Chapter 09

문제 9-1의 해답

○… (1) request 메소드를 호출하면 새로운 쓰레드가 한 개 기동한다.

○… (2) request 메소드 반환 값의 형은 Data 인터페이스지만 실제 반환 값은 FutureData의 인스턴스이다.

➜ request 메소드 안에서 FutureData 클래스의 인스턴스를 new하고 있습니다.

×… (3) setRealData 메소드를 호출하는 것은 메인 쓰레드인다.

➜ setRealData 메소드를 호출하는 것은 request 메소드 안에서 기동한 쓰레드입니다.

○… (4) RealData 클래스의 getContent 메소드를 실행하는 것은 메인 쓰레드이다.

×… (5) request 메소드가 복수의 쓰레드로부터 호출된다면 request 메소드는 synchronized로 해야 한다.

➜ request 메소드는 복수의 쓰레드가 공유하는 필드에 대해 아무런 처리도 하지 않기 때문에 synchronized 메소드로 할 필요는 없습니다. request의 인수(sount, c)나 지역 변수(future)도 이 request를 호출한 쓰레드에서만 액세스 됩니다.

문제 9-2의 해답

메인 쓰레드 이외에 3개 쓰레드가 만들어집니다. 3개 쓰레드는 각각 RealData의 인스턴스를 작성하는 업무를 하고 있습니다.

:: new RealData(10, 'A')를 실행하고 있는 쓰레드 - A 쓰레드

:: new RealData(20, 'B')를 실행하고 있는 쓰레드 - B 쓰레드

:: new RealData(30, 'C')를 실행하고 있는 쓰레드 - C 쓰레드

가령 이와 같이 이름을 붙이면 예제 프로그램의 실행 예(그림 9-3)의 표시는 각각 〈그림 9-1〉의 쓰레드가 하게 됩니다.

그림 9-1 어떤 쓰레드가 예제 프로그램을 출력(그림 9-3)하고 있는가

쓰레드의 종류	표시내용
메인 쓰레드	main BEGIN
메인 쓰레드	request(10, A) BEGIN
메인 쓰레드	request(10, A) END
메인 쓰레드	request(20, B) BEGIN
메인 쓰레드	request(20, B) END
메인 쓰레드	request(30, C) BEGIN
A 쓰레드	making RealData(10, A) BEGIN
B 쓰레드	making RealData(20, B) BEGIN
메인 쓰레드	request (30, C)
메인 쓰레드	main otherJob
C 쓰레드	making RealData(30, C) BEGIN
A 쓰레드	making RealData(10, A) END
B 쓰레드	making RealData(20, B) END
메인 쓰레드	main otherJob END
메인 쓰레드	data1 = AAAAAAAAAA
메인 쓰레드	data2 = BBBBBBBBBBBBBBBBBBBB
C 쓰레드	making RealData (30, C) END
메인 쓰레드	data3 = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
메인 쓰레드	main END

문제 9-3의 해답

우선 AsyncContentImpl 클래스의 인스턴스를 반환하도록 Retriever 클래스를 수정합니다(리스트 9-1). 그런 다음 AsyncContentImpl 클래스를 〈리스트 9-2〉와 같이 작성합니다. AsyncContentImpl 클래스가 Future 역할, SyncContentImpl 클래스가 RealData 역할이 됩니다.

주의 … 웹 페이지를 가져오는데 걸리는 시간은 환경의 영향을 받으므로 여기에서의 계측 시간은 어디까지나 참고로 해 주세요.

리스트 9-1 AsyncContentImpl의 인스턴스를 반환하도록 수정한 Retriever 클래스 (Retriever.java)

```
package content;

public class Retriever {
    public static Content retrieve(final String urlstr) {
        final AsyncContentImpl future = new AsyncContentImpl();

        new Thread() {
            public void run() {
```

```
                future.setContent(new SyncContentImpl(urlstr));
            }
        }.start();

        return future;
    }
}
```

⇒ 예제파일 경로 : 부록CD/src/Future/A9-3a/content

리스트 9-2 AsyncContentImpl 클래스 (AsyncContentImpl.java)

```
package content;

class AsyncContentImpl implements Content {
    private SyncContentImpl synccontent;
    private boolean ready = false;
    public synchronized void setContent(SyncContentImpl synccontent) {
        this.synccontent = synccontent;
        this.ready = true;
        notifyAll();
    }
    public synchronized byte[] getBytes() {
        while (!ready) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        return synccontent.getBytes();
    }
}
```

⇒ 예제파일 경로 : 부록CD/src/Future/A9-3a/content

그림 9-2 문제 9-3의 클래스 다이어그램 (멀티 쓰레드 판)

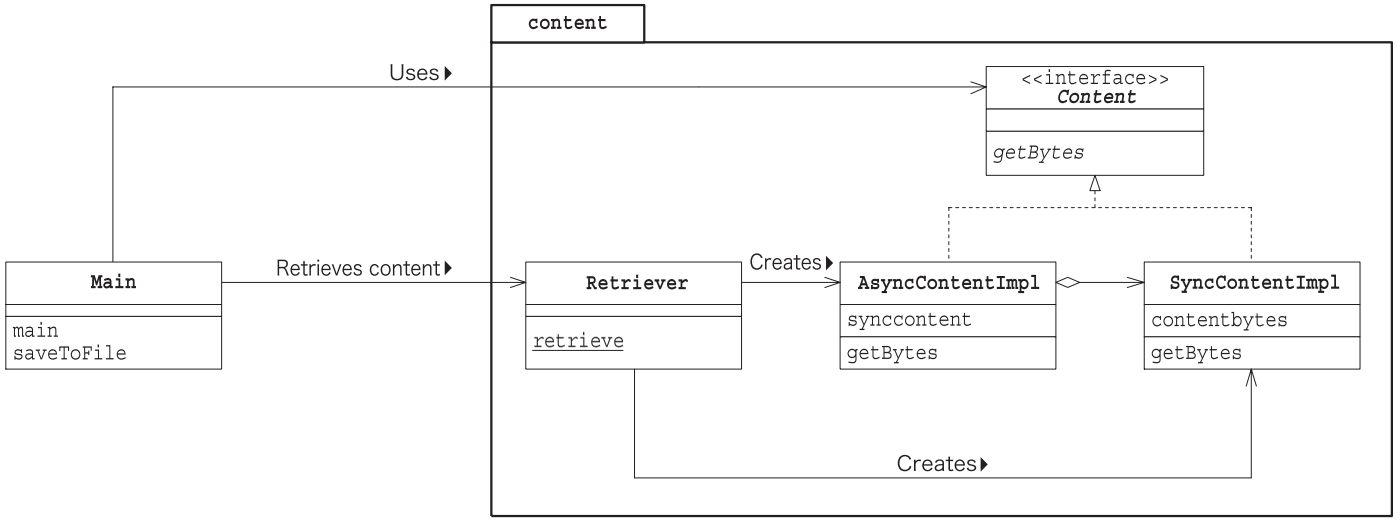

그림 9-3 실행 예

```
Thread-0: Getting http://www.yahoo.html
Thread-1: Getting http://www.google.html
Thread-2: Getting http://www.hyuki.html
main: Saving to yahoo.html
main: Saving to google.html
main: Saving to hyuki.html
Elapsed time = 1062msec.
```

FutureTask 클래스를 사용한 해답

〈리스트 9-3~리스트 9-4〉는 java.util.concurrent.FutureTask 클래스를 사용한 또 다른 해답입니다.

〈리스트 9-3〉은 java.util.concurrent.Callable를 사용하도록 수정한 Retriever 클래스입니다.

「SyncContetnImpl의 인스턴스를 생성한다」고 하는 처리가 시간이 걸리는 부분이므로 이 처리를 call 메소드로 실행하는 Callable 객체를 만듭니다. 그리고 그 Callable 객체를 AsyncContentImpl 생성자에게 건넵니다.

AsyncContentImpl의 인스턴스는 새로 생성한 쓰레드를 사용하여 실행합니다.

리스트 9-3 java.util.concurrent.Callable를 사용하도록 수정한 Retriever 클래스 (Retriever.java)

```
package content;

import java.util.concurrent.Callable;

public class Retriever {
    public static Content retrieve(final String urlstr) {
        AsyncContentImpl future = new AsyncContentImpl(
            new Callable<SyncContentImpl>() {
                public SyncContentImpl call() {
                    return new SyncContentImpl(urlstr);
                }
            }
        );

        new Thread(future).start();

        return future;
    }
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-3b/content*

〈리스트 9-4〉는 java.util.concurrent.FutureTask를 확장하여 만든 AsyncContentImpl 클래스입니다. 생성자에서는 인수로 주어진 callable을 슈퍼 클래스 FutureTask에게 전달합니다.

getBytes 메소드에서는 get 메소드로 구한 SyncContentImpl의 인스턴스에 대해 getBytes 메소드를 호출하여 목적하는 byte[]를 구합니다.

이 처리의 흐름은 보강에서 만든 프로그램과 거의 동일합니다.

리스트 9-4 java.util.concurrent.FutureTask를 확장하여 만든 AsyncContentImpl (AsyncContentImpl.java)

```
package content;

import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;
import java.util.concurrent.ExecutionException;

class AsyncContentImpl extends FutureTask<SyncContentImpl> implements Content {
    public AsyncContentImpl(Callable<SyncContentImpl> callable) {
        super(callable);
    }
    public byte[] getBytes() {
        byte[] bytes = null;
        try {
            bytes = get().getBytes();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
        return bytes;
    }
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-3b/content*

문제 9-4의 해답

java.util.concurrent.FutureTask를 사용하지 않는 해답(해답 1)과 FutureTask를 사용한 해답(해답 2)을 제시합니다.

◆ 해답 1 : java.util.concurrent.FutureTask를 사용하지 않는 해답

부록 G의 [Lea]에 소개되어 있는 방법(예외의 랩핑)을 사용합니다.

:: RealData의 인스턴스를 작성할 때 예외가 발생했다면, 그 예외를 FutureData 클래스에 설정하도록 Host 클래스를 수정합니다(리스트 9-5).

:: FutureData 클래스에 예외를 설정하는 메소드 setException을 추가합니다(리스트 9-7).

:: 실제로 발생한 예외를 「포함하기」위해서 예외 java.util.concurrent.ExecutionException을 사용합

니다(리스트 9-7).

:: getContent 메소드가 예외 ExecutionException을 통보합니다(리스트 9-6, 리스트 9-7).

리스트 9-5 해답 1 : 수정 후 Host 클래스 (Host.java)

```
public class Host {
    public Data request(final int count, final char c) {
        System.out.println("request(" + count + ", " + c + ") BEGIN");

        // (1) FutureData의 인스턴스를 만든다
        final FutureData future = new FutureData();

        // (2) RealData를 만들기 위한 새로운 쓰레드를 기동한다
        new Thread() {
            public void run() {
                try {
                    RealData realdata = new RealData(count, c);
                    future.setRealData(realdata);
                } catch (Exception e) {
                    future.setException(e);
                }
            }
        }.start();

        System.out.println("request(" + count + ", " + c + ") END");

        // (3) FutureData의 인스턴스를 반환 값으로 한다
        return future;
    }
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-4a*

리스트 9-6 해답 1 : 수정 후 Data 인터페이스 (Data.java)

```
import java.util.concurrent.ExecutionException;

public interface Data {
    public abstract String getContent() throws ExecutionException;
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-4a*

FutureData 클래스(리스트 9-7)의 setException 메소드는 setRealData 메소드와 거의 동일하지만 대입하는 곳이 realdata 필드가 아니라 exception 필드입니다.

exception에 대입하는 것은 ExecutionException의 인스턴스입니다.

리스트 9-7 해답 1 : 수정 후 FutureData 클래스 (FutureData.java)

```
import java.util.concurrent.ExecutionException;

public class FutureData implements Data {
    private RealData realdata = null;
    private ExecutionException exception = null;
    private boolean ready = false;
    public synchronized void setRealData(RealData realdata) {
        if (ready) {
            return;
        }
        this.realdata = realdata;
        this.ready = true;
        notifyAll();
    }
    public synchronized void setException(Throwable throwable) {
        if (ready) {
            return;
        }
        this.exception = new ExecutionException(throwable);
        this.ready = true;
        notifyAll();
    }
    public synchronized String getContent() throws ExecutionException {
        while (!ready) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        if (exception != null) {
            throw exception;
        }
        return realdata.getContent();
    }
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-4a*

그림 9-4 해답 1 : 실행 결과

```
main BEGIN
     request (-1, N) BEGIN
     request (-1, N) END
          making RealData (-1, N) BEGIN
java.util.concurrent.ExecutionException:
java.lang.NegativeArraySizeException
          at FutureData.setException(FutureData.java:19)
          at Host$1.run(Host.java:15)
Caused by: java.lang.NegativeArraySizeException ← 요인이 된 예외
         at RealData.<init>(RealData.java:5)
         at Host$1.run(Host.java:12)          ←  여기에서 프로그램은 종료한다
```

이것은 Future 역할로부터 필요한 값을 구하려 할 때까지 예외의 발생을 늦추는 것입니다. 저녁에 케이크 교환권을 가지고 빵집에 갔더니 "죄송해요, 오븐이 망가져서 오늘은 케이크를 못 만들어요"하는 것과 같은 이치입니다.

◆ 해답 2 : java.util.concurrent.FutureTask를 사용한 해답

java.util.concurrent.FutureTask를 사용한 해답을 제시합니다.

:: RealData 클래스, Main 클래스 … 문제문과 동일

:: Data 인터페이스 … 해답 1과 동일

:: Host 클래스 … 리스트 9-6과 동일

〈리스트 9-8〉의 FutureData 클래스는 〈리스트 9-7〉과 거의 같고 ExecutionException을 catch하지 않는 점만 다릅니다.

실행 결과는 〈그림 9-5〉와 같습니다.

리스트 9-8 해답 2 : java.util.concurrent.FutureTask를 확장하여 만든 FutureData 클래스 (FutureData.java)

```
import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;
import java.util.concurrent.ExecutionException;

public class FutureData extends FutureTask<RealData> implements Data {
    public FutureData(Callable<RealData> callable) {
        super(callable);
    }
    public String getContent() throws ExecutionException {
```

```
        String string = null;
        try {
            string = get().getContent();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return string;
    }
}
```

⇒ *예제파일 경로 : 부록CD/src/Future/A9-4*

그림 9-5 해답 2 : 실행 결과

```
main BEGIN
     request (-1, N) BEGIN
     request (-1, N) END
         making RealData (-1, N) BEGIN
java.util.concurrent.ExecutionException:
java.lang.NegativeArraySizeException
       at java.util.concurrent.FutureTask$Sync.innerGet(Unknown Source)
       at java.util.concurrent.FutureTask.get(Unknown Source)
       at FutureData.getContent(FutureData.java:12)
       at Main.main(Main.java:9)
Caused by: java.lang.NegativeArraySizeException
       at RealData.<init>(RealData.java:5)
       at Host$1.call(Host.java:12)
       at Host$1.call(Host.java:11)
       at java.util.concurrent.FutureTask$Sync.innerGet(Unknown Source)
       at java.util.concurrent.FutureTask.run(Unknown Source)
       at java.langThread.run(Unknown Source)
```

잠깐! 한 마디 : 예외 java.util.concurrent.ExecutionException

연습문제 9-4에서는 예외를 포함한 예외로서 java.util.concurrent.ExecutionException을 사용했습니다. ExecutionException이라고 하는 또 다른 예외로 포함함으로써 실제로 발생하는 예외의 형과는 관계없이 getContent의 throw절을 기술할 수 있습니다.

예외를 포함하는 방법에 대해서는 아래 페이지를 참고하세요.

:: Chained Exception Facility : http://java.sun.com/j2se/1.5.0/docs/guide/lang/chained-exceptions.html