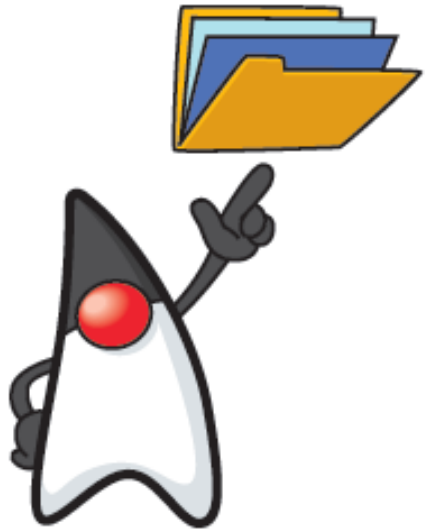
15장
파일 입출력

박숙영
blue@sookmyung.ac.kr

15장의 목표

1. 텍스트 파일을 오픈하여서 내용을 화면에 출력할 수 있나요?
2. 이미지 파일을 복사하는 프로그램을 작성할 수 있나요?
3. 객체를 파일에 저장할 수 있나요?
4. 디렉토리에 있는 파일의 크기나 이름을 알아낼 수 있나요?
5. 파일을 암호화하는 프로그램을 작성할 수 있나요?



스트림(stream)

- 스트림(stream)은 "순서가 있는 데이터의 연속적인 흐름"이다.
- 스트림은 입출력을 물의 흐름처럼 간주하는 것이다.

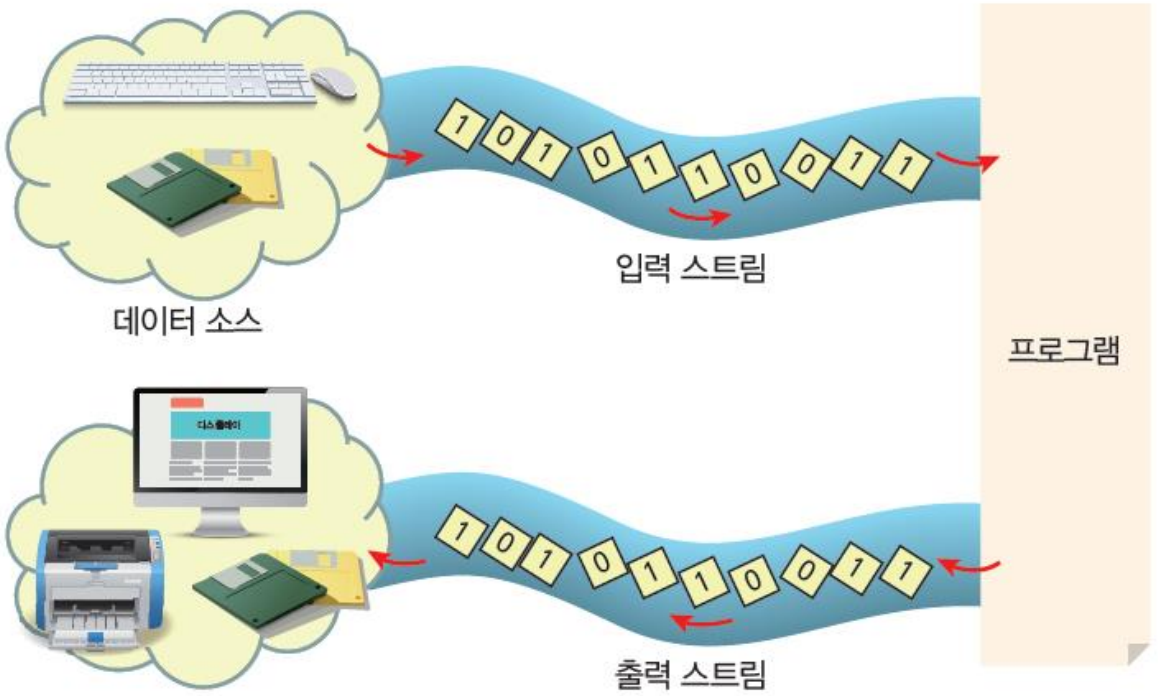


그림 15.1 스트림의 개념

스트림의 분류

- 입출력의 단위에 따라서 분류

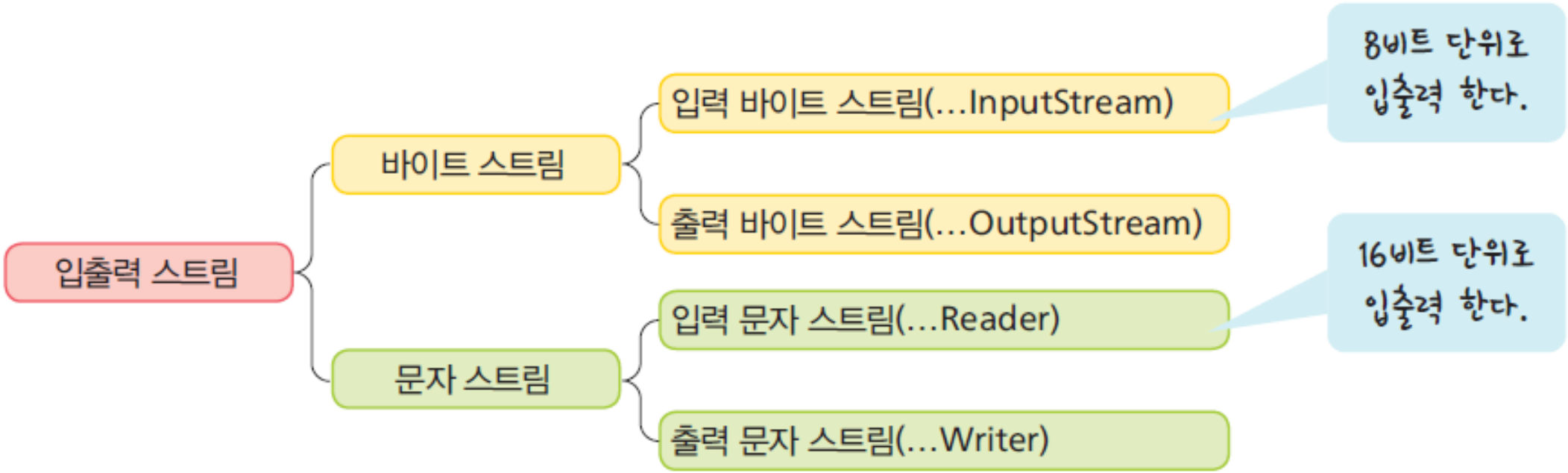


그림 15.2 스트림의 분류

바이트 스트림과 문자 스트림

- 바이트 스트림(byte stream)은 바이트 단위로 입출력하는 클래스
- 바이트 스트림 클래스들은 추상 클래스인 InputStream와 OutputStream에서 파생된다.
- 바이트 스트림 클래스 이름에는 InputStream(입력)과 OutputStream(출력)이 붙는다.

- 문자 스트림(character stream)은 문자 단위로 입출력하는 클래스
- 이들은 모두 기본 추상 클래스인 Reader와 Writer 클래스에서 파생된다.
- 문자 스트림 클래스 이름에는 Reader(입력)와 Writer(출력)가 붙는다.

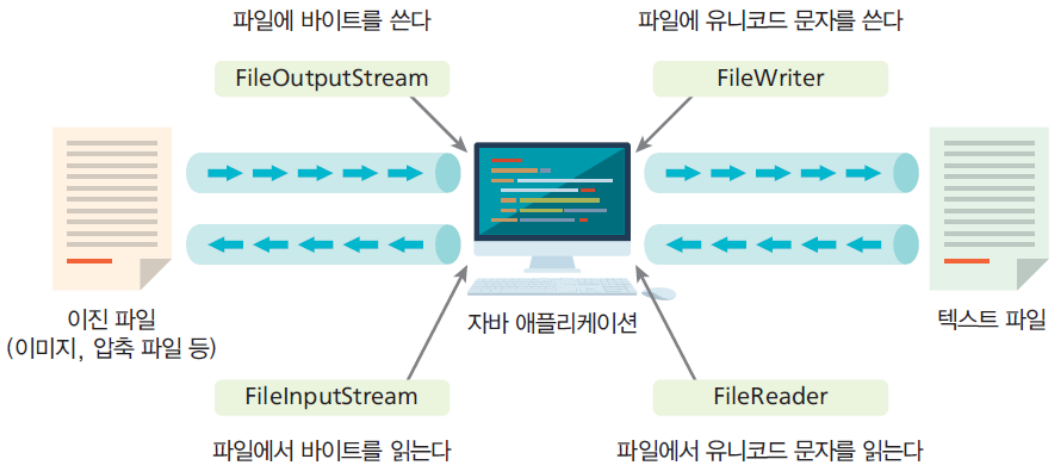


그림 15.2 파일 입출력 바이트 스트림

문자 스트림

- 문자 스트림(a)에서는 입출력 단위가 문자 (바이트가 아니다!)
- 자바 플랫폼은 유니코드를 사용해서 문자를 저장
- 문자 스트림은 자동적으로 이 유니코드 문자를 지역 문자 집합으로 변환한다.

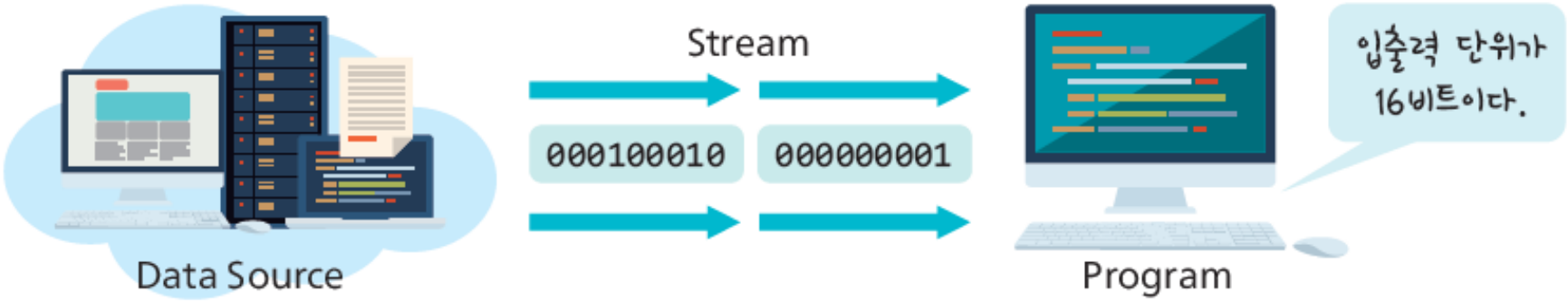
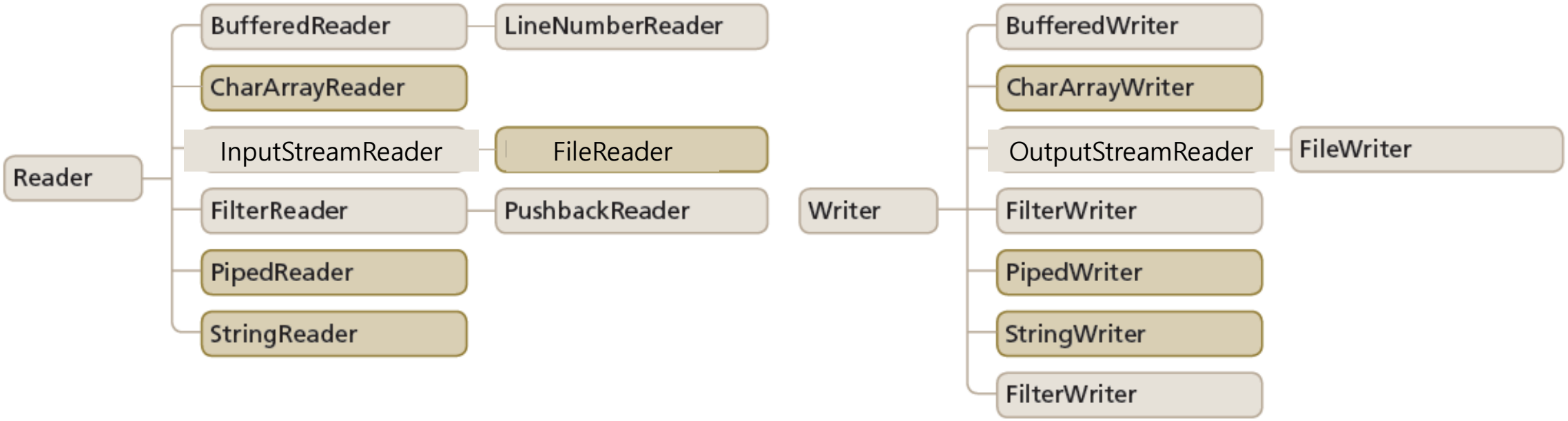


그림 15.4 문자 스트림의 개념

문자 스트림


Reader
BufferedReader
LineNumberReader
CharArrayReader
InputStreamReader
FileReader
FilterReader
PushbackReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
OutputStreamReader
FileWriter
FilterWriter
PipedWriter
StringWriter
FilterWriter

문자 스트림

- 파일에서 문자를 읽거나 쓰려면 FileReader와 FileWriter를 사용한다. 문자 스트림에서는 read()와 write() 메소드가 주력 메소드이다.

반환형	메소드	설명
int	read()	입력 스트림에서 한개의 문자를 읽는다. 반환값은 0에서 65535 범위(0x0000에서 0xFFFF)의 유니코드 값이다. 스트림이 종료되면 -1을 반환한다.
int	read(char[] cbuf)	입력 스트림에서 문자를 읽어서 cbuf[]에 저장하고 읽은 개수를 반환한다.
int	skip(long n)	입력 스트림에서 n만큼의 문자를 건너뛴다
void	close()	입력 스트림을 닫는다.

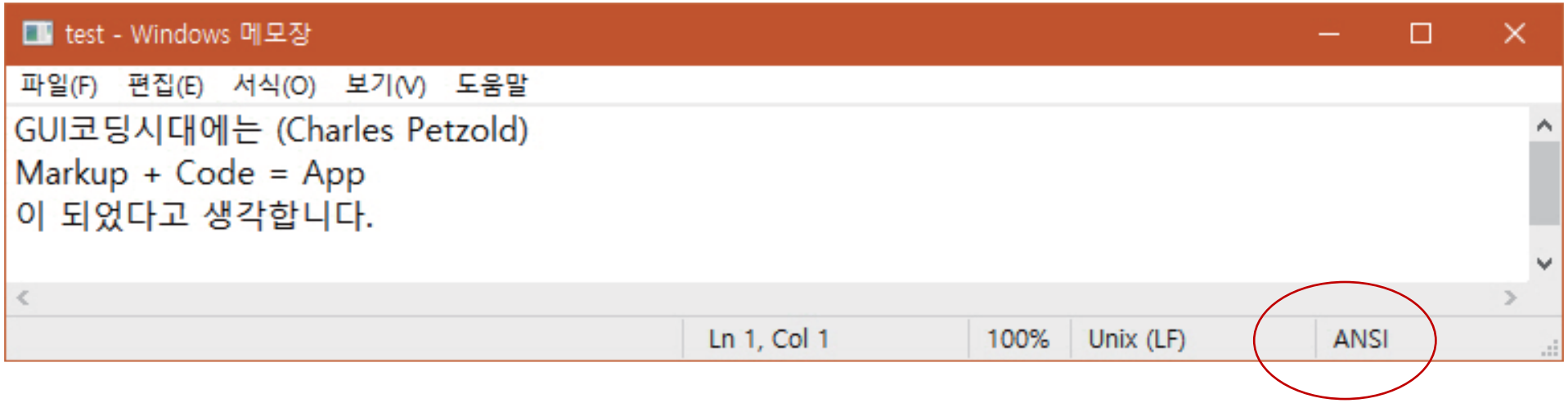
Reader 클래스의 주요 메소드

- 파일에서 문자들을 읽는 경우 일반적으로 다음과 같은 반복 루프를 사용한다.

```
int ch;
while ((ch = fr.read()) != -1)
    System.out.print((char) ch +" ");
```

예제: 텍스트 파일 읽기

- 기본 예제로 하드 디스크에 있는 한글 텍스트 파일을 읽어서 화면에 출력하는 프로그램을 작성해보자. 이때 텍스트 파일은 ANSI 엔코딩으로 저장하여야 한다. UTF-8로 저장하면 문자들이 깨져서 출력된다.



예제: 텍스트 파일 읽기

```
import java.io.*;

public class FileReaderExample2 {
        public static void main(String args[]) {
                FileReader fr;
                try {                                                   // (1)
                        fr = new FileReader("test.txt"); // (2)
                        int ch;
                        while ((ch = fr.read()) != -1)                  // (3)
                                System.out.print((char) ch + " ");
                        fr.close();                                             // (4)
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

GUI코딩시대에는 (Charles Petzold)
Markup + Code = App
이 되었다고 생각합니다.

try-with-resources 사용

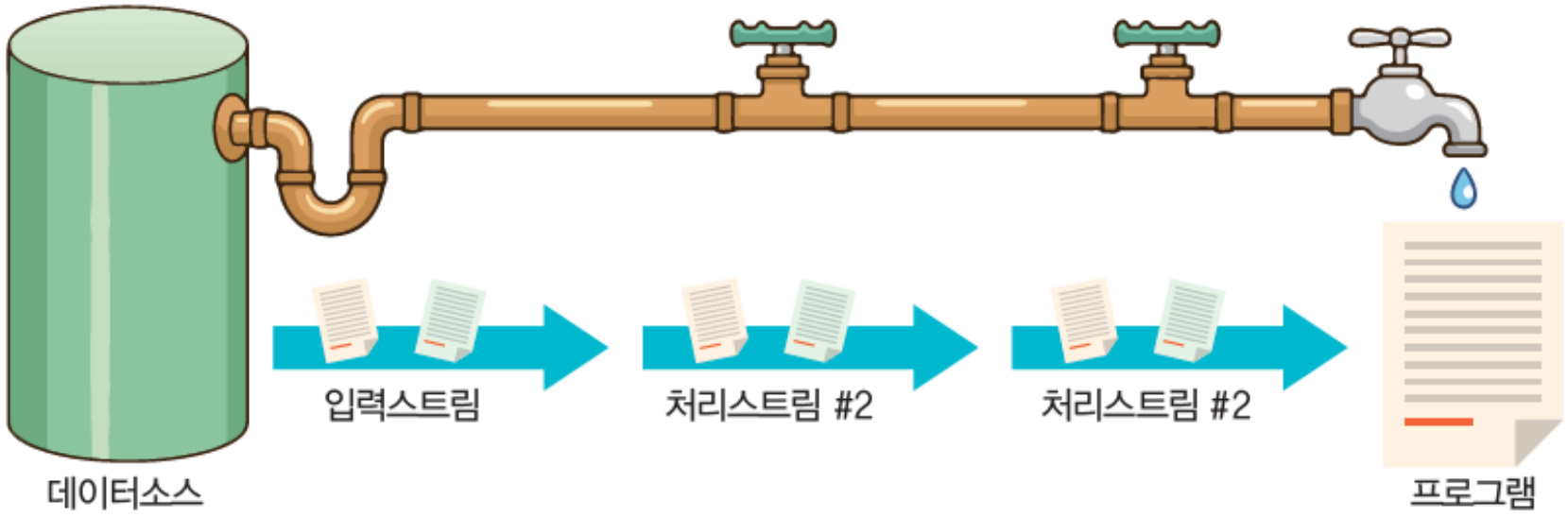
- try-with-resources 구문을 사용해보자. 이 경우에는 close()를 따로 호출하지 않아도 자동으로 호출된다.

```
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExample2 {
        public static void main(String args[]) throws Exception {
                try (FileReader fr = new FileReader("test.txt")) {
                        int ch;
                        while ((ch = fr.read()) != -1)
                                System.out.print((char) ch);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

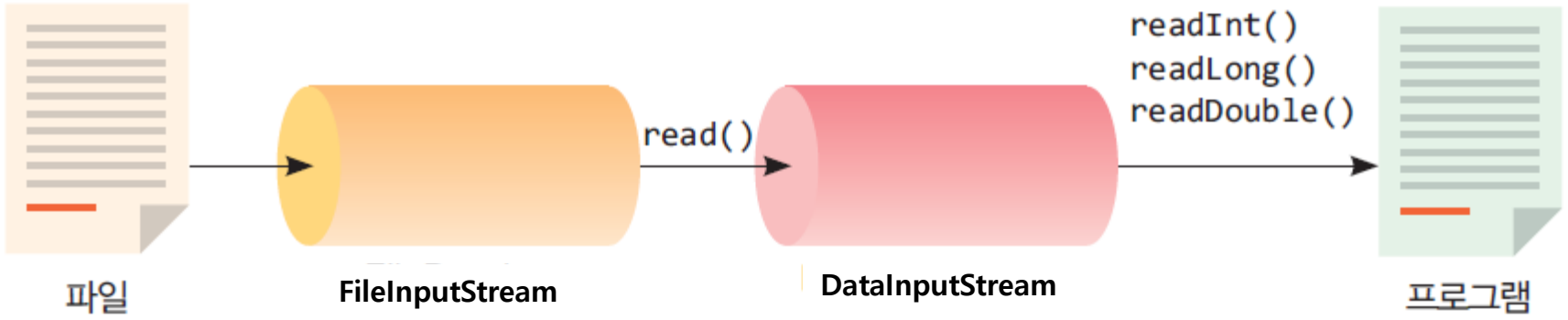
중간 처리 스트림

- 아래 그림처럼 파이프들이 서로 결합할 수 있듯이 스트림들도 서로 결합할 수 있다. 이렇게 되면 스트림을 통해 흘러가는 데이터에 대하여 다양한 가공 처리를 할 수 있다.



이진 파일에서 정수를 읽고 싶다면

```
DataInputStream dataSt = new DataInputStream(new FileInputStream("data.bin"));
int i = dataSt.readInt();
```



예제:

- 자료형이 다른 몇 개의 데이터를 파일에 출력하였다가 다시 읽어보자.

```
import java.io.*;

public class DataStreamTest {
        public static void main(String[] args) throws IOException {
                DataInputStream in = null;
                DataOutputStream out = null;
                try {
                        out = new DataOutputStream(new FileOutputStream("data.bin"));
                        out.writeInt(123);
                        out.writeFloat(123.456F);
                        out.close();

                        in = new DataInputStream(new FileInputStream("data.bin"));
                        int aint = in.readInt();
                        float afloat = in.readFloat();

                        System.out.println(aint);
                        System.out.println(afloat);
                }
```

예제:

- 자료형이 다른 몇 개의 데이터를 파일에 출력하였다가 다시 읽어보자.

```
                finally {
                        if (in != null) {
                                in.close();
                        }


                        if (out != null) {
                                out.close();
                        }
                }
        }
}
```

```
123
123.456
```

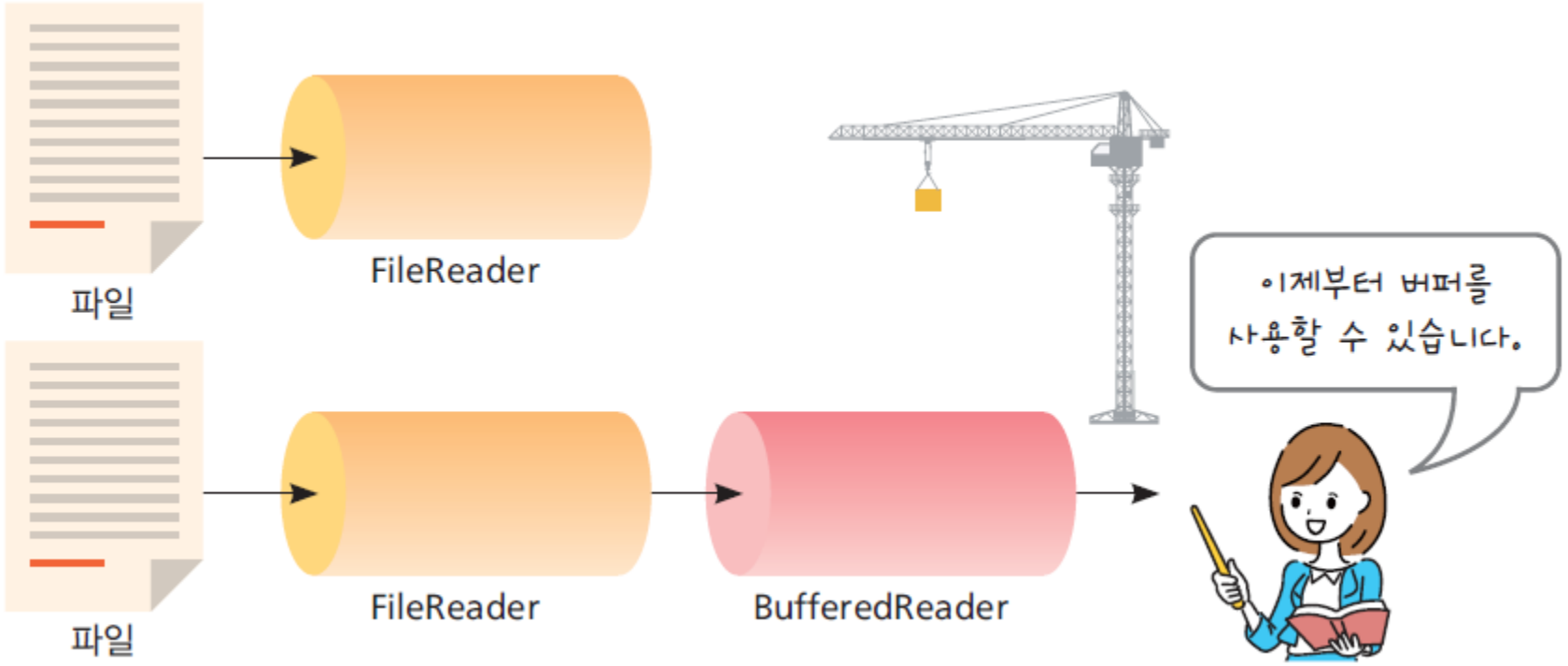
버퍼 스트림

- 버퍼 입력 스트림은 입력 장치에서 한번에 많이 읽어서 버퍼에 저장한다. 프로그램이 입력을 요구하면 버퍼에서 꺼내서 반환한다. 버퍼가 비었을 때만 입력 장치에서 읽는다.

그림 15.6 버퍼 스트림의 개념

버퍼 스트림

```
inputStream = new BufferedReader(new FileReader("input.txt"));
outputStream = new BufferedWriter(new FileWriter("output.txt"));
```



예제: 줄 단위로 복사하기

- 문자 단위가 아니라 한 줄 단위로 입출력해야 하는 경우도 종종 있다. 이럴 때는 BufferedReader와 PrintWriter 클래스를 사용하면 된다. 복사 프로그램을 줄 단위로 복사하도록 변경하여 보자.

```
import java.io.*;

public class CopyLines {

    public static void main(String[] args) {
        try( BufferedReader in = new BufferedReader(new FileReader("test.txt"));
             PrintWriter out = new PrintWriter(new FileWriter("output.txt"))) {
                String line;
                while (( line = in.readLine()) != null) {
                    out.println( line);
                }
        } catch (IOException e) {
                e.printStackTrace();
        }
    }
}
```

InputStreamReader와 OutputStreamWriter 클래스

- 바이트 스트림과 문자 스트림을 연결하는 두 개의 범용의 브릿지 스트림이 있다.

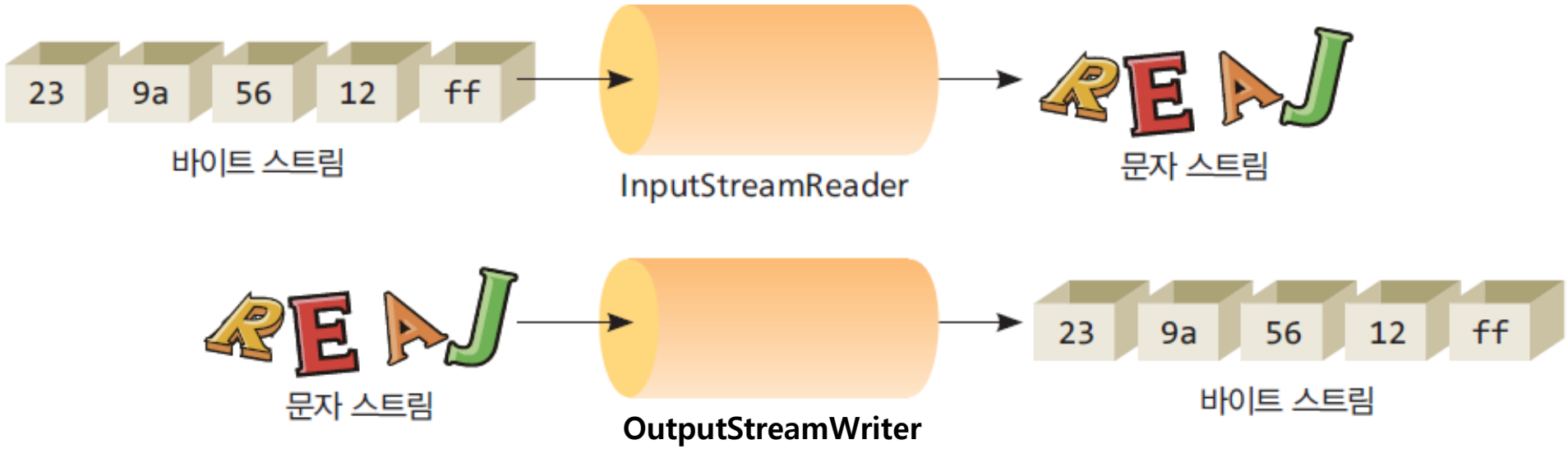
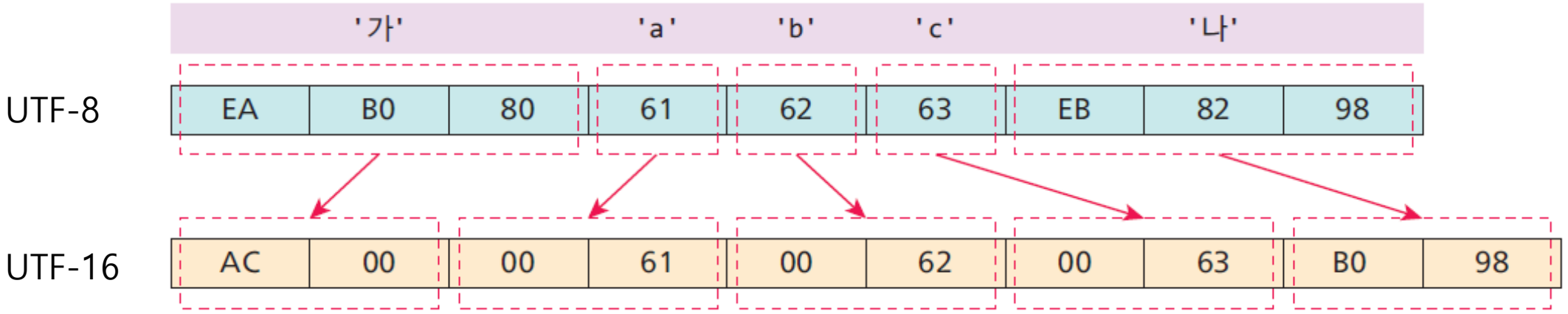


그림 15.6 브릿지 스트림

InputStreamReader

- InputStreamReader는 바이트 스트림을 문자 스트림으로 변환한다.

```
BufferedReader in = new BufferedReader(new InputStreamReader(
    new FileInputStream(fileDir), "UTF8"));
```



예제: UTF-8 코딩 파일 읽기


```
public class CharEncodingTest {
        public static void main(String[] args) throws IOException {
                File fileDir = new File("input.txt");
                BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(fileDir), "UTF8"));
                String str;
                while ((str = in.readLine()) != null) {
                        System.out.println(str);
                }
                in.close();
        }
}
```

```
GUI코딩시대에는 (Charles Petzold)
Markup + Code = App
이 되었다고 생각합니다.
```

예제: 줄단위로 입출력하기

- 문자 단위가 아니라 한 줄 단위로 입출력해야 하는 경우도 종종 있다. 이럴 때는 BufferedReader와 PrintWriter 클래스를 사용하면 된다.

```
public class CopyLines {
   public static void main(String[] args) throws IOException {
      BufferedReader inputStream = null;
      PrintWriter outputStream = null;
      try {
         inputStream = new BufferedReader(new FileReader("input.txt"));
         outputStream = new PrintWriter(new FileWriter("output.txt"));
         String l;
         while ((l = inputStream.readLine()) != null) {
            outputStream.println(l);
         }
      } finally {
         if (inputStream != null) { inputStream.close();   }
         if (outputStream != null) { outputStream.close();  }
      }
   }
}
```

객체 저장하기

- 객체를 파일에 저장하려면 어떻게 해야 할까? 물론 객체의 데이터를 하나씩 꺼내서 저장하여도 되지만 더 편리한 방법이 있다. -> 객체 직렬화

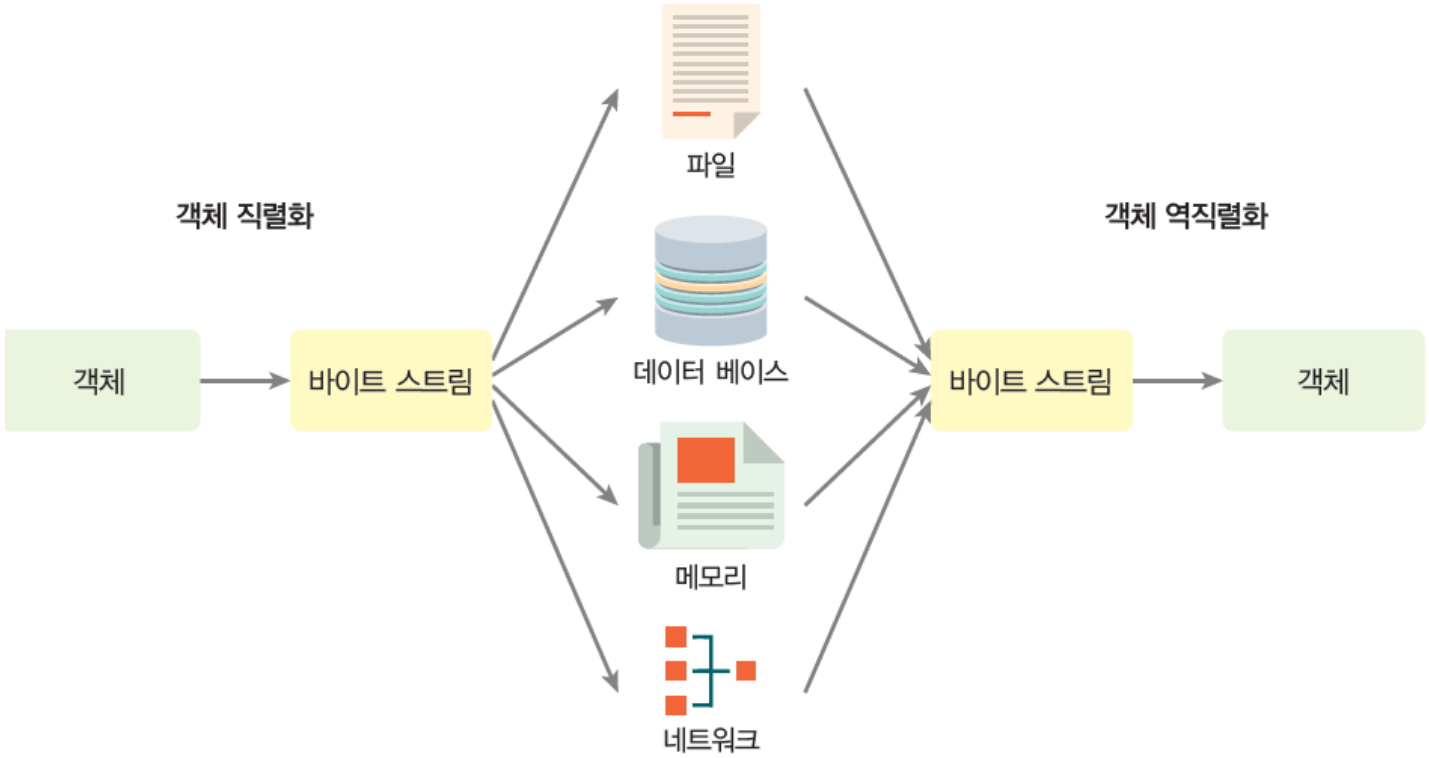


그림 15.7 객체 직렬화의 개념

객체 직렬화

- 객체 직렬화는 객체가 가진 데이터들을 순차적인 데이터로 변환한다. 순차적인 데이터가 되면 파일에 쉽게 저장할 수 있다. 어떤 클래스가 직렬화를 지원하려면 Serializable라는 인터페이스를 구현하면 된다.

- 객체가 직렬화된 데이터를 읽어서 자신의 상태를 복구하는 것을 역직렬화(deserialization)라고 한다.

예제: Date 객체 저장하기

- 자바가 기본적으로 제공하는 Date 클래스를 이용하여서 현재 날짜를 나타내는 객체를 저장하였다가 다시 읽어서 콘솔에 표시하는 소스는 다음과 같다.

```
public class ObjectStreamTest {
    public static void main(String[] args) throws Exception {
        ObjectInputStream in = null;
        ObjectOutputStream out = null;
            int c;

            out = new ObjectOutputStream(new FileOutputStream("object.dat"));
            out.writeObject(new Date());
            out.close();

            in = new ObjectInputStream(new FileInputStream("object.dat"));
            Date d = (Date) in.readObject();
            System.out.println(d);
            in.close();
    }
}
```

```
Sat Jan 06 14:46:32 KST 2018
```

Path 객체

- Path 클래스는 경로를 나타내는 클래스로서 "D:₩sources₩test.txt"와 같은 경로를 받아서 객체를 반환한다.

```
public class PathTest {

        public static void main(String[] args) {
                Path path = Paths.get("D:\\sources\\test.txt");
                System.out.println("전체 경로: "+ path);
                System.out.println("파일 이름: "+path.getFileName());

                System.out.println("부모 이름: "+path.getParent().getFileName());
        }
}
```

```
전체 경로: D:\sources\test.txt
파일 이름: test.txt
부모 이름: sources
```

File 객체

- File 클래스는 파일을 조작하고 검사하는 코드를 쉽게 작성하게 해주는 클래스이다. File 객체는 파일이 아닌 파일 이름을 나타내는 객체이다.
  - File file = new File("data.txt");

표 15.1 File 클래스의 메소드

반환형	메소드	설명
boolean	canExecute()	파일을 실행할 수 있는지의 여부
boolean	canRead()	파일을 읽을 수 있는지의 여부
boolean	canWrite()	파일을 변경할 수 있는지의 여부
static File	createTempFile(String prefix, String suffix)	임시 파일을 생성
boolean	delete()	파일을 삭제
void	deleteOnExit()	가상 기계가 종료되면 파일을 삭제
boolean	exists()	파일의 존재 여부
String	getAbsolutePath()	절대 경로를 반환
String	getCanonicalPath()	정규 경로를 반환
String	getName()	파일의 이름을 반환
String	getParent()	부모 경로 이름을 반환

예제: 파일 속성 알아보기

- 특정 디렉터리 안의 각 파일에 대하여 파일의 속성을 표시하여 보자.

```
==============================
이름: .eclipseproduct
경로: c:\eclipse\.eclipseproduct
부모: c:\eclipse
절대경로: c:\eclipse\.eclipseproduct
정규경로: C:\eclipse\.eclipseproduct
디렉터리 여부:false
파일 여부:true
==============================
...
```

```
public class FileTest {
  public static void main(String[] args) throws IOException {
    String name = "c:/eclipse";
    File dir = new File(name);
    String[] fileNames = dir.list(); // 현재 디렉토리의 전체 파일 리스트
    for (String s : fileNames) {
        File f = new File(name + "/" + s); // 절대 경로로 이름을 주어야 함
        System.out.println("==============================");
        System.out.println("이름: " + f.getName());
        System.out.println("경로: " + f.getPath());
        System.out.println("부모: " + f.getParent());
        System.out.println("절대경로: " + f.getAbsolutePath());
        System.out.println("정규경로: " + f.getCanonicalPath());
        System.out.println("디렉토리 여부:" + f.isDirectory());
        System.out.println("파일 여부:" + f.isFile());
        System.out.println("==============================");
    }
  }
}
```

스트림 라이브러리로 파일 처리하기

- 파일 처리에서도 스트림 라이브러리를 사용할 수 있다. 예를 들어서 현재 디렉터리의 모든 파일을 출력하는 코드는 다음과 같이 작성할 수도 있다.

```
Files.list(Paths.get(".")).forEach(System.out::println);
```

- 예를 들어서 파일을 읽어서 각 줄 끝에 있는 불필요한 공백을 제거하고 빈 줄을 필터링한 후에 출력하는 코드는 다음과 같다.

```
Files.lines(new File("test.txt").toPath())
        .map(s -> s.trim())
        .filter(s -> !s.isEmpty())
        .forEach(System.out::println);
```

Lab:

- 특정한 디렉터리 안의 C 소스 파일을 모두 찾아서 소스의 첫 번째 줄에 다음과 같은 문장을 추가하는 자바 프로그램을 작성해보자.
  - #define _CRT_SECURE_NO_WARNINGS

```
#include <stdio.h>
int main(void)
{
   printf("Hello");
   return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main(void)
{
   printf("Hello");
   return 0;
}
```

Sol:

```
public class Test {
    public static void main(String args[]) throws IOException {

        File directoryPath = new File("D:/src");
        File filesList[] = directoryPath.listFiles();

        Scanner sc = null;
        for (File file : filesList) {
            System.out.println("파일 이름: " + file.getName());
            System.out.println("파일 경로: " + file.getAbsolutePath());
            System.out.println("파일 크기:" + file.getTotalSpace());

            sc = new Scanner(file);
            String input;
            StringBuffer sb = new StringBuffer();

            sb.append("#define _CRT_SECURE_NO_WARNINGS\n");
            while (sc.hasNextLine()) {
                input = sc.nextLine();
                sb.append(input + "\n");
            }
```

Sol:

```
                String oldName = file.getAbsolutePath();
                String fileName;
                if (oldName.indexOf(".") > 0)
                    fileName = oldName.substring(0, oldName.lastIndexOf("."));
                else
                    fileName = oldName;

                System.out.println(fileName);
                BufferedWriter writer = new BufferedWriter(new FileWriter(fileName + "1.c"));
                writer.write(sb.toString());
                writer.close();
            }
        }
}
```

Lab: 압축 파일

- 여기서는 ZIP 방식으로 압축된 파일을 압축 해제하여 원본 파일을 얻는 방법을 살펴보자.

```
public class UnzipTest {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.println("압축 파일 이름을 입력하시오: ");
        String inname = sc.next();
        System.out.println("원본 파일 이름을 입력하시오: ");
        String outname = sc.next();
        ZipInputStream inStream = new ZipInputStream(new FileInputStream(inname));
        OutputStream outStream = new FileOutputStream(outname);
        byte[] buffer = new byte[1024];
        int read;
        ZipEntry entry;
        if ((entry = inStream.getNextEntry()) != null) {
            while ((read = inStream.read(buffer)) > 0) {
                outStream.write(buffer, 0, read);
            }
        }
        outStream.close();
        inStream.close();
    }
}
```

```
압축 파일 이름을 입력하시오: test.zip
원본 파일 이름을 입력하시오: test.txt
```

Lab: 파일 암호화하기

- 암호화 방법은 XOR 암호화 방법이다. 이 알고리즘에서는 파일 안의 모든 문자에 대하여 암호키와 비트 XOR 연산자를 적용한다. 출력을 해독하려면 동일한 키를 사용하여 XOR 함수를 다시 적용하면 된다.

Sol: 파일 암호화하기

```
public class XorEnc {
        static byte[] key = { 10, 20, 30, 40 };
        public static void main(String[] args) throws Exception {
                FileInputStream is = new FileInputStream("test.txt");
                FileOutputStream os = new FileOutputStream("test.enc");

                byte[] data = new byte[1024];
                int read = is.read(data);
                int index = 0;
                while( read != -1 ) {
                   for( int k=0; k<read; k++ ) {
                      data[k] ^= key[index % key.length];
                      index++;
                   }
                   os.write(data, 0, read);
                   read = is.read(data);
                }
                os.flush();
                os.close();
                is.close();
        }
}
```

Sol: 파일에서 특정 문자 횟수 세기

- 주어진 파일에서 특정한 문자 a가 파일에 나타나는 횟수를 세는 예제를 작성하라.
- 파일 이름은 사용자가 입력할 수 있도록 하라.

```
파일 이름 입력: test.txt
'a'의 횟수: 1
```

```
public class CountLetter {
   public static int getCount(char c, File f) throws Exception {
      int count = 0;
      try (InputStream in = new FileInputStream(f);
         BufferedReader reader = new BufferedReader(new InputStreamReader(in))) {
         String line = null;
         while ((line = reader.readLine()) != null) {
            for (int i = 0; i < line.length(); i++) {
               if (c == line.charAt(i)) {                    count++;                    }
            }
         }
      }
      return count;
   }
   public static void main(String[] args) throws Exception {
       Scanner sc = new Scanner(System.in);
       System.out.println("파일 이름 입력: ");
       String fname = sc.next();
      File file = new File(fname);
      int count = getCount('a', file);
      System.out.println("'의 횟수: "+count);
   }
}
```