Packet System

상속과 디자인 패턴을 활용한 Packet System

목차

- 개요
- Design
  - 구현 목표
  - Key idea
- Packet Protocol
- 구현
  1. 시스템 시나리오
  2. Class Diagram
  3. Sequence Diagram
  4. 파일 구조
- 코드 샘플
- 블로그 & Github 주소

개요 1. 구현 목표

- 패킷의 크기가 고정되지 않아 패킷의 크기 변화가 자유로움
- 새로운 패킷 종류를 추가하더라도 기존 코드의 변경을 최소화

Design 2. Key idea : Inheritance

- Description

  - 모든 패킷의 Base class가 되는 Packet_Base class를 구현하고, 각각의 패킷은 이를 상속하는 개별 클래스로 구현.

- Benefits

  - 개별적인 패킷의 공통 부분을 묶어 간소화하고 패킷 클래스 간의 구조를 체계화
  - Packet_Base를 상속하는 패킷 클래스를 정의하여 새로운 패킷을 간단히 추가 가능

Design 2. Key idea : Strategic pattern

- Description
  - 각 패킷 별로 서로 다른 패킷 처리 과정(de/serialization, packet handle process)을 가지므로, 이러한 처리 과정을 패킷 별 고유한 행위라 가정, 패킷의 처리과정을 해당 패킷 클래스의 오버라이딩 멤버 메소드로 구현.
  - 패킷을 받으면 패킷의 종류에 따라 패킷 클래스 별로 구현된 패킷 프로세스 알고리즘이 작동
- Benefits
  - 패킷 처리 구문이 서버 측의 함수로 구현되지 않기 때문에 패킷 종류가 많아도 서버 측의 패킷 처리 함수가 복잡해질 우려가 없음.
  - 새로운 패킷이 추가가 되더라도 기존 코드를 수정할 필요가 없기 때문에 패킷 시스템의 확장이 용이함.

Design 2. Key idea : Template method pattern

- Description
  - 패킷 클래스 별로 패킷 처리 과정에서 공통되는 부분(패킷 헤더 필드 작성)을 묶고, 차이가 있는 부분(패킷 클래스 별로 다른 멤버 필드 de/serialize 과정)을 오버라이딩 멤버 메소드로 구현.

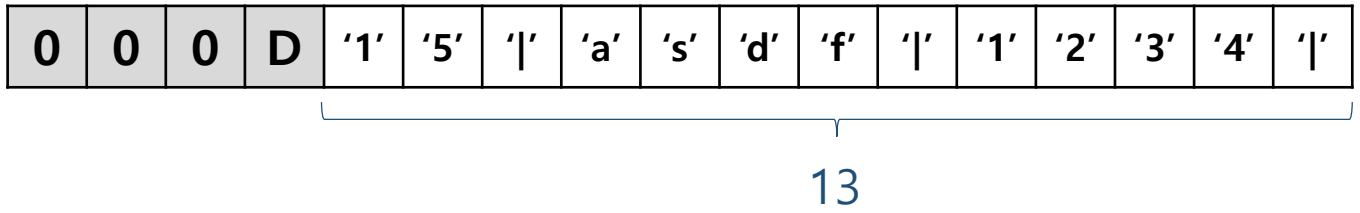
- Benefits
  - 공통적인 부분을 묶어 구현 시 작업량을 줄이고 구현 과정에서의 실수 확률이 줄어듦
  - 코드 수정 시 공통되는 부분을 한번에 수정할 수 있어 유지보수성 증가

Packet Protocol 1. 구조

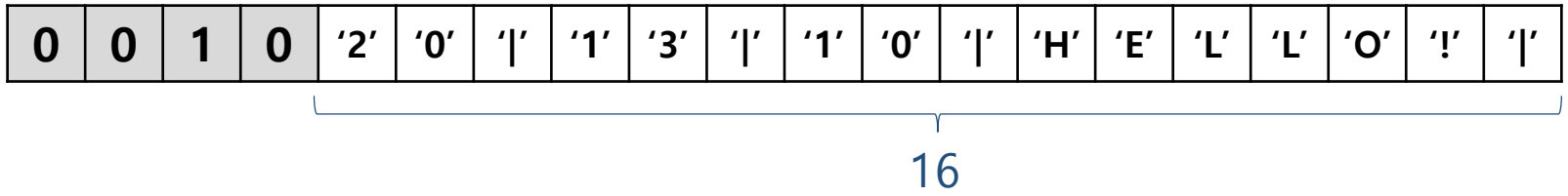
- 구조
  - 패킷은 header 부분과 buffer 부분으로 나뉘며, header에는 이어서 받아야할 buffer의 길이가 들어감.
  - Header의 길이는 sizeof(size_t)로 고정하고, Buffer의 길이는 임의의 길이가 가능.
  - 각 Packet class 별로 serialize() 메소드가 각각 정의되며, Buffer 부분은 이 메소드에 정의 된 규칙에 따라 serialize 됨. Deserialize 과정도 serialize 와 동일한 방식이 적용됨.
  - Packet class의 각 필드 값을 구분하기 위해 구분자로 '|' 문자를 사용. 구분자와 겹치지 않기 위해 입력 값 중에 같은 문자가 있는 경우 이를 특정한 다른 문자(space 또는 '_' 등)로 변경하여 저장.

Packet Protocol 2. 예시

- 프로토콜 예시
  - PK_CS_LOGIN_REQUEST : packet id(15), id("asdf"), password("1234")
    header space(13)



  - PK_CS_CHAT_CHAT: packet id(20), user key(13), room key(10), 입력 내용("HELLO!")
    header space(16)

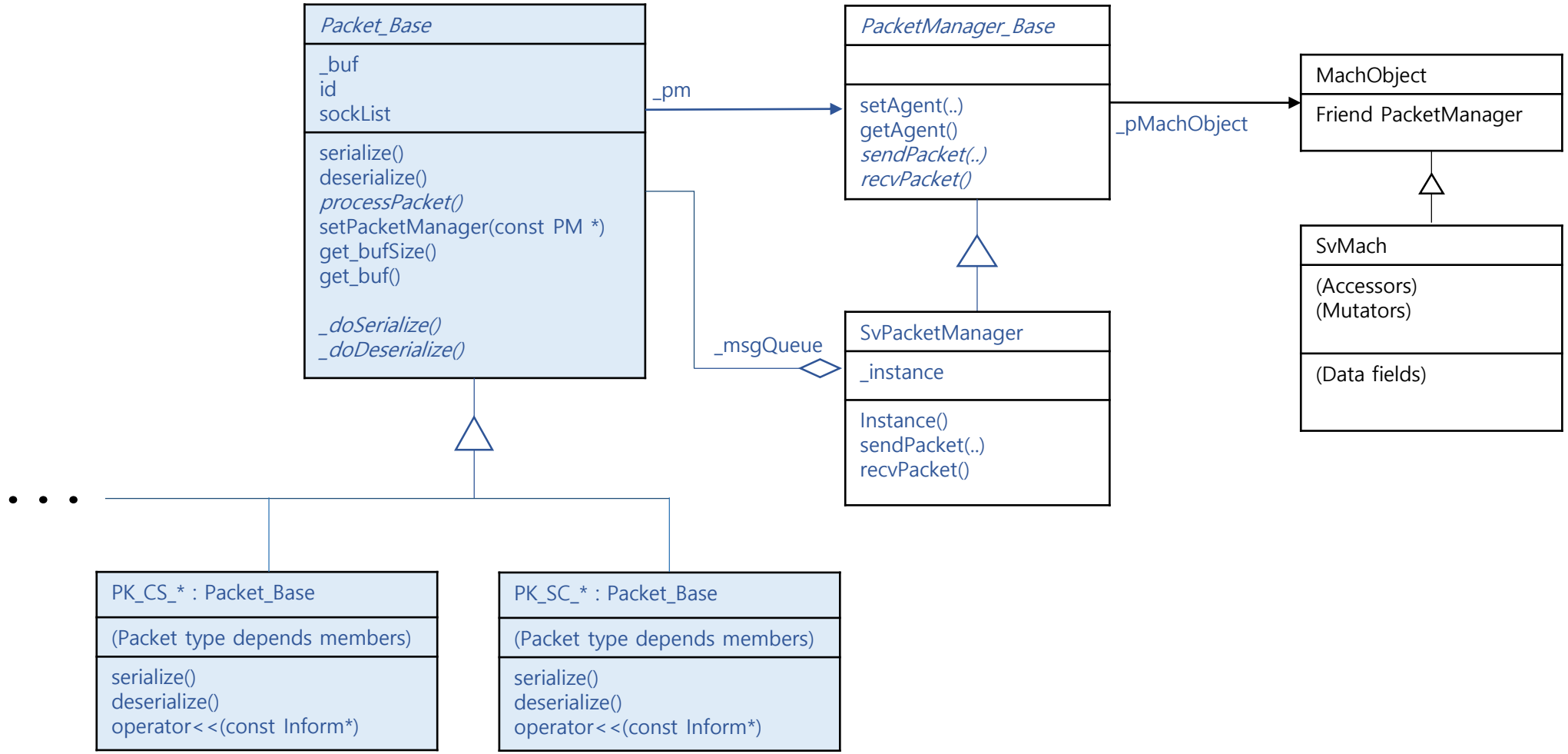


구현 1. 시스템 시나리오

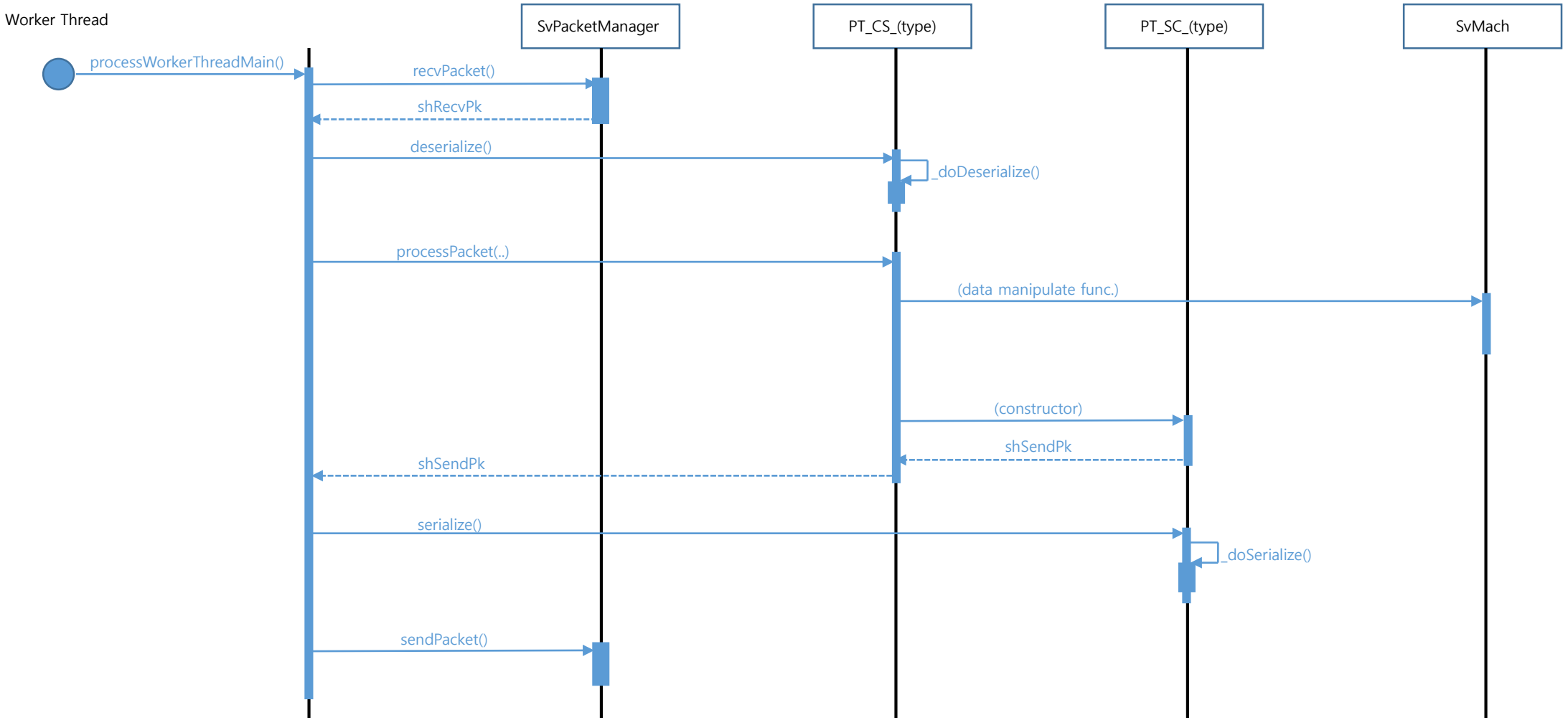
- 각각의 패킷을 모두 개별적인 패킷 클래스로 구현.
- 각 패킷 클래스는 해당 패킷이 가지고 있어야하는 정보를 멤버 필드로 가짐.
- 각 패킷 클래스는 자신(패킷)이 처리되는 방법을 정의한 오버라이딩 멤버 메소드(패킷 처리 함수)를 가짐.
- 각 패킷 클래스는 자신(패킷)이 de/serialize 되는 방법을 정의한 오버라이딩 멤버 메소드를 가짐.

- 전송 패킷 생성
  1. 생성하려는 패킷에 따라 패킷 클래스의 객체 생성
  2. 패킷 객체의 멤버 필드를 작성
  3. 패킷 클래스의 serialize() call하면 작성된 멤버 필드가 패킷 객체의 버퍼에 serialization.
  4. 패킷 객체 전송. 이 때 serialize 된 패킷의 버퍼가 전송됨.
- 패킷 수신 시 처리
  1. 수신한 패킷에서 패킷 id를 읽어 해당하는 패킷 클래스의 객체 생성
  2. 수신한 패킷의 내용을 패킷 객체의 버퍼에 복사. (여기까지 완료된 패킷은 수신 패킷 큐로 보내져 패킷 처리 차례를 기다림.)
  3. 패킷 처리 시엔 버퍼를 deserialize 해 패킷의 멤버 필드 복원. 그 후 패킷 클래스 내부에 정의된 패킷 처리 함수에 따라 패킷을 처리

구현 2. Class Diagram : Packet system

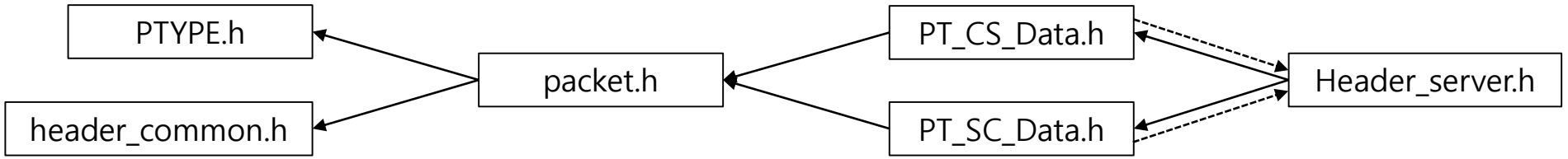


구현 3. Sequence Diagram : Packet system



구현 4. 파일 구조 : Packet system

- header_common.h : 서버와 클라이언트에 공통적으로 쓰이는 data class를 정의.
- Header_server.h : 서버에 쓰이는 data class를 정의
- PTYPE.h : packe에 따른 고유 번호를 정의.
- packet.h : 모든 packet의 base class 정의. Packet manage의 base class 정의.
- PT_CS_Data.h : 클라이언트에서 서버로 가는 packet 정의.
- PT_SC_Data.h : 서버에서 클라이언트로 가는 packet 정의.



Code Sample : Packet system 1 / 2

```
/*************************************************************************
 * Packet_Base class
 *************************************************************************/
class PacketManager_Base;
struct Packet_Base
{
public:
	static const size_t HEADER_SIZE = sizeof(size_t);
public:
	/* Member method */
	Packet_Base(PTYPE pType, const char * buf, size_t bufLen);
	virtual ~Packet_Base();

	static int ptoi(PTYPE);

	void serialize();		// Template method for serialize process
	void deserialize();		// Template method for serialize process
	virtual std::shared_ptr<Packet_Base> processPacket(MachObject& targetMObject) = 0;  //
Process received packet using strategy pattern
	void setProcessInfo(ProcInfo::ProcCode resCode, std::string&& msg = "");

	// Accessor
	size_t get_packetSize(); // The size of whole packet include header space.
	size_t get_bufSize();			// The size of buffer(serialized information).
	const char * get_bufAddr() const;
	const std::stringstream& get_buf() const;
	const ProcInfo& get_ProcInfo() const;

	Packet_Base& operator<<(const char * buf);
public:
	/* Member field */
	const PTYPE id;
	std::vector<SOCKET> sockList;
protected:
	/* Member method */
	Packet_Base(PTYPE);

	virtual void _doSerialProc() = 0;
	virtual void _doDeserialProc() = 0;
protected:
	/* Member field */
	std::stringstream _buf;
	ProcessInfo _ProcInfo;
private:
	/* Member method */
	Packet_Base();
	void _setHeaderSpace();  // Make (header) space for packet(_buf) size in _buf.
	void _writeHeader();	// At the end of serialize process, write _buf size on header
space calling at the end of serialization.
	size_t _bufSize();			// The size of serialized information excluding
header space
};
```

```
/*************************************************************************
 * PK_CS_LOGIN_REQUEST class
 *************************************************************************/
struct PK_CS_LOGIN_REQUEST : public Packet_Base
{
public:
	/* Member method */
	PK_CS_LOGIN_REQUEST();
	PK_CS_LOGIN_REQUEST(PTYPE ptype, const char* buf, size_t bufLen);

	virtual std::shared_ptr<Packet_Base> processPacket(MachObject& targetMObject);	//
Received packet process procedure using strategy pattern
public:
	/* Member field */
	std::string userId;
	std::string userPassword;
protected:
	/* Member method */
	virtual void _doSerialProc();
	virtual void _doDeserialProc();
protected:
	/* Member field */

};
```

Code Sample : Packet system 2 / 2

```
/*************************************************************************
 * PacketManager_Base class
*************************************************************************/
class PacketManager_Base
{
public:
	/* Member method */
	virtual ~PacketManager_Base();

	void setAgent(MachObject * pMObj);
	MachObject& getAgent();

	virtual void sendPacket(std::shared_ptr<Packet_Base>) = 0;		// add
packet to outgoing queue.
	virtual std::shared_ptr<Packet_Base> recvPacket() = 0;	// get packet from incoming
queue.
public:
	/* Member field */
protected:
	/* Member method */
	PacketManager_Base();

protected:
	/* Member field */
	MachObject * _pMachObject;
};
```

```
/*************************************************************************
 * SvPacketManager class
	- Singleton pattern.
*************************************************************************/
class SvPacketManager : public PacketManager_Base
{
public:
	/* Member method */
	static SvPacketManager& Instance();

	void sendPacket(std::shared_ptr<Packet_Base> spPk);// transmit packet via network.
	std::shared_ptr<Packet_Base> recvPacket();// get packet from incoming packet queue.
public:
	/* Member field */
protected:
	/* Member method */
protected:
	/* Member field */
	static SvPacketManager _instance;
	std::queue<std::shared_ptr<Packet_Base>> _mapQueue;		// incoming packet queue

	friend DWORD WINAPI recvThreadMain(LPVOID);
	friend DWORD WINAPI packetProcessWorkerThreadMain(LPVOID);
};
```

블로그 & Github 주소

- 블로그 주소 : https://devwoodo.blogspot.kr
- Project repos. 주소 : https://github.com/devwoodo/chatters
- Portfolio repos. 주소 : https://github.com/devwoodo/folio