# TCT-기술인증테스트 시스템&솔루션개발 실기형 문제지

[ 2021년 #차수 ]

| 사번 | | 성명 | |
|---|---|---|---|
| 유의 사항 | 1. 공정한 평가를 위해 동료를 도와주는 행위, 보여주는 행위를 금지하고 있습니다.<br><br>2. 부정행위 적발 시, 응시한 평가는 0점 처리됩니다.<br><br>3. 본 시험지는 응시장 외부로 유출할 수 없으며, 시험 종료 후 감독관에게 제출해야 합니다. | | |

# Messaging Service

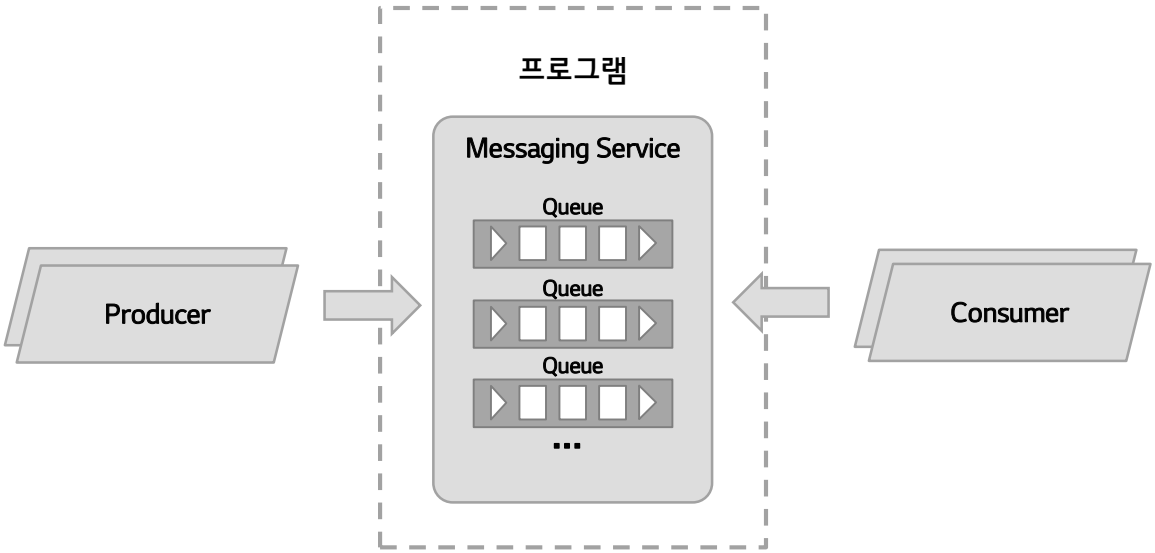
## 개요

해당 시스템 구현은 동해 요구사항 분석, 파일처리, HTTP Server/Client 구현, 비동기 처리 등의 기술역량 및 프로그램 구현 역량을 측정하기 위한 문제입니다.

## 설명

본 프로그램은 콘솔 입출력 또는 HTTP 요청/응답을 통해 Message를 송신/수신할 수 있는 Queue를 생성하고, Message를 Queue에 송신하거나 Queue에서 Message를 수신할 수 있는 서비스를 제공하는 'Messaging Service' 프로그램입니다.

### 구현 범위




[기능 요약]

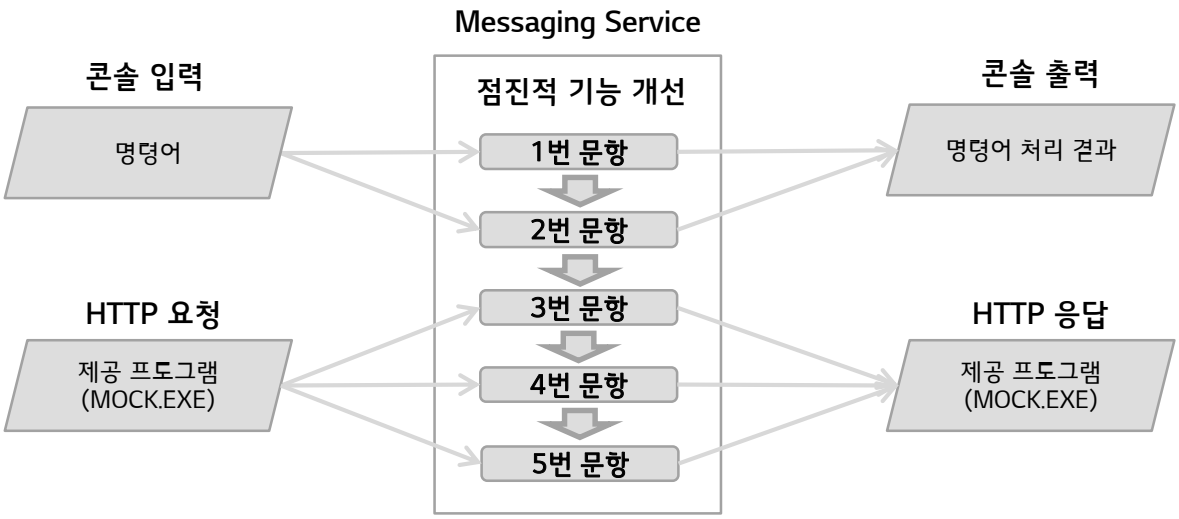
- 'Producer'는 콘솔 입력 또는 HTTP 요청/응답 방식으로 Messaging Service에 Queue를 생성하거나, Queue에 Message를 송신한다.
- 'Consumer'는 콘솔 입/출력 또는 HTTP 요청/응답 방식으로 Queue에 저장된 Message를 수신하여 핸들링 (비즈니스 로직 수행) 한다.
- 'Consumer'에서 Message 핸들링이 완료되면 Queue에서 Message를 삭제하고, 실패하면 Message를 수신 가능한 상태로 복원한다.
- 다양한 Queue 속성 설정에 따라 Queue의 동작을 제어하고, 백업 및 복구 기능을 제공한다.

## 주의사항

**실행 결과로 평가하고 부분점수는 없으므로 아래사항을 필히 주의해야 함**

- 구현된 프로그램은 실행 완결성 필수 (명확한 실행&종료 처리, 정확한 결과 출력, 동상의 실행 시간)
- 소 문항별 결과 검수 필수 (선행문항 오류 시, 후속문항 전체에 오류가 발생할 수 있음)
- 제시된 조건이 없는 한 선행요구사항 유지 필수 (소 문항별 입출력 관계도 참고)
- 프로그램 실행 위치 및 실행결과출력 (위치, 파일명, 데이터포맷)은 요구사항과 정확히 일치 필수
- 제시된 모든 위치는 상대경로 사용 필수 (프로그램 실행 위치 기준)
- 종료조건에 맞는 자동종료 처리 필수 (불필요한 종료방해처리(pause/입력대기 등)를 하면 안됨)

## 문항 관계



## 문제

아래 제시된 문항은 문항번호가 증가할수록 점진적 개선을 요구하는 방식으로 구성되어 있으며, 제시된 문항번호 별로 각각 **구현된 소스와 컴파일 된 실행파일을 제출**하시오.
cf) 1번 구현 → 1번 소스복사 → 2번 구현 → 2번 소스복사 → ...

1. 콘솔을 통해 명령어를 입력하면 Message를 Queue에 저장하거나 출력하는 Messaging Service를 구현하시오. (20점)

   - Message 송신 명령어 처리 ('※ Message 송신' 참조)
   - Message 수신 명령어 처리 ('※ Message 수신' 참조)

## 상세설명

※ Message 송신
- 콘솔 입력 명령어 형식 : SEND <Message>
- 동작 : <Message>를 Queue에 저장
- 콘솔 출력 : 없음 (빈칸/빈줄을 포함한 어떠한 출력도 하지 않음)

※ Message 수신
- 콘솔 입력 명령어 형식 : RECEIVE
- 동작 : Queue에 가장 먼저 저장된 Message 하나를 출력(수신)하고, 해당 Message를 삭제
- 콘솔 출력 : 출력할 Message가 존재할 경우 해당 Message 출력, 없는 경우 출력 없음

## 형식정보

※ 콘솔 입/출력
- 입력 포맷 : Message 송신 명령어 또는 Message 수신 명령어
- 출력 포맷 : 명령어 실행 결과

```
C:\>SP_TEST<엔터키>          ← 구현한 프로그램 실행 (Argument 없음)
SEND HELLO<엔터키>           ← 콘솔 입력
SEND WORLD<엔터키>           ← 콘솔 입력
RECEIVE<엔터키>              ← 콘솔 입력
HELLO                        ← 콘솔 출력
SEND LGCNS<엔터키>           ← 콘솔 입력
RECEIVE<엔터키>              ← 콘솔 입력
WORLD                        ← 콘솔 출력
RECEIVE<엔터키>              ← 콘솔 입력
LGCNS                        ← 콘솔 출력
```

## 평가대상

프로그램 정상 실행, 콘솔 출력 결과

## 문제

2. 위 1번 문항까지 구현된 내용을 기준으로, 복수의 Queue를 사용할 수 있도록 변경하여 Messaging Service를 구현하시오. (15점)
   - Queue 생성 명령어 추가 ('※ Queue 생성' 참조)
   - Message 송신, Message 수신 명령어 형식 및 동작 변경
     ('※ Message 송신', '※ Message 수신' 참조)

## 상세설명

※ Queue 생성
- 콘솔 입력 명령어 형식 : CREATE <Queue Name> <Queue Size>
- 동작 : <Queue Name>의 Queue를 최대 <Queue Size>개의 Message를 저장할 수 있도록 생성
- 콘솔 출력 : <Queue Name>의 Queue가 이미 존재하는 경우 "Queue Exist"을 출력, 정상인 경우 출력 없음

※ Message 송신
- 콘솔 입력 명령어 형식 : SEND <Queue Name> <Message>
- 동작 : <Message>를 <Queue Name>의 Queue에 저장
- 콘솔 출력 : <Queue Name>의 Queue에 <Queue Size>개의 Message가 이미 저장된 경우 "Queue Full"을 출력, 정상인 경우 출력 없음.

※ Message 수신
- 콘솔 입력 명령어 형식 : RECEIVE <Queue Name>
- 동작 : <Queue Name>의 Queue에 가장 먼저 저장된 Message 하나를 출력(수신)하고, Queue에서 해당 Message를 삭제
- 콘솔 출력 : 출력할 Message가 존재할 경우 해당 Message 출력, 없는 경우 출력 없음

## 형식정보

※ 콘솔 입/출력
- 입력 포맷 : Queue 생성 / Message 송신 및 Message 수신 명령어
- 출력 포맷 : 명령어 실행 결과

```
C:\>SP_TEST<엔터키>          ← 구현한 프로그램 실행 (Argument 없음)
CREATE PLAY 2<엔터키>        ← 콘솔 입력
CREATE LOG 5<엔터키>         ← 콘솔 입력
CREATE PLAY 2<엔터키>        ← 콘솔 입력
Queue Exist                  ← 콘솔 출력
SEND PLAY HELLO<엔터키>      ← 콘솔 입력
SEND PLAY WORLD<엔터키>      ← 콘솔 입력
SEND PLAY LGCNS<엔터키>      ← 콘솔 입력
Queue Full                   ← 콘솔 출력
SEND LOG START<엔터키>       ← 콘솔 입력
RECEIVE PLAY<엔터키>         ← 콘솔 입력
HELLO                        ← 콘솔 출력
RECEIVE PLAY<엔터키>         ← 콘솔 입력
WORLD                        ← 콘솔 출력
RECEIVE LOG<엔터키>          ← 콘솔 입력
START                        ← 콘솔 출력
```

## 평가대상

프로그램 정상 실행, 콘솔 출력 결과

## 문제

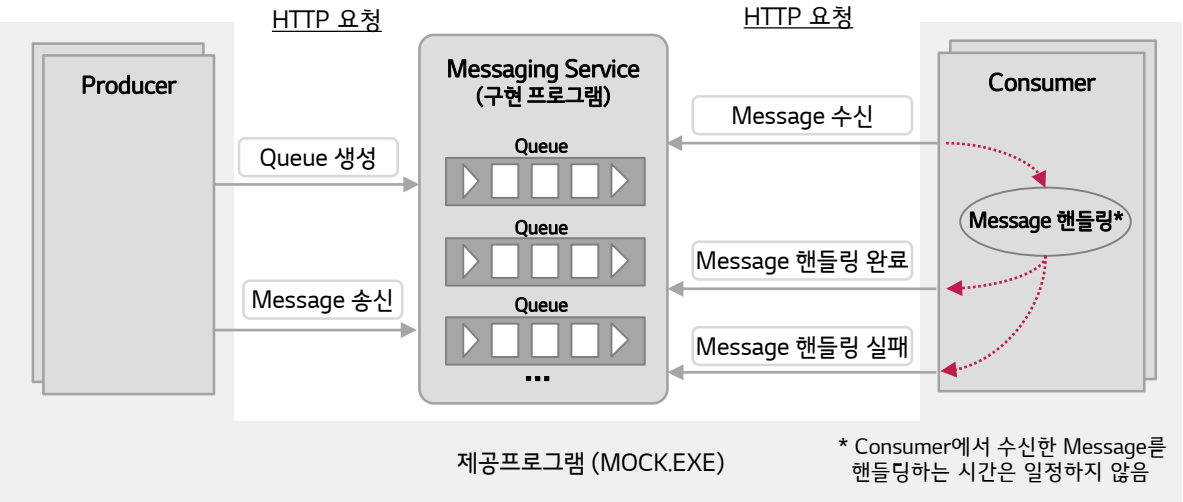
3. 위 2번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 HTTP 기반의 Messaging Service를 구현하시오. (15점)

- 콘솔 입/출력 방식에서 HTTP 방식으로 요청하고 응답 받도록 변경
  ('※ HTTP 기반 Messaging Service', '※ Queue 생성', '※ Message 송신', '※ Message 수신', '※ Message 핸들링 완료', '※ Message 핸들링 실패' 참조)
- Message 라이프사이클 변경
  ('※ Message 수신', '※ Message 핸들링 완료', '※ Message 핸들링 실패' 참조)

## 상세설명

※ HTTP 기반 Messaging Service

- HTTP 요청에 대해 Messaging Service는 **즉시 응답**해야 하며 대기하지 않아야 함
- 모든 HTTP 요청은 요청 받은 순서대로 처리
- 'Producer'와 'Consumer'의 역할은 제공프로그램이 수행 ('※ 제공프로그램 (MOCK.EXE)' 참조)




※ Queue 생성

- URI : POST http://127.0.0.1:8080/CREATE/<Queue Name>
- 요청 Body : JSON 문자열 형식, { "QueueSize" : <Queue Size>}
- 동작 : <Queue Name>의 Queue를 최대 <Queue Size>개의 Message를 저장할 수 있도록 생성
- 응답 Body : JSON 문자열 형식
  1) <Queue Name>의 Queue가 이미 존재하는 경우 {"Result": "Queue Exist"}
  2) 정상적으로 생성한 경우 {"Result": "Ok"}

※ Message 송신

- URI : POST http://127.0.0.1:8080/SEND/<Queue Name>
- 요청 Body : JSON 문자열 형식, { "Message" : "<Message>" }
- 동작 : <Message>를 <Queue Name>의 Queue에 저장
- 응답 Body : JSON 문자열 형식
  1) <Queue Name>의 Queue가 <Queue Size>개의 Message를 저장한 경우 {"Result": "Queue Full"}
  2) 정상적으로 Queue에 저장한 경우 {"Result": "Ok"}

## 상세설명 (계속)

※ Message 수신

- URI : GET http://127.0.0.1:8080/RECEIVE/<Queue Name>
- 동작 :
  1) <Queue Name>의 Queue에 저장된 수신 가능한 Message 중 가장 먼저 저장된 Message 하나를 응답
  2) 수신된 Message는 Queue에서 삭제되지 않지만, 다른 Consumer가 수신할 수 없음
  3) 각 Message를 식별할 수 있도록 고유의 <Message ID> 값을 응답에 포함
- 응답 Body : JSON 문자열 형식
  1) 응답할 Message가 존재할 경우, Message의 Message ID와 Message를 응답
     {"Result": "Ok", "MessageID": "<Message ID>", "Message": "<Message>"}
  2) 응답할 Message가 없는 경우 {"Result": "No Message"}

※ Message 핸들링 완료

- URI : POST http://127.0.0.1:8080/ACK/<Queue Name>/<Message ID>
- 설명 : Consumer에서 'Message 핸들링'이 정상적으로 완료되었음을 알림
- 동작 : <Message ID>의 Message 를 <Queue Name>의 Queue에서 삭제
- 응답 Body : {"Result": "Ok"}

※ Message 핸들링 실패

- URI : POST http://127.0.0.1:8080/FAIL/<Queue Name>/<Message ID>
- 설명 : Consumer에서 'Message 핸들링'에 실패하였음을 알림
- 동작 : <Message ID>의 Message를 **다른 Consumer가 수신할 수 있도록 복원**.
  단, 복원된 Message는 Queue에 저장된 **기존의 순서가 유지**되어야 함
  ex) [ M1, M2, M3 ]가 저장된 Queue에서 M1, M2가 수신된 후, M1이 핸들링 실패할 경우, 다음 Consumer는 M3가 아닌 M1을 먼저 수신하게 됨
- 응답 Body : {"Result": "Ok"}

※ 제공프로그램(MOCK.EXE)

- MOCK.EXE를 콘솔에서 실행하면 테스트 시나리오에 따라 Messaging Service의 기능을 순차적으로 테스트함
- MOCK.EXE는 자가 검수의 기능도 수행하며, 모든 테스트 시나리오 성공 시 다음의 문구가 콘솔에 출력

```
C:\>MOCK.EXE<엔터키>          ← 제공프로그램 실행
…                             ← 테스트 시나리오에 따른 테스트 실행
테스트에 성공했습니다!        ← 소문항의 모든 테스트시나리오 성공
```

## 평가대상

프로그램 정상 실행, HTTP 응답 결과 (JSON)

## 문제

4. 위 3번 문항까지 구현된 내용을 기준으로, 아래사항을 추가로 반영한 Message Service를 구현하시오.(20점)

- Queue 생성 변경 ('※ Queue 생성' 참조)
- Queue에 Process Timeout 속성 추가 ('※ Process Timeout 시간 초과' 참조)
- Dead Letter Queue 및 Dead Letter Queue 수신 추가
  ('※ Dead Letter Queue', '※ Dead Letter Queue 수신' 참조)
- Message 수신 대기 기능 추가 ('※ Message 수신 대기' 참조)

## 상세설명

※ Queue 생성

- URI : POST http://127.0.0.1:8080/CREATE/<Queue Name>
- 요청 Body : JSON 문자열 형식

```
{
  "QueueSize": <Queue Size>
  "ProcessTimeout": <Process Timeout>
  "MaxFailCount": <Max Fail Count>
  "WaitTime": <Wait Time>
}
```

- 동작 :
  1) 요청된 속성에 따라 Queue 생성
  2) <Queue Size> 속성에 따른 동작은 변경 없음
  3) <Process Timeout> 속성에 따른 동작은 '※ Process Timeout 시간 초과' 참조
  4) <Max Fail Count> 속성에 따른 동작은 '※ Dead Letter Queue' 참조
  5) <Wait Time> 속성에 따른 동작은 '※ Message 수신 대기' 참조
- 응답 Body : JSON 문자열 형식
  1) <Queue Name>의 Queue가 이미 존재하는 경우 {"Result": "Queue Exist"}
  2) 정상적으로 생성한 경우 {"Result": "Ok"}

※ Process Timeout 시간 초과

- <Process Timeout> 속성의 값이 0 보다 클 경우에 동작 (0일 경우에는 Timeout 없음)
- Consumer가 Queue에서 Message를 수신한 후 <Process Timeout>초 이내에 'Message 핸들링 완료' 또는 'Message 핸들링 실패' 요청이 없는 경우 Message를 다른 Consumer가 수신 가능하도록 복원 (즉, Message 핸들링 실패로 간주)
- 단, 복원된 Message는 Queue에 저장된 **기존의 순서가 유지**되어야 함 (Message 핸들링 실패와 동일 조건)

## 상세설명 (계속)

※ Dead Letter Queue

- Message 핸들링에 여러 번 실패한 Message(Dead Letter)를 보관하는 Queue
- 수신한 Message에 대해 'Message 핸들링 실패' 또는 'Process Timeout 시간 초과'가 발생한 횟수가 <Max Fail Count> 속성 값을 초과한 경우, 해당 Message를 복원하지 않고 Dead Letter Queue로 이동

※ Dead Letter Queue 수신

- URI : GET http://127.0.0.1:8080/DLQ/<Queue Name>
- 동작 : <Queue Name>의 Dead Letter Queue에 저장된 Message중 가장 먼저 저장된 Message 하나를 응답하고 해당 Message를 삭제
- 응답 Body :
  1) 응답할 Message가 존재할 경우
     {"Result": "Ok", "MessageID": "<Message ID>", "Message": "<Message>"}
  2) 응답할 Message가 없는 경우 {"Result": "No Message"}

※ Message 수신 대기

- <Wait Time> 속성의 값이 0 보다 클 경우에 'Message 수신' 요청에서 동작 (0일 경우에는 즉시 응답)
- Consumer의 'Message 수신' 요청 시 Queue에 Message가 없는 경우 최대 <Wait Time> 초 동안 수신 대기하며, **Message가 송신되거나 복원되는 즉시** 해당 Message를 대기중인 Consumer에게 응답
- <Wait Time> 초 동안 응답할 Message가 없는 경우, 대기중인 Consumer에게 {"Result": "No Message"} 를 응답
- 복수의 Consumer가 'Message 수신 대기' 하는 경우에는 **먼저 대기한 Consumer에게 우선 응답**

## 평가대상

프로그램 정상 실행, HTTP 응답 결과 (JSON)

## 문제

5. 위 4번 문항까지 구현된 내용을 기준으로, 아래사항을 추가로 반영한 Messaging Service를 구현하시오.(10점)

   - Shutdown 요청 시 백업 후 프로그램 종료 추가 ('※ Shutdown 요청' 참조)
   - Messaging Service 복구 기능 추가 ('※ Messaging Service 복구' 참조)

## 상세설명

※ Shutdown 요청

- URI : http://127.0.0.1:8080/SHUTDOWN
- 동작 :
  1) 'Message 수신 대기' 중인 모든 Consumer의 요청에 대해 즉시 {"Result": "Service Unavailable"} 을 응답
  2) Consumer가 수신한 모든 Message는 핸들링 완료된 것으로 간주하여 Queue에서 삭제
  3) 현재의 모든 Queue (Dead Letter Queue 포함)와 Message의 상태를 저장하고 **Messaging Service 종료**
- 응답 Body : JSON 문자열
  {"Result": "Ok"}

※ Messaging Service 복구

- 'Shutdown 요청' 으로 ('※ Shutdown 요청' 참조) 종료 시 저장된 Queue와 Message의 상태를 'Messaging Service'가 다시 시작될 때 Shutdown 전 상태로 복구

## 평가대상

프로그램 정상 실행, SHUTDOWN 요청시 정상 종료, HTTP 응답 결과 (JSON)

**폴더 정보**

**※ 프로그램 및 파일 위치 정보 (실행위치 기반 상대경로 사용 필수)**

- 구현할 프로그램 위치 및 실행 위치 : 각 소문항 홈 (SUB1/SUB2/SUB3/SUB4/SUB5)
- 자가 검수용 파일명 : SAMPLE 폴더 내 SAMPLE.TXT, CMP_CONSOLE.TXT (SUB1/SUB2)
- 제공되는 MOCK 프로그램 위치 : 각 소문항 홈 아래 MOCK.EXE (SUB3/SUB4/SUB5)

*** 제공되는 파일들은 문항에 따라 다를 수 있음**

**실행 방식**

**※ 구현할 프로그램 형식**

- 프로그램 형태 : 콘솔(Console) 프로그램
- 프로그램 파일명 : SP_TEST
- 실행 방식(문항1~2) : 콘솔 실행→결과처리 (종료 없음)

```
C:\>SP_TEST<엔터키>                ← 구현한 프로그램 실행 (Argument 없음)
SEND HELLO<엔터키>                 ← 콘솔 입력
RECEIVE<엔터키>                    ← 콘솔 입력
HELLO                              ← 콘솔 출력
```

- 실행방식(문항3~5) : 콘솔 실행→실시간 HTTP 요청 수신 (종료 없음)

```
C:\>SP_TEST<엔터키>                ← 구현한 프로그램 실행 (Argument 없음)
...
```

## 제공 및 제출

✓ 각 언어별 제공파일 압축 해제 후 자동 생성된 폴더 사용 필수

✓ 제공되는 주요 내용

- 샘플 파일 (SAMPLE.TXT, CMP_CONSOLE.TXT)
- 제공 프로그램 실행파일 (MOCK.EXE)
- 제출시 사용할 문항별 폴더 구조

✓ 제출 파일 및 폴더 상세 내용 (각 언어별 실기 가이드 참고)

**<주의사항>**

**※** 제출 파일 관련 내용 (폴더위치, 파일명, 프로그램명 등) 이 틀린 경우 및 상대경로를 사용하지 않은 경우에는 평가 시 불이익이 발생할 수 있으므로 반드시 요구되는 내용과 일치시켜 제출해야 함.

## 테스트 방법

**<주의사항>**

**※** 개발도구(Eclipse, Visual Studio)에서 제공되는 Debug, Run 기능의 실행환경은 콘솔의 SP_TEST 명령어 실행환경과 차이가 있으므로, **반드시 콘솔에서 SP_TEST 명령으로 테스트를 수행하여야 함**

**※ 자가 검수를 위해 제공되는 샘플은 검수용 데이터와 다를 수 있음**

검수를 위한 샘플 결과 파일은 각 문항 폴더(SAMPLE)에 사전 제공됨

[문항1]

- SP_TEST를 실행한 후 다음과 같이 입력하여 제공된 샘플 결과 파일(CMP_CONSOLE.TXT)과 동일한지 비교. 입력 시 SAMPLE.TXT의 데이터를 한줄 씩 복사/붙여넣기 하여 사용.

```
C:\>SP_TEST
SEND HELLO
SEND WORLD
RECEIVE
HELLO
SEND LGCNS
RECEIVE
WORLD
RECEIVE
LGCNS
```

## 테스트 방법 (계속)

[문항2]

- SP_TEST를 실행한 후 다음과 같이 입력하여 제공된 샘플 결과 파일(CMP_CONSOLE.TXT)과 동일한지 비교. 입력 시 SAMPLE.TXT의 데이터를 한줄씩 복사/붙여넣기 하여 사용.

```
C:\>SP_TEST
CREATE PLAY 2
CREATE LOG 5
CREATE PLAY 5
Queue Exist
SEND PLAY HELLO
SEND PLAY WORLD
SEND PLAY LGCNS
Queue Full
SEND LOG START
RECEIVE PLAY
HELLO
RECEIVE PLAY
WORLD
SEND PLAY LGCNS
RECEIVE LOG
START
```

[문항3]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 테스트 결과 출력( 테스트 실패 사유가 출력 되므로 자가 검수에 참고)

```
C:\>MOCK.EXE
3번 문항을 테스트합니다.
1. Queue 생성
……
2. Message 송신/수신
……
3. Message 핸들링 완료/실패
……
테스트에 성공했습니다!
```

- 테스트 중에 아래와 같은 형식으로 Mock의 요청 내용과, SP_TEST의 응답 내용이 출력

```
요청 : RECEIVE/PLAY       응답 : <응답 Body 내용>
요청 : RECEIVE/PLAY       응답 : <응답 Body 내용>
요청 : SEND/PLAY  <요청 Body 내용>      응답 : <응답 Body 내용>
요청 : SEND/PLAY  <요청 Body 내용>      응답 : <응답 Body 내용>
```

- 테스트 실패 시 아래와 같이 정답과 일치하지 않는 부분이 출력
  (아래 예시 : { "Result":"Ok" } 가 응답되어야 하지만 { "Result": "Queue Full" }이 응답되어 오답)

```
요청 : SEND/PLAY <요청 Body 내용>  응답 : {"Result": "Queue Full"}
테스트 실패! : 응답 Result 값이 일치하지 않습니다. Expected Result : Ok, but
Yours : Queue Full
```

**테스트 방법 (계속)**

[문항4]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 테스트 결과 출력( 테스트 실패 사유가 출력 되므로 자가 검수에 참고)

```
C:\>MOCK.EXE
4번 문항을 테스트합니다.
1. Queue 생성
……
2. Message 송신/수신
……
3. Message 핸들링 완료/실패
……
4. Process Timeout 결과 테스트
……
5. Dead Letter Queue 테스트
……
6. Message 수신 대기 테스트
……
테스트에 성공했습니다!
```

[문항5]

- SP_TEST를 실행한 후, MOCK.EXE를 실행
- MOCK.EXE의 콘솔에 테스트 결과 출력( 테스트 실패 사유가 출력 되므로 자가 검수에 참고)

```
C:\>MOCK.EXE
5번 문항을 테스트합니다.
1. Queue 생성
……
2. Message 송신/수신
……
3. 백업 복구 테스트(Message 수신 대기 'Service Unavailable' 확인)
SHUTDOWN          result=Ok
 - 프로그램(SP_TEST)을 다시 시작하고 엔터키를 입력하면 테스트 시작합니다.
```

- 백업 복구 테스트를 위해 SHUTDOWN이 요청되면 프로그램(SP_TEST)는 종료되어야 함
  수동으로 다시 프로그램(SP_TEST)를 실행하고 엔터키를 입력

```
3. 백업 복구 테스트(저장된 정보 복구 확인)
……
테스트에 성공했습니다!
```