIndex].getNumConsecutivePages(); i++) {
00192                 this->aPageIndex[pageIndex + i].finalize();
00193             }
00194             for (int i = 0; i < this->aPageIndex[pageIndex].getNumConsecutivePages(); i++) {
00195                 this->aPageIndex[pageIndex + i].finalize();
00196             }
00197         }
00198         int startConsecutivePages = this->numPages - 1;
00199         for (int i = pageIndex; i < numPages; i++) {
00200             if (this->aPageIndex[i].isAllocated()) {
00201                 startConsecutivePages = i - 1;
00202                 break;
00203             }
00204         }
00205         int numConsecutivePages = 1;
00206         for (int i = startConsecutivePages; i >= 0 || this->aPageIndex[pageIndex].isAllocated(); i--)
    {
00207             this->aPageIndex[i].setNumConsecutivePages(numConsecutivePages);
00208             numConsecutivePages++;
00209         }
00210         // ===============
00211         this->collectGarbage();
00212         // ===============
00213     }
00214 
00225     SlotIndex* findASlotIndex(void* pObject) {
00226         size_t offset = (size_t)pObject - (size_t)(this->pBuffer);
00227         int pageIndex = (int)(offset / sizePage);
```

```
00228         SlotIndex* pSlotIndexFound = this->aPageIndex[pageIndex].findASlotIndex(pObject);
00229         return pSlotIndexFound;
00230     }
00231
00242     PageIndex findAPageIndex(void* pObject) {
00243         size_t offset = (size_t)pObject - (size_t)(this->pBuffer);
00244         int pageIndex = (int)(offset / sizePage);
00245         return this->aPageIndex[pageIndex];
00246     }
00247
00256     void showStatus() {
00257         printf("Start=========================================\n");
00258         for (int i = 0; i < numPages; i++) {
00259             for (int j = 0; j < aPageIndex[i].getNumConsecutivePages(); j++) {
00260                 printf("PageIndex%d(SizeSlot=%d, ConsecutivePages=%d)\n", i + j, (int)aPageIndex[i +
      j].getSizeSlot(), aPageIndex[i + j].getNumConsecutivePages());
00261             }
00262             this->aPageIndex[i].showStatus();
00263             i = i + aPageIndex[i].getNumConsecutivePages() - 1;
00264         }
00265         printf("End===========================================\n");
00266     }
00267 };
```

5.10 Object.h

```
00001 #pragma once
00002
00003 #include "Type.h"
00004 #include "IObject.h"
00005 #include "IMemoryManager.h"
00006 #include "ISlotIndex.h"
00007
00016 class Object: public IObject
00017 {
00018 private:
00019     int id;
00020     char* pNameClass;
00021
00022 public:
00023     static int s_counterId;
00024     static IMemoryManager* s_pMemoryManager;
00025
00036     void* operator new (size_t size, const char* pName) {
00037         void* pAllocated = s_pMemoryManager->allocate(size, pName);
00038         return pAllocated;
00039     }
00040
00049     void operator delete(void* pAllocated) {
00050         s_pMemoryManager->dellocate(pAllocated);
00051     }
00052
00062     void operator delete(void* pObject, const char* pName) {
00063         s_pMemoryManager->dellocate(pObject);
00064     }
00065
00074     Object(const char* pClassName = "Object") :
00075         id(s_counterId++)
00076     {
00077         this->pNameClass = strcpy(pClassName);
00078     }
00079
00086     virtual ~Object() {
00087     }
00088
00096     int getId() { return this->id; }
00097
00105     char* getNameClass() { return this->pNameClass; }
00106
00114     char* getNameObject() {
00115         ISlotIndex* pSlotIndex = s_pMemoryManager->findASlotIndex(this);
00116         return pSlotIndex->getNameObject();
00117     }
00118
00126     size_t getSizeSlot() {
00127         PageIndex pSlotIndex = s_pMemoryManager->findAPageIndex(this);
00128         return pSlotIndex.getSizeSlot();
00129     }
00130 };
```

5.11 PageIndex.h

```
00001 #pragma once
00002
00003 #include "SlotIndex.h"
00004 #include "IObject.h"
00005
00014 class PageIndex {
00015
00016 private:
00017     static int s_countId;
00018
00019     int id;
00020     char* pPage;
00021     size_t sizePage;
00022     int numConsecutivePages;
00023     bool bAllocated;
00024
00025     size_t sizeSlot;
00026     int numMaxSlots;
00027     SlotIndex* aSlotIndex;
00028     int numSlotsAvailable;
00029
00030 public:
00040     void* operator new(size_t size) {
00041         return malloc(size);
00042     }
00043
00052     void operator delete(void* pObject) {
00053         free(pObject);
00054     }
00055
00064     int getId() { return this->id; }
00065
00074     void setPPage(char* pPage) { this->pPage = pPage; }
00075
00084     void setSizePage(size_t sizePage) { this->sizePage = sizePage; }
00085
00094     void setNumConsecutivePages(int numConsecutivePages) { this->numConsecutivePages =
      numConsecutivePages; }
00095
00104     int getNumConsecutivePages() { return this->numConsecutivePages; }
00105
00114     bool isAllocated() { return this->bAllocated; }
00115
00124     void setBAllocated(bool bAllocated) { this->bAllocated = bAllocated; }
00125
00134     size_t getSizeSlot() { return this->sizeSlot; }
00135
00144     PageIndex() :
00145         id(s_countId++),
00146         pPage(nullptr),
00147         sizePage(NOT_DEFINED),
00148         numConsecutivePages(NOT_DEFINED),
00149         bAllocated(false),
00150
00151         sizeSlot(NOT_DEFINED),
00152         numMaxSlots(NOT_DEFINED),
00153         aSlotIndex(nullptr),
00154         numSlotsAvailable(NOT_DEFINED)
00155     {
00156     }
00157
00165     ~PageIndex() {}
00166
00176     void initialize(size_t sizeSlot) {
00177         this->sizeSlot = sizeSlot;
00178         this->numMaxSlots = (int)(sizePage < sizeSlot ? 1 : sizePage / sizeSlot);
00179         this->numSlotsAvailable = numMaxSlots;
00180         this->aSlotIndex = new SlotIndex[numMaxSlots];
00181         char* pCurrent = this->pPage;
00182         for (int i = 0; i < numMaxSlots; i++) {
00183             aSlotIndex[i].setPMemory(pCurrent);
00184             pCurrent = pCurrent + sizeSlot;
00185         }
00186         this->setBAllocated(true);
00187     }
00188
00196     void finalize() {
00197         this->setBAllocated(false);
00198     }
00199
00211     void* allocate(size_t size, const char* pNameObject) {
00212         void* pMemoryAllocated = nullptr;
00213         for (int i = 0; i < this->numMaxSlots; i++) {
00214             if (!this->aSlotIndex[i].isAllocated()) {
00215                 this->aSlotIndex[i].setNameObject(pNameObject);
```

```
00216                 this->aSlotIndex[i].setBAllocated(true);
00217                 pMemoryAllocated = this->aSlotIndex[i].getPMemory();
00218                 this->numSlotsAvailable--;
00219                 break;
00220             }
00221         }
00222         return pMemoryAllocated;
00223     }
00224
00235     bool dellocate(void* pMemory) {
00236         size_t offset = (char*)pMemory - pPage;
00237         int indexSlot = (int)(offset / sizeSlot);
00238         this->aSlotIndex[indexSlot].setBAllocated(false);
00239         this->numSlotsAvailable++;
00240         bool isEmpty = false;
00241         if (this->numSlotsAvailable == this->numMaxSlots) {
00242             isEmpty = true;
00243         }
00244         return isEmpty;
00245     }
00246
00257     SlotIndex* findASlotIndex(void* pMemory) {
00258         size_t offset = (char*)pMemory - pPage;
00259         int indexSlot = (int)(offset / sizeSlot);
00260         return &(this->aSlotIndex[indexSlot]);
00261     }
00262
00271     void showStatus() {
00272         printf("------------------------------------------------\n");
00273         if (!bAllocated) {
00274             printf("|  EMPTY SLOT\n");
00275         }
00276         else {
00277             for (int i = 0; i < numMaxSlots; i++) {
00278                 if (this->aSlotIndex[i].isAllocated()) {
00279                     IObject* pObject = (IObject*)(this->aSlotIndex[i].getPMemory());
00280                     printf("|  %s::%s(%d)\n", pObject->getNameClass(), aSlotIndex[i].getNameObject(),
      pObject->getId());
00281                 }
00282                 else {
00283                     printf("|  EMPTY SLOT\n");
00284                 }
00285             }
00286         }
00287         printf("------------------------------------------------\n\n");
00288     }
00289 };
```

5.12 Recorder.h

```
00001 #pragma once
00002
00003 #include "Object.h"
00004
00005 #define NUM_RECORDERS 20
00006
00015 class Recorder : public Object {
00016 private:
00017     int data[NUM_RECORDERS] = {};
00018 public:
00027     Recorder(const char* pName = "Recorder") :
00028         Object(pName)
00029     {
00030     }
00031
00039     virtual ~Recorder() {
00040     }
00041
00050     void run() {
00051         for (int i = 0; i < NUM_RECORDERS; i++) {
00052             this->data[i] = i;
00053         }
00054         int sum = 0;
00055         for (int i = 0; i < NUM_RECORDERS; i++) {
00056             sum = sum + this->data[i];
00057         }
00058         printf("%s::%s(%d)::run()=%d\n", this->getNameClass(), this->getNameObject(), this->getId(),
      sum);
00059     }
00060 };
```

5.13 SlotIndex.h

```
00001 #pragma once
00002
00003 #include "ISlotIndex.h"
00004 #include "Type.h"
00005
00014 class SlotIndex : public ISlotIndex {
00015 private:
00016     void* pMemory;
00017     char* pNameObject;
00018     bool bAllocated;
00019 public:
00028     SlotIndex() :
00029         pMemory(nullptr),
00030         pNameObject(nullptr),
00031         bAllocated(false)
00032     {
00033     }
00034
00042     ~SlotIndex() {}
00043
00052     void setPMemory(void* pMemory) { this->pMemory = pMemory; }
00053
00062     void* getPMemory() { return this->pMemory; }
00063
00072     void setNameObject(const char* pNameObject) {
00073         this->pNameObject = strcpy(pNameObject);
00074     }
00075
00084     char* getNameObject() { return this->pNameObject; }
00085
00094     void setBAllocated(bool bAllocated) { this->bAllocated = bAllocated; }
00095
00104     bool isAllocated() { return this->bAllocated; }
00105 };
```

5.14 string.h

```
00001 #pragma once
00002 namespace o2system
00003 {
00004     #define NOT_DEFINED -1
00005     #define SIZE_NAME 20
00006     #define EOS '\0'
00007
00008     char* to_pchar(int number);
00009     size_t strlen(const char* pString);
00010     char* strcpy(const char* pNameSource);
00011 };
00012
```

5.15 Type.h

```
00001 #pragma once
00002
00003 #include "string.h"
00004 using namespace o2system;
00005
00006 #include <stdio.h>
00007 #include <stdlib.h>
```

Index