KB금융그룹 | 국민의 평생 금융파트너

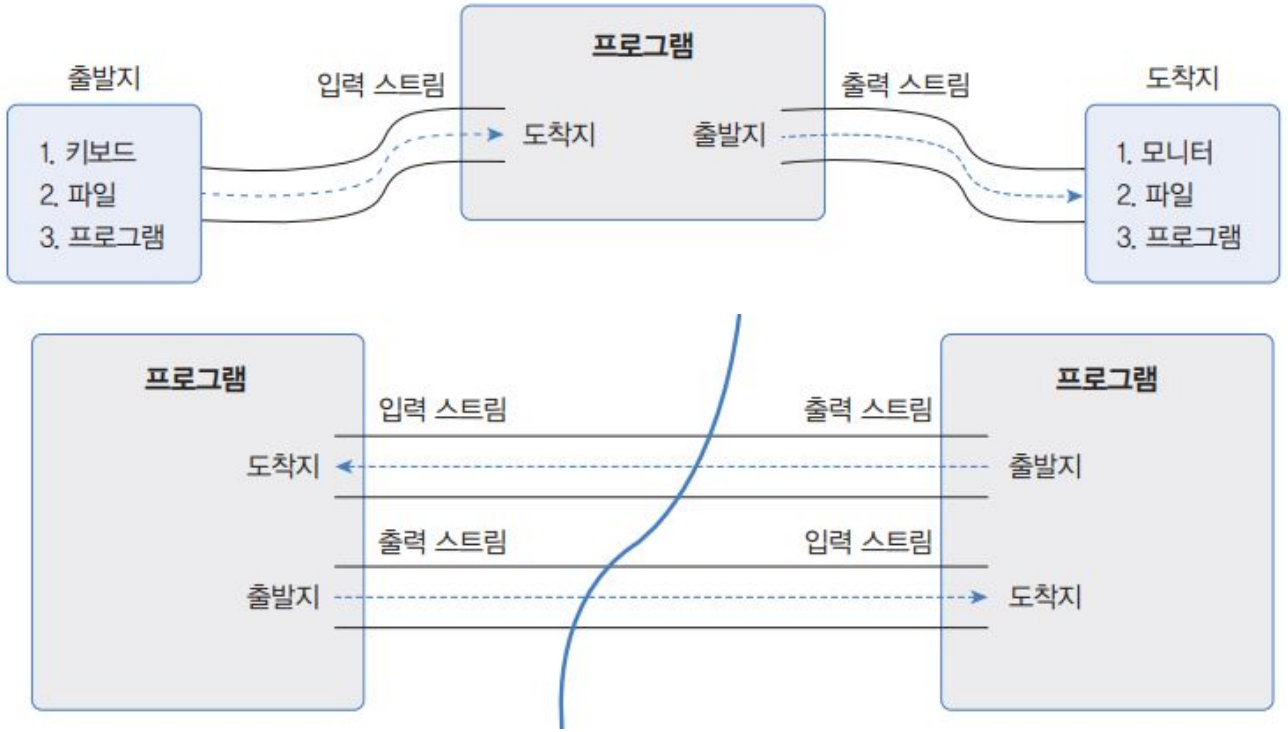
2025년 상반기 K-디지털 트레이닝

데이터 입출력

[KB] IT's Your Life

KB 국민은행

- **입력 스트림과 출력 스트림**
  - 프로그램을 기준으로 데이터가 들어오면 입력 스트림, 데이터가 나가면 출력 스트림
  - 프로그램이 다른 프로그램과 데이터를 교환하려면 양쪽 모두 입력 스트림과 출력 스트림이 필요

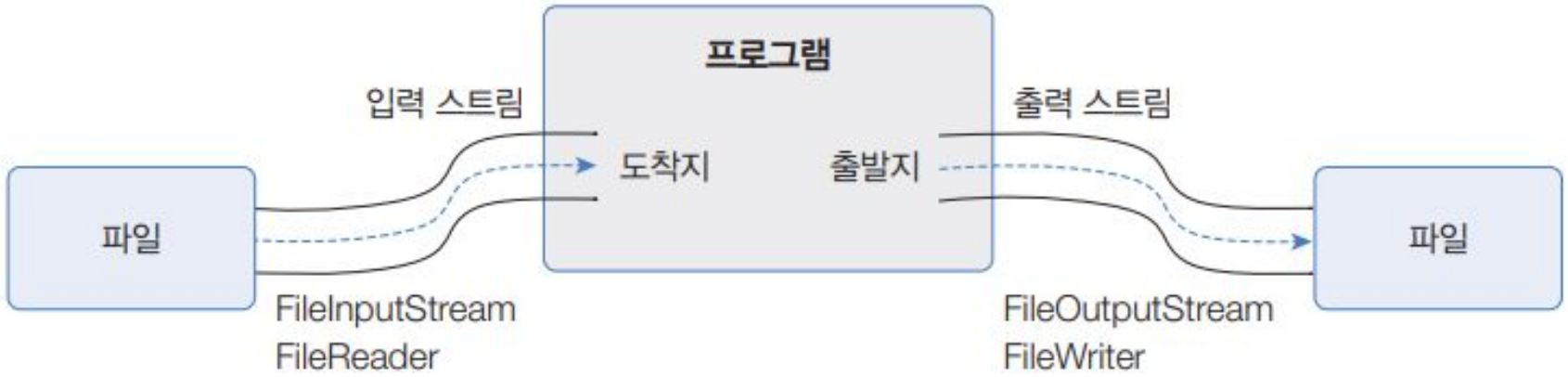


  - 바이트 스트림: 그림, 멀티미디어, 문자 등 모든 종류의 데이터를 입출력할 때 사용
  - 문자 스트림: 문자만 입출력할 때 사용

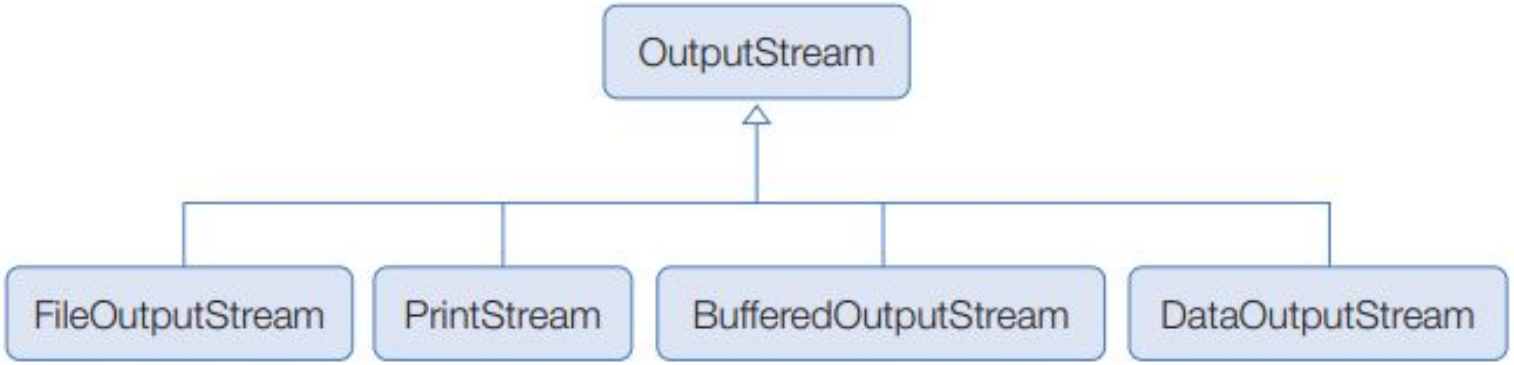
- **입력 스트림과 출력 스트림**
  - 자바는 데이터 입출력과 관련된 라이브러리를 java.io 패키지에서 제공

구분	바이트 스트림		문자 스트림	
	입력 스트림	출력 스트림	입력 스트림	출력 스트림
최상위 클래스	InputStream	OutputStream	Reader	Writer
하위 클래스 (예)	XXXInputStream (FileInputStream)	XXXOutputStream (FileOutputStream)	XXXReader (FileReader)	XXXWriter (FileWriter)

  - 바이트 입출력 스트림의 최상위 클래스는 InputStream과 OutputStream
  - 문자 입출력 스트림의 최상위 클래스는 Reader와 Writer



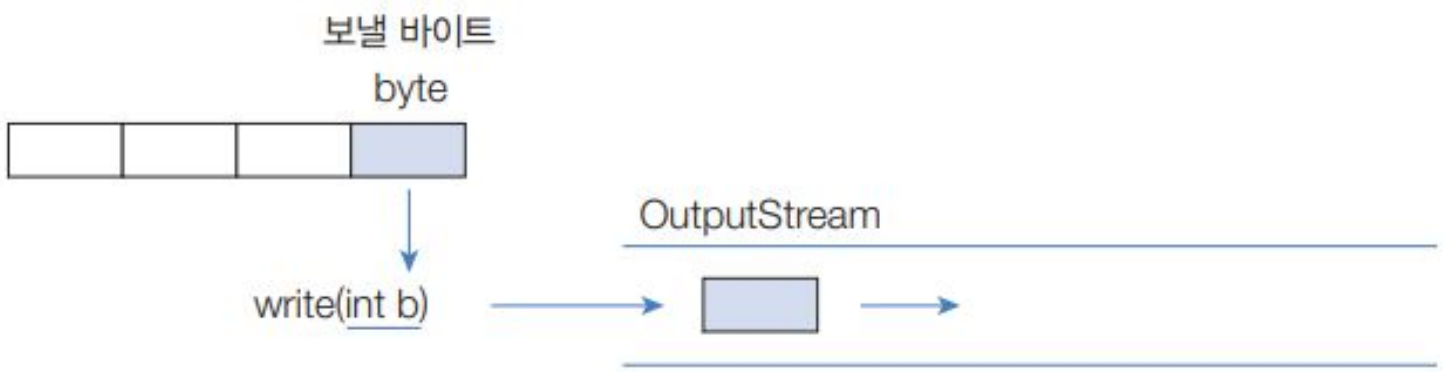
- **OutputStream**
  - OutputStream은 바이트 출력 스트림의 최상위 클래스로 추상 클래스
  - 모든 바이트 출력 스트림 클래스는 이 OutputStream 클래스를 상속받아서 만들어짐



- **OutputStream**
  - 모든 바이트 출력 스트림이 기본적으로 가져야 할 메소드가 정의됨

리턴 타입	메소드	설명
void	wrtie(int b)	1byte를 출력
void	write(byte[ ] b)	매개값으로 주어진 배열 b의 모든 바이트를 출력
void	write(byte[ ] b, int off, int len)	매개값으로 주어진 배열 b[off]부터 len개의 바이트를 출력
void	flush( )	출력 버퍼에 잔류하는 모든 바이트를 출력
void	close( )	출력 스트림을 닫고 사용 메모리 해제

- **1 바이트 출력**
  - write(int b) 메소드: 매개값 int(4byte)에서 끝 1byte만 출력. 매개변수는 int 타입



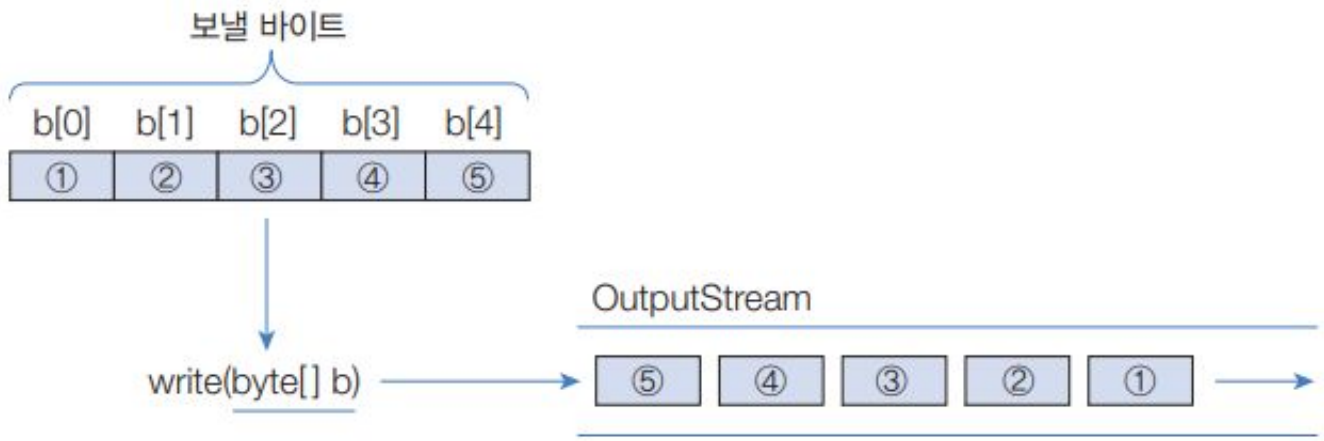
- **WriteExample.java**

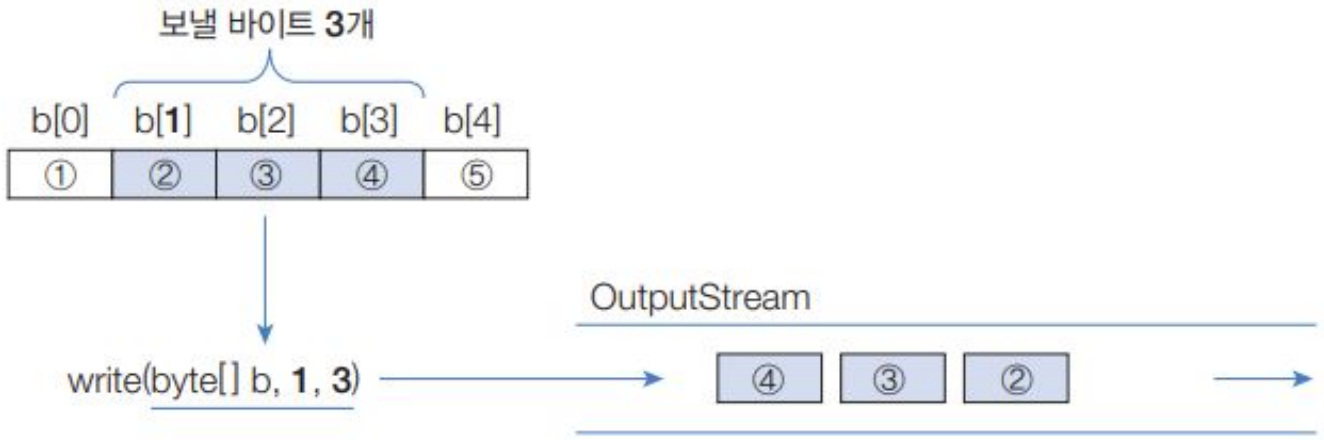
```
package ch18.sec02.exam01;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class WriteExample {
	public static void main(String[] args) {
		try {
			OutputStream os =
					new FileOutputStream("C:/temp/test1.db");

			byte a = 10;
			byte b = 20;
			byte c = 30;

			os.write(a);
			os.write(b);
			os.write(c);

			os.flush();
			os.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

- **WriteExample.java**

```
package ch18.sec02.exam01;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class WriteExample {
	public static void main(String[] args) {
		try(OutputStream os = new FileOutputStream("C:/temp/test1.db")) {
			byte a = 10;
			byte b = 20;
			byte c = 30;

			os.write(a);
			os.write(b);
			os.write(c);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

- **바이트 배열 출력**
  - write(byte[ ] b) 메소드: 매개값으로 주어진 배열의 모든 바이트를 출력
  - 배열의 일부분을 출력하려면 write(byte[ ] b, int off, int len) 메소드를 사용





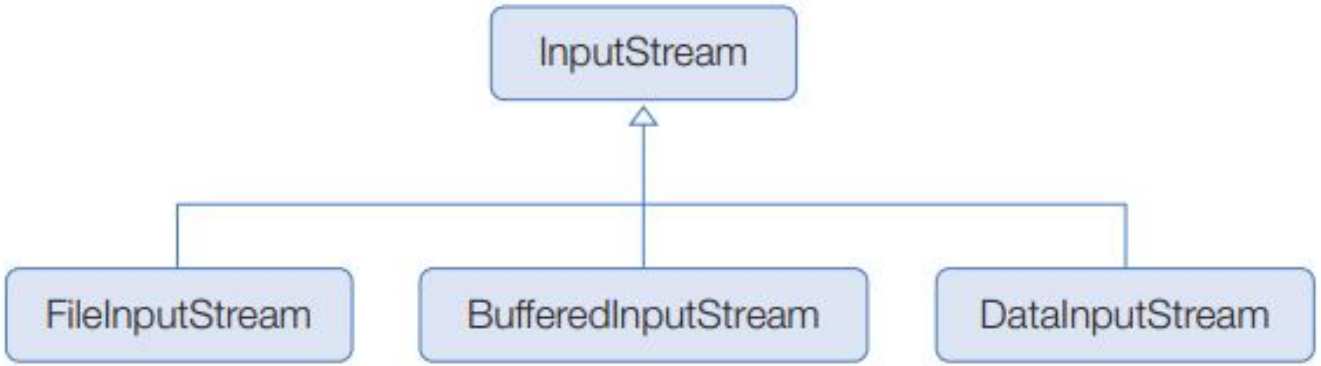
- **WriteExample.java**

```
package ch18.sec02.exam02;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class WriteExample {
	public static void main(String[] args) {
		try(OutputStream os = new FileOutputStream("C:/Temp/test2.db")) {
			byte[] array = { 10, 20, 30 };

			os.write(array);

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

- **WriteExample.java**

```
package ch18.sec02.exam02;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class WriteExample {
	public static void main(String[] args) {
		try(OutputStream os = new FileOutputStream("C:/Temp/test2.db")) {
			byte[] array = { 10, 20, 30, 40, 50 };

			os.write(array, 1, 3);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

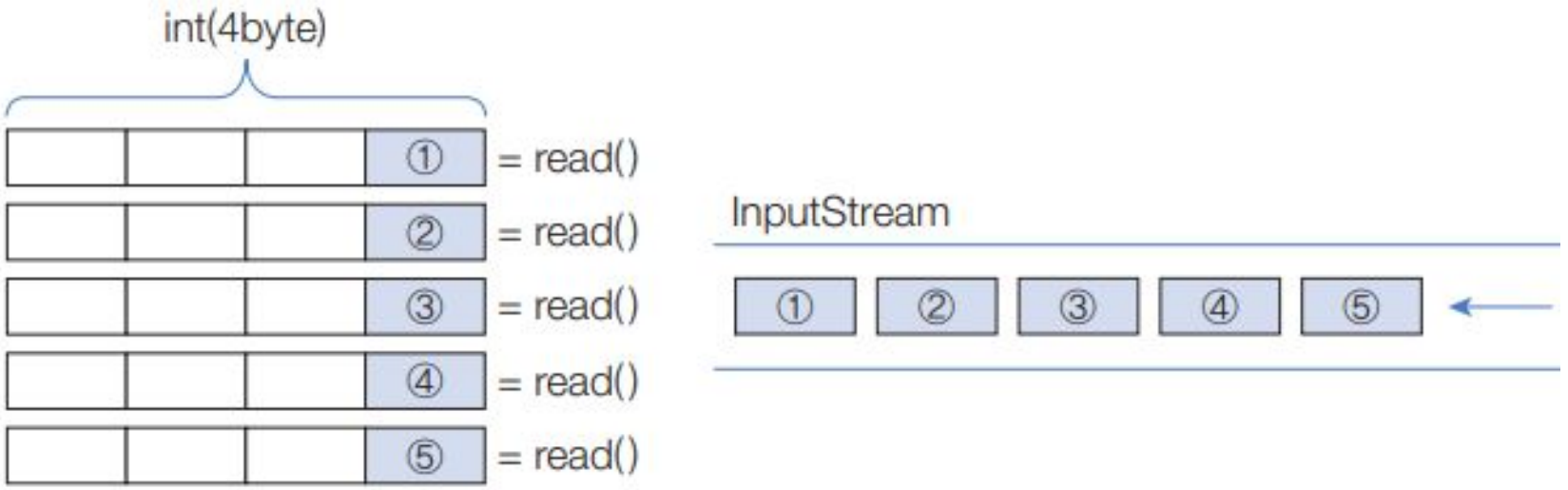
- **InputStream**
  - InputStream은 바이트 입력 스트림의 최상위 클래스로, 추상 클래스
  - 모든 바이트 입력 스트림은 InputStream 클래스를 상속받아 만들어짐



- **InputStream**
  - InputStream 클래스에는 바이트 입력 스트림이 기본적으로 가져야 할 메소드가 정의됨

리턴 타입	메소드	설명
int	read( )	1byte를 읽은 후 읽은 바이트를 리턴
int	read(byte[ ] b)	읽은 바이트를 매개값으로 주어진 배열에 저장 후 읽은 바이트 수를 리턴
void	close( )	입력 스트림을 닫고 사용 메모리 해제

- **1 바이트 입력**
  - read() 메소드
    - 입력 스트림으로부터 1byte를 읽고 int(4byte) 타입으로 리턴.
    - 리턴된 4byte 중 끝 1byte에만 데이터가 들어 있음



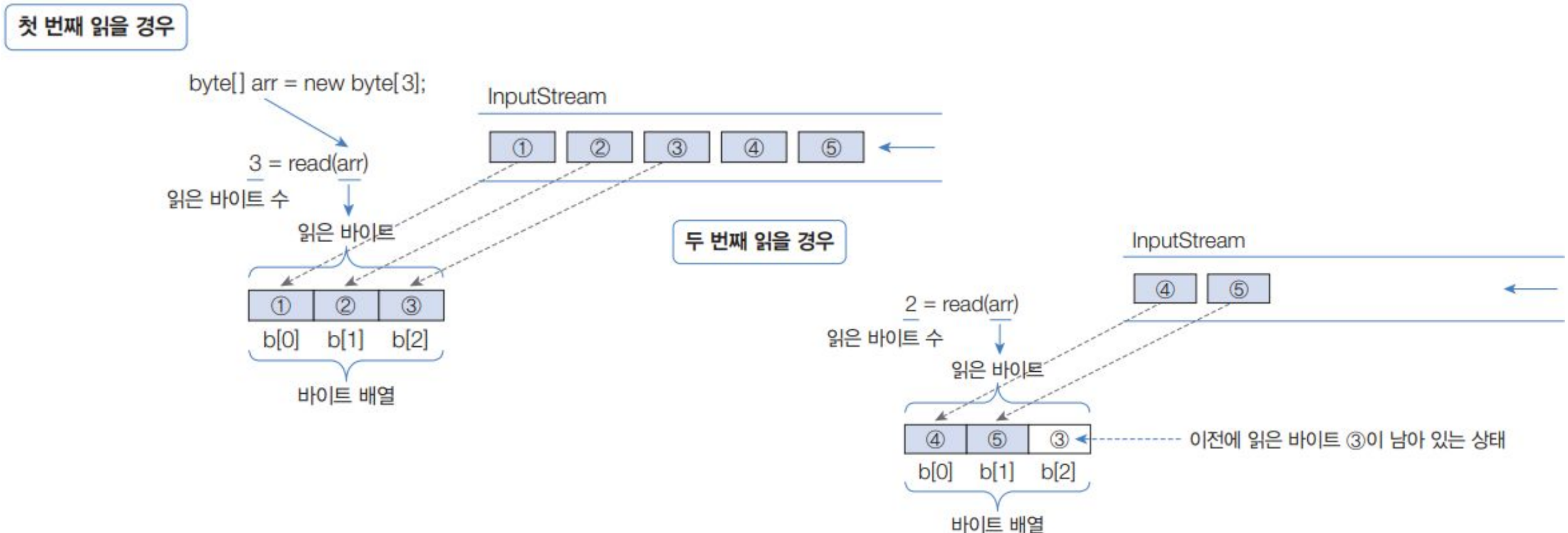
- **1 바이트 입력**
  - 더 이상 입력 스트림으로부터 바이트를 읽을 수 없다면 read() 메소드는 -1을 리턴.
  - 읽을 수 있는 마지막 바이트까지 반복해서 한 바이트씩 읽을 수 있음

```
InputStream is = …;
while (true) {
  int data = is.read();     //1 바이트를 읽고 리턴
  if (data == -1) break;    //-1을 리턴했을 경우 while 문 종료
}
```

- **ReadExample.java**

```
package ch18.sec03.exam01;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class ReadExample {
	public static void main(String[] args) {
		try (InputStream is = new FileInputStream("C:/Temp/test1.db")){
			while(true) {
				int data = is.read();		// 1byte씩 읽기
				if(data == -1) break;		// 파일 끝에 도달했을 경우
				System.out.println(data);
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

- **바이트 배열로 읽기**
  - read(byte[ ] b) 메소드
    - 입력 스트림으로부터 주어진 배열의 길이만큼 바이트를 읽고 배열에 저장한 다음 읽은 바이트 수를 리턴
    - 입력 스트림으로부터 바이트를 더 이상 읽을 수 없다면 -1을 리턴.
    - 읽을 수 있는 마지막 바이트까지 반복해서 읽을 수 있음



- 바이트 배열로 읽기

```
InputStream is = …;
byte[] data = new byte[100];
while (true) {
  int num = is.read(data);    //최대 100byte를 읽고, 읽은 바이트는 배열 data 저장, 읽은
                                수는 리턴
  if (num == -1) break;       //-1을 리턴하면 while 문 종료
}
```

- **ReadExample.java**

```
package ch18.sec03.exam02;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class ReadExample {
	public static void main(String[] args) {
		try(InputStream is =
				new FileInputStream("C:/Temp/test2.db")) {

			byte[] data = new byte[100];

			while(true) {
				int num = is.read(data);   // 최대 100 byte 읽기
				if(num == -1) break;       // 파일 끝 도달

				for(int i=0; i<num; i++) {
					System.out.println(data[i]);
				}
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

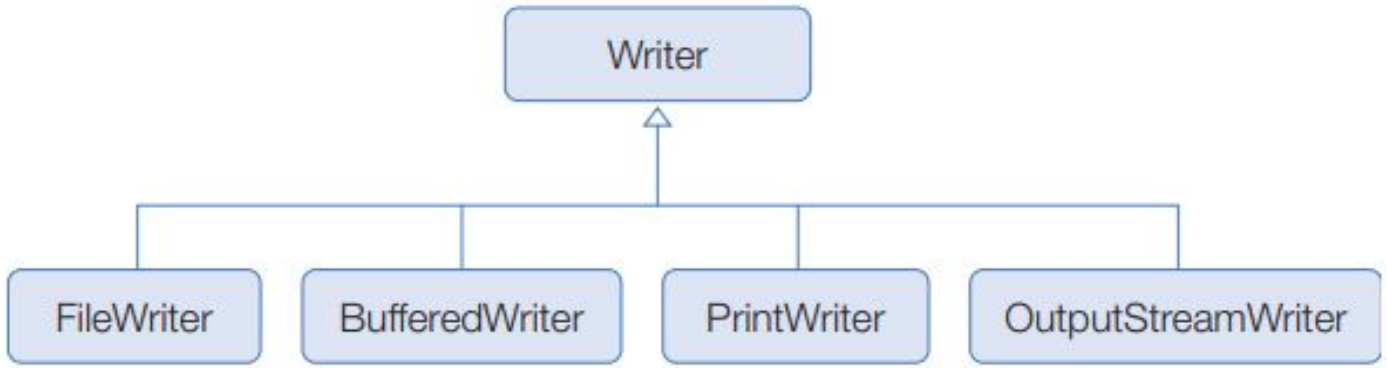
- **CopyExample.java**

```
package ch18.sec03.exam03;
…

public class CopyExample {
        public static void main(String[] args) throws Exception {
                String originalFileName = "C:/Temp/test.jpg";
                String targetFileName = "C:/Temp/test2.jpg";

                try(
                        InputStream is = new FileInputStream(originalFileName);
                        OutputStream os = new FileOutputStream(targetFileName);
                ) {

                        byte[] data = new byte[1024];        // 배열 버퍼 생성
                        while (true) {
                                int num = is.read(data);              // 최대 1024바이트 읽기
                                if (num == -1) break;                       // 파일을 다 읽으면 while 문 종료
                                os.write(data, 0, num);               // 읽은 데이터 파일에 쓰기
                        }

                        os.flush();        // 내부 버퍼 잔류 바이트를 출력하고 버퍼를 비움
```

- **CopyExample.java**

```
                System.out.println("복사가 잘 되었습니다.");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

- **문자 출력**
  - Writer는 문자 출력 스트림의 최상위 클래스로, 추상 클래스.
  - 모든 문자 출력 스트림 클래스는 Writer 클래스를 상속받아서 만들어짐



- **문자 출력**
  - Writer 클래스에는 모든 문자 출력 스트림이 기본적으로 가져야 할 메소드가 정의됨

리턴 타입	메소드	설명
void	wrtie(int c)	매개값으로 주어진 한 문자를 출력
void	write(char[ ] cbuf)	매개값으로 주어진 배열의 모든 문자를 출력
void	write(char[ ] cbuf, int off, int len)	매개값으로 주어진 배열에서 cbuf[off]부터 len개까지의 문자를 출력
void	write(String str)	매개값으로 주어진 문자열을 출력
void	write(String str, int off, int len)	매개값으로 주어진 문자열에서 off 순번부터 len개까지의 문자를 출력
void	flush( )	버퍼에 잔류하는 모든 문자를 출력
void	close( )	출력 스트림을 닫고 사용 메모리를 해제

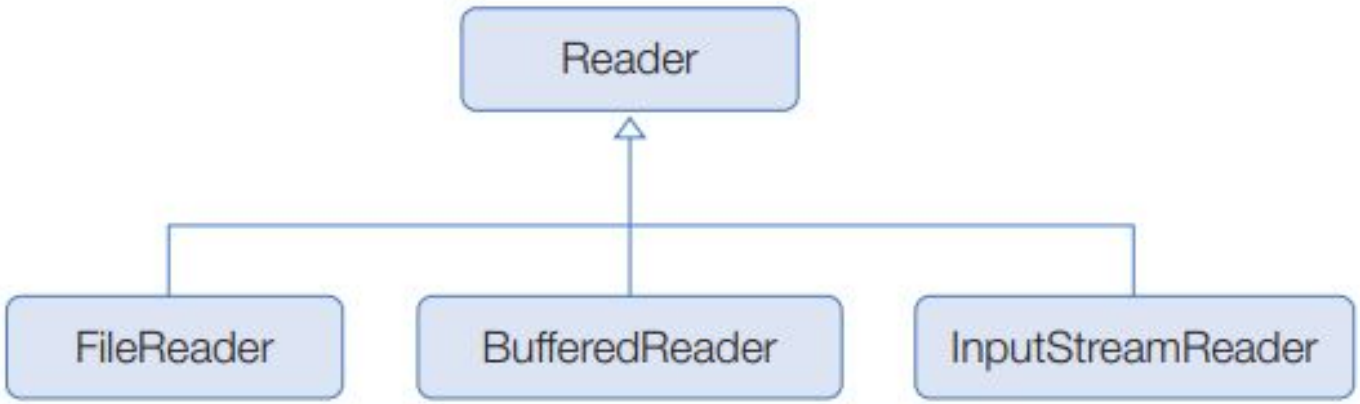
- **WriteExample.java**

```
package ch18.sec04.exam01;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

public class WriteExample {
	public static void main(String[] args) {
		//문자 기반 출력 스트림 생성
		try(Writer writer = new FileWriter("C:/Temp/test.txt")) {

			//1 문자씩 출력
			char a = 'A';
			writer.write(a);
			char b = 'B';
			writer.write(b);

			//char 배열 출력
			char[] arr = { 'C', 'D', 'E' };
			writer.write(arr);
```

- **CopyExample.java**

```
            //문자열 출력
            writer.write("FGH");

            //버퍼에 잔류하고 있는 문자들을 출력하고, 버퍼를 비움
            writer.flush();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Reader**
  - Reader는 문자 입력 스트림의 최상위 클래스로, 추상 클래스
  - 모든 문자 입력 스트림 클래스는 Reader 클래스를 상속받아서 만들어짐



- **Reader**
  - Reader 클래스에는 문자 입력 스트림이 기본적으로 가져야 할 메소드가 정의됨

메소드		설명
int	read( )	1개의 문자를 읽고 리턴
int	read(char[ ] cbuf)	읽은 문자들을 매개값으로 주어진 문자 배열에 저장하고 읽은 문자 수를 리턴
void	close( )	입력 스트림을 닫고, 사용 메모리 해제

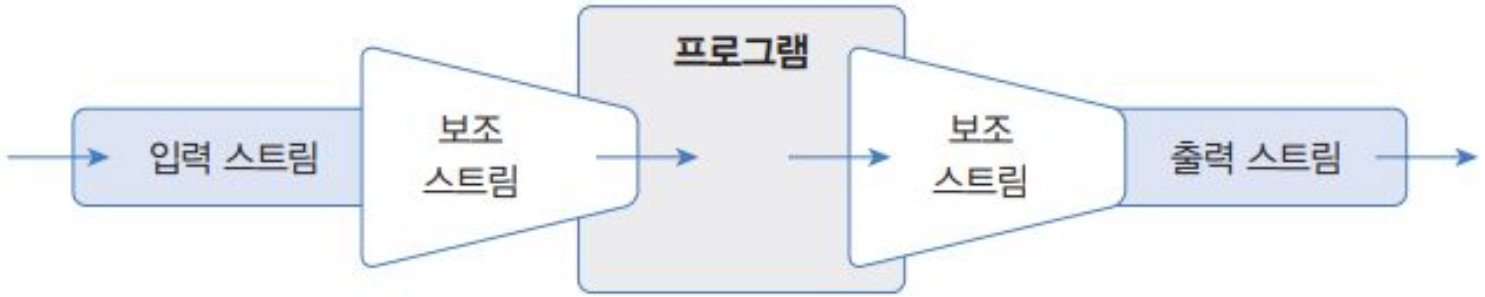
- **ReadExample.java**

```
package ch18.sec04.exam02;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;

public class ReadExample {
	public static void main(String[] args) {
		try {
			Reader reader = null;

			//1 문자씩 읽기
			reader = new FileReader("C:/Temp/test.txt");
			while(true) {
				int data = reader.read();
				if(data == -1) break;
				System.out.print((char)data);
			}
			reader.close();
			System.out.println();
```

- **ReadExample.java**

```
                //문자 배열로 읽기
                reader = new FileReader("C:/Temp/test.txt");
                char[] data = new char[100];
                while(true) {
                        int num = reader.read(data);
                        if(num == -1) break;
                        for(int i=0; i<num; i++) {
                                System.out.print(data[i]);
                        }
                }
                reader.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    }
}
```

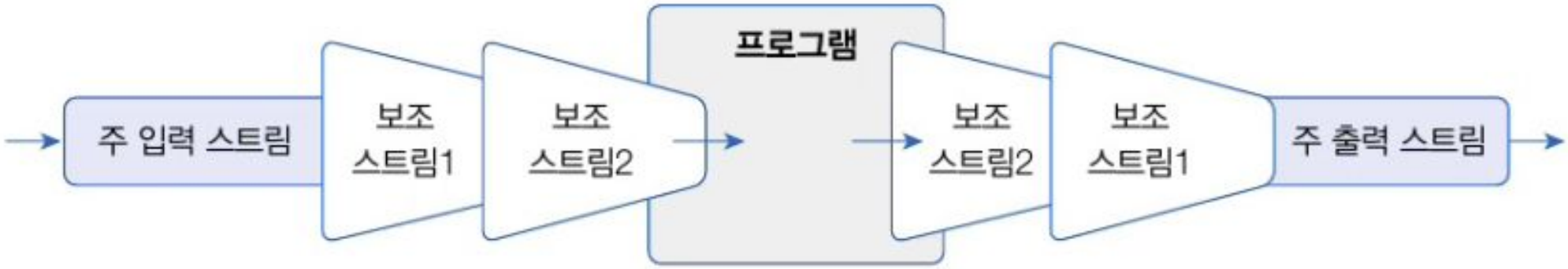
- **보조 스트림**
  - 다른 스트림과 연결되어 여러 편리한 기능을 제공해주는 스트림. 자체적으로 입출력을 수행할 수 없기 때문에 입출력 소스로부터 직접 생성된 입출력 스트림에 연결해서 사용



  - 입출력 스트림에 보조 스트림을 연결하려면 보조 스트림을 생성할 때 생성자 매개값으로 입출력 스트림을 제공

  - 보조스트림 변수 = new 보조스트림(입출력스트림);

- **스트림 체인 구성**
  - 여러 보조 스트림 연결해서 사용하기



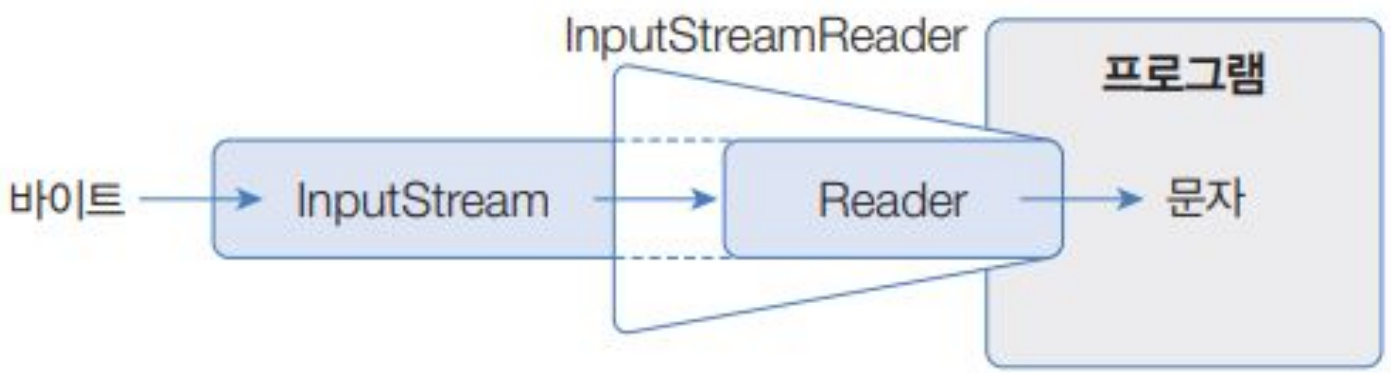
보조스트림2 변수 = new 보조스트림2(보조 스트림1);

```
InputStream is = new FileInputStream("…");
InputStreamReader reader = new InputStreamReader( is );
BufferedReader br = new BufferedReader( reader );
```

- 보조 스트림

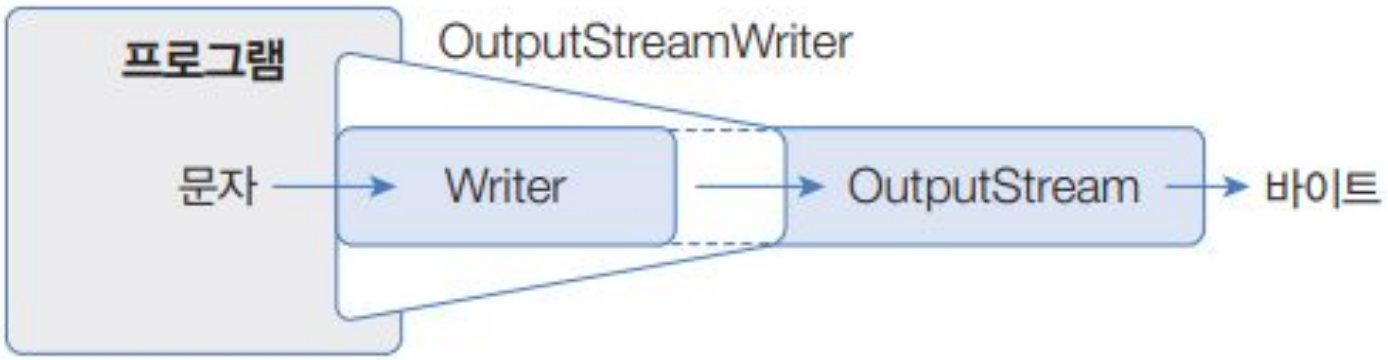
보조 스트림	기능
InputStreamReader	바이트 스트림을 문자 스트림으로 변환
BufferedInputStream, BufferedOutputStream BufferedReader, BufferedWriter	입출력 성능 향상
DataInputStream, DataOutputStream	기본 타입 데이터 입출력
PrintStream, PrintWriter	줄바꿈 처리 및 형식화된 문자열 출력
ObjectInputStream, ObjectOutputStream	객체 입출력

- **InputStream을 Reader로 변환**
  - InputStream을 Reader로 변환하려면 InputStreamReader 보조 스트림을 연결




```
InputStream is = new FileInputStream("C:/Temp/test.txt");
Reader reader = new InputStreamReader(is);
```

- **OutputStream을 Writer로 변환**
  - OutputStream을 Writer로 변환하려면 OutputStreamWriter 보조 스트림을 연결




```
OutputStream os = new FileOutputStream("C:/Temp/test.txt");
Writer writer = new OutputStreamWriter(os);
```

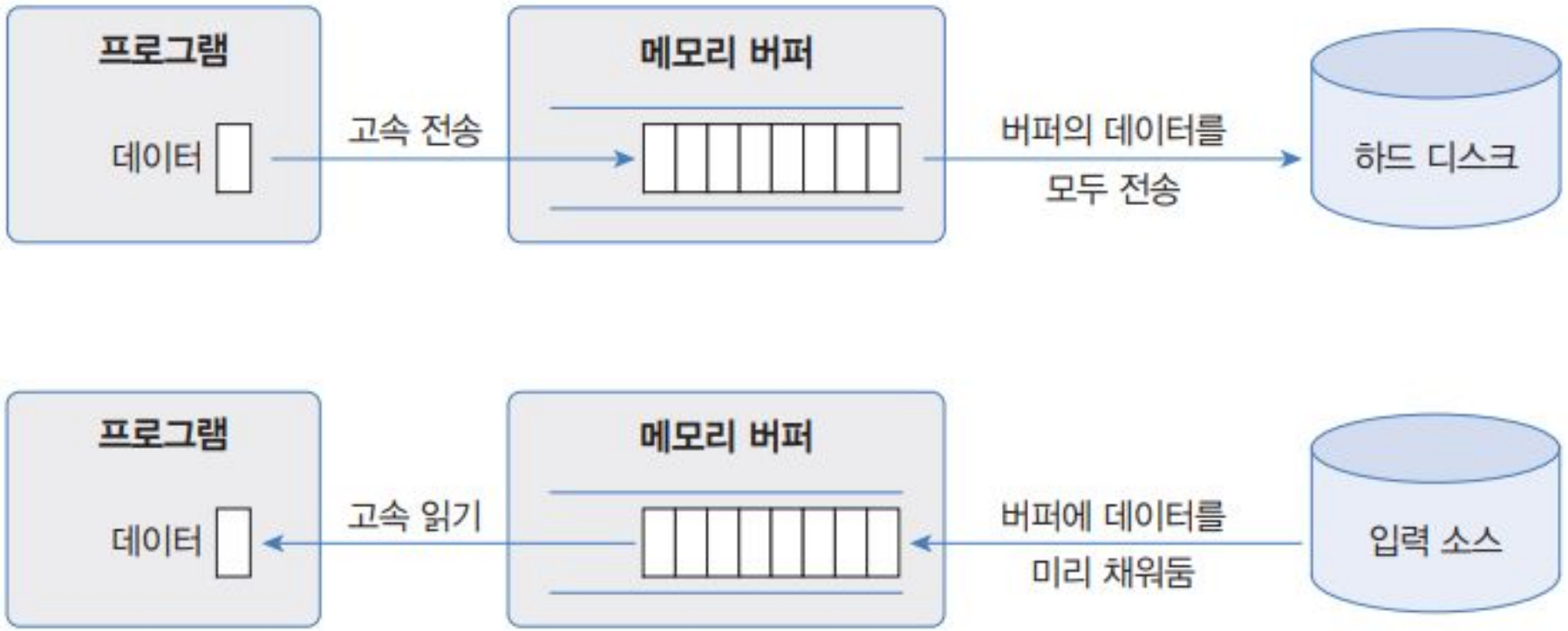
- **CharacterConvertStreamExample.java**

```
package ch18.sec06;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Reader;
import java.io.Writer;

public class CharacterConvertStreamExample {

	public static void main(String[] args) throws Exception {
		write("문자 변환 스트림을 사용합니다.");
		String data = read();
		System.out.println(data);
	}
```

- **CharacterConvertStreamExample.java**

```
    public static void write(String str) throws Exception {
        OutputStream os = new FileOutputStream("C:/Temp/test.txt");
        Writer writer = new OutputStreamWriter(os, "UTF-8");
        writer.write(str);
        writer.flush();
        writer.close();
    }

    public static String read() throws Exception {
        InputStream is = new FileInputStream("C:/Temp/test.txt");
        Reader reader = new InputStreamReader(is, "UTF-8");
        char[] data = new char[100];
        int num = reader.read(data);
        reader.close();
        String str = new String(data, 0, num);
        return str;
    }
}
```

- **메모리 버퍼로 실행 성능을 높이는 보조 스트림**
  - 프로그램이 중간에 메모리 버퍼buffer와 작업해서 실행 성능 향상 가능
  - 출력 스트림의 경우 직접 하드 디스크에 데이터를 보내지 않고 메모리 버퍼에 데이터를 보냄으로써 출력 속도를 향상. 입력 스트림에서도 버퍼를 사용하면 읽기 성능 향상



- 메모리 버퍼로 실행 성능을 높이는 보조 스트림
  - 바이트 스트림에는 BufferedInputStream, BufferedOutputStream이 있고 문자 스트림에는 BufferedReader, BufferedWriter가 있음

```
BufferedInputStream bis = new BufferedInputStream(바이트 입력 스트림);
BufferedOutputStream bos = new BufferedOutputStream(바이트 출력 스트림);
```

```
BufferedReader br = new BufferedReader(문자 입력 스트림);
BufferedWriter bw = new BufferedWriter(문자 출력 스트림);
```

- **BufferExample.java**

```
package ch18.sec07.exam01;

import java.io.*;

	public class BufferExample {
		public static void main(String[] args) throws Exception {
			//입출력 스트림 생성
			String originalFilePath1 =
					BufferExample.class.getResource("originalFile1.jpg").getPath();
			String targetFilePath1 = "C:/Temp/targetFile1.jpg";
			FileInputStream fis = new FileInputStream(originalFilePath1);
			FileOutputStream fos = new FileOutputStream(targetFilePath1);

			//입출력 스트림 + 버퍼 스트림 생성
			String originalFilePath2 =
					BufferExample.class.getResource("originalFile2.jpg").getPath();
			String targetFilePath2 = "C:/Temp/targetFile2.jpg";
			FileInputStream fis2 = new FileInputStream(originalFilePath2);
			FileOutputStream fos2 = new FileOutputStream(targetFilePath2);
			BufferedInputStream bis = new BufferedInputStream(fis2);
			BufferedOutputStream bos = new BufferedOutputStream(fos2);
```

- **BufferExample.java**

```
        //버퍼를 사용하지 않고 복사
        long nonBufferTime = copy(fis, fos);
        System.out.println("버퍼 미사용:\t" + nonBufferTime + " ns");

        //버퍼를 사용하고 복사
        long bufferTime = copy(bis, bos);
        System.out.println("버퍼 사용:\t" + bufferTime + " ns");

        fis.close();
        fos.close();
        bis.close();
        bos.close();
    }
```

- **BufferExample.java**

```
        public static long copy(InputStream is, OutputStream os) throws Exception {
            //시작 시간 저장
            long start = System.nanoTime();
            //1 바이트를 읽고 1 바이트를 출력
            while(true) {
                int data = is.read();
                if(data == -1) break;
                os.write(data);
            }
            os.flush();
            //끝 시간 저장
            long end = System.nanoTime();
            //복사 시간 리턴
            return (end-start);
        }
    }
```

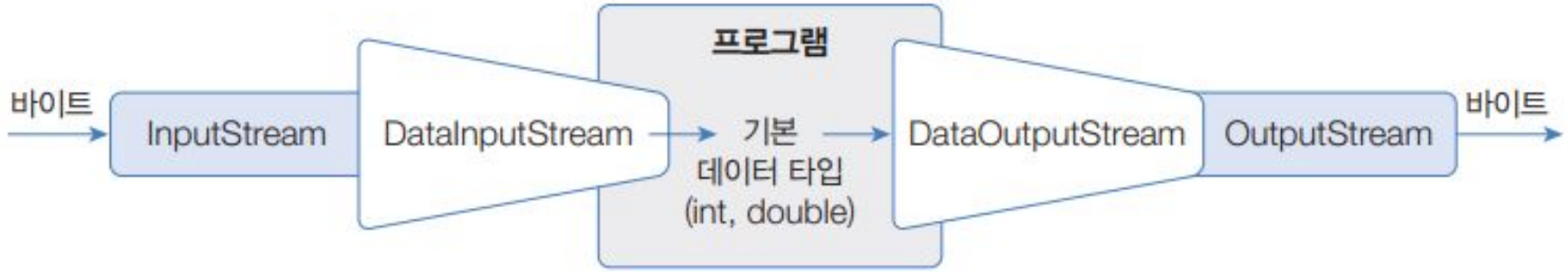
- 텍스트 파일의 행단위 읽기

```
BufferedReader br = new BufferedReader(new FileReader("…"));
while(true) {
  String str = br.readLine();    //파일에서 한 행씩 읽음
  if(str == null) break;         //더 이상 읽을 행이 없을 경우(파일 끝) while 문 종료
}
```

- **ReadLineExample.java**

```
package ch18.sec07.exam02;

import java.io.*;

public class ReadLineExample {
	public static void main(String[] args) throws Exception {
		BufferedReader br = new BufferedReader(
			new FileReader("src/ch18/sec07/exam02/ReadLineExample.java")
		);

		int lineNo = 1;
		while(true) {
			String str = br.readLine();
			if(str == null) break;
			System.out.println(lineNo + "\t" + str);
			lineNo++;
		}

		br.close();
	}
}
```

- 기본 타입 스트림
  - 바이트 스트림에 DataInputStream과 DataOutputStream 보조 스트림을 연결하면 기본 타입(boolean, char, short, int, long, float, double) 값을 입출력할 수 있음




```
DataInputStream dis = new DataInputStream(바이트 입력 스트림);
DataOutputStream dos = new DataOutputStream(바이트 출력 스트림);
```

- 기본 타입 스트림

DataInputStream		DataOutputStream	
boolean	readBoolean( )	void	writeBoolean(boolean v)
byte	readByte( )	void	writeByte(int v)
char	readChar( )	void	writeChar(int v)
double	readDouble( )	void	writeDouble(double v)
float	readFloat( )	void	writeFloat(float v)
int	readInt( )	void	writeInt(int v)
long	readLong( )	void	writeLong(long v)
short	readShort( )	void	writeShort(int v)
String	readUTF( )	void	writeUTF(String str)

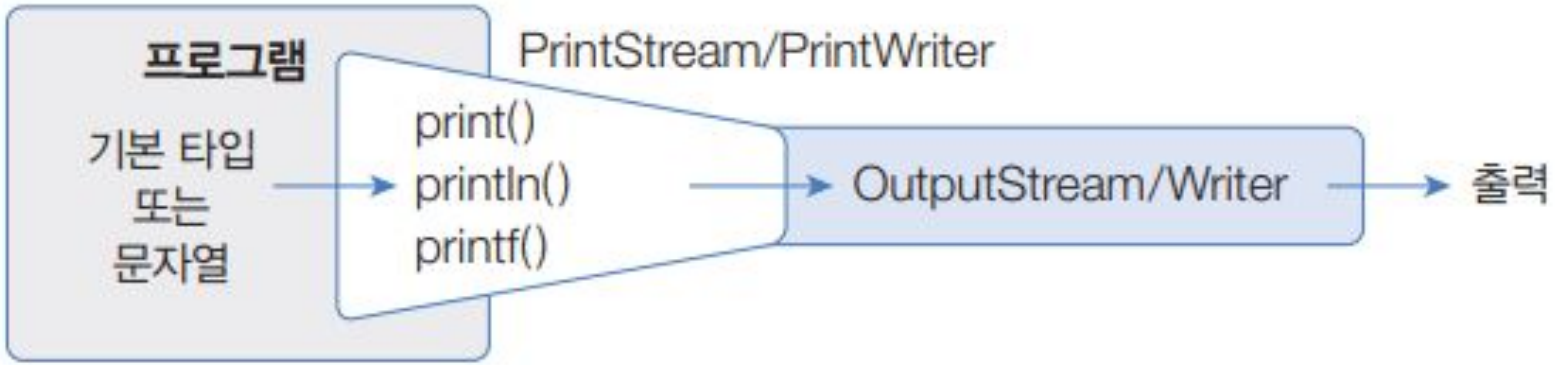
- **DataInputOutputStreamExample.java**

```
package ch18.sec08;

import java.io.*;

public class DataInputOutputStreamExample {
	public static void main(String[] args) throws Exception {
		//DataOutputStream 생성
		FileOutputStream fos = new FileOutputStream("C:/Temp/primitive.db");
		DataOutputStream dos = new DataOutputStream(fos);

		//기본 타입 출력
		dos.writeUTF("홍길동");
		dos.writeDouble(95.5);
		dos.writeInt(1);

		dos.writeUTF("감자바");
		dos.writeDouble(90.3);
		dos.writeInt(2);

		dos.flush(); dos.close(); fos.close();
```

- **DataInputOutputStreamExample.java**

```
        //DataInputStream 생성
        FileInputStream fis = new FileInputStream("C:/Temp/primitive.db");
        DataInputStream dis = new DataInputStream(fis);

        //기본 타입 입력
        for(int i=0; i<2; i++) {
            String name = dis.readUTF();
            double score = dis.readDouble();
            int order = dis.readInt();
            System.out.println(name + " : " + score + " : " + order);
        }

        dis.close(); fis.close();
    }
}
```

- **PrintStream과 PrintWriter**
  - 프린터와 유사하게 출력하는 print(), println(), printf() 메소드를 가진 보조 스트림



  - PrintStream은 바이트 출력 스트림과 연결되고, PrintWriter는 문자 출력 스트림과 연결

```
PrintStream ps = new PrintStream(바이트 출력 스트림);
PrintWriter pw = new PrintWriter(문자 출력 스트림);
```

- **PrintStream과 PrintWriter**
  - 주요 메소드

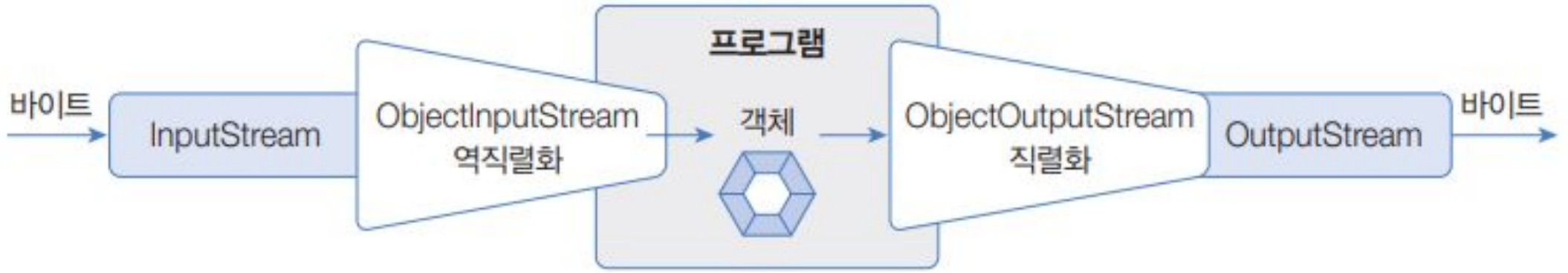
PrintStream / PrintWriter			
void	print(boolean b)	void	println(boolean b)
void	print(char c)	void	println(char c)
void	print(double d)	void	println(double d)
void	print(float f)	void	println(float f)
void	print(int i)	void	println(int i)
void	print(long l)	void	println(long l)
void	print(Object obj)	void	println(Object obj)
void	print(String s)	void	println(String s)
		void	println( )

- **PrintStreamExample.java**

```
package ch18.sec09;

import java.io.FileOutputStream;
import java.io.PrintStream;

public class PrintStreamExample {
	public static void main(String[] args) throws Exception {
		FileOutputStream fos = new FileOutputStream("C:/Temp/printstream.txt");
		PrintStream ps = new PrintStream(fos);

		ps.print("마치 ");
		ps.println("프린터가 출력하는 것처럼 ");
		ps.println("데이터를 출력합니다.");
		ps.printf("| %6d | %-10s | %10s | \n", 1, "홍길동", "도적");
		ps.printf("| %6d | %-10s | %10s | \n", 2, "감자바", "학생");

		ps.flush();
		ps.close();
	}
}
```

```
마치 프린터가 출력하는 것처럼
데이터를 출력합니다.
|      1 | 홍길동       |       도적 |
|      2 | 감자바       |       학생 |
```

- **직렬화와 역직렬화**
  - 직렬화: 메모리에 생성된 객체를 파일 또는 네트워크로 출력하기 위해 필드값을 일렬로 늘어선 바이트로 변경하는 것
  - 역직렬화: 직렬화된 바이트를 객체의 필드값으로 복원하는 것.
  - ObjectOutputStream은 바이트 출력 스트림과 연결되어 객체를 직렬화하고, ObjectInputStream은 바이트 입력 스트림과 연결되어 객체로 복원하는 역직렬화




```
ObjectInputStream ois = new ObjectInputStream(바이트 입력 스트림);
ObjectOutputStream oos = new ObjectOutputStream(바이트 출력 스트림);
```

- **직렬화**
  - ObjectOutputStream의 writeObject() 메소드

    oos.writeObject(객체);

- **역직렬화**
  - ObjectInputStream의 readObject() 메소드

    객체타입 변수 = (객체타입)ois.readObject();

- **Member.java**

```
package ch18.sec10;

import java.io.Serializable;

public class Member implements Serializable {
	private String id;
	private String name;

	public Member(String id, String name) {
		this.id = id;
		this.name = name;
	}

	@Override
	public String toString() { return id + ": " + name; }
}
```

- **Product.java**

```
package ch18.sec10;

import java.io.Serializable;

public class Product implements Serializable {
	private String name;
	private int price;

	public Product(String name, int price) {
		this.name = name;
		this.price = price;
	}

	@Override
	public String toString() { return name + ": " + price; }
}
```

- **ObjectInputOutputStreamExample.java**

```
package ch18.sec10;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Arrays;

public class ObjectInputOutputStreamExample {
	public static void main(String[] args) throws Exception {
		//FileOutputStream에 ObjectOutputStream 보조 스트림 연결
		FileOutputStream fos = new FileOutputStream("C:/Temp/object.dat");
		ObjectOutputStream oos = new ObjectOutputStream(fos);

		//객체 생성
		Member m1 = new Member("fall", "단풍이");
		Product p1 = new Product("노트북", 1500000);
		int[] arr1 = { 1, 2, 3 };
```

- **ObjectInputOutputStreamExample.java**

```
        //객체를 역직렬화해서 파일에 저장
        oos.writeObject(m1);
        oos.writeObject(p1);
        oos.writeObject(arr1);

        oos.flush(); oos.close(); fos.close();

        //FileInputStream에 ObjectInputStream 보조 스트림 연결
        FileInputStream fis = new FileInputStream("C:/Temp/object.dat");
        ObjectInputStream ois = new ObjectInputStream(fis);

        //파일을 읽고 역질렬화해서 객체로 복원
        Member m2 = (Member) ois.readObject();
        Product p2 = (Product) ois.readObject();
        int[] arr2 = (int[]) ois.readObject();

        ois.close(); fis.close();

        //복원된 객체 내용 확인
        System.out.println(m2);
        System.out.println(p2);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
fall: 단풍이
노트북: 1500000
[1, 2, 3]
```

- **Serializable 인터페이스**
  - 멤버가 없는 빈 인터페이스이지만, 객체를 직렬화할 수 있다고 표시하는 역할
  - 인스턴스 필드값은 직렬화 대상.
  - 정적 필드값과 transient로 선언된 필드값은 직렬화에서 제외되므로 출력되지 않음

```
public class XXX implements Serializable {
  public int field1;
  protected int field2;
  int field3;
  private int field4;
  public static int field5;    //정적 필드는 직렬화에서 제외
  transient int field6;        //transient로 선언된 필드는 직렬화에서 제외
}
```

- **serialVersionUID 필드**
  - 직렬화할 때 사용된 클래스와 역직렬화할 때 사용된 클래스는 동일한 클래스여야 함
  - 클래스 내용이 다르더라도 두 클래스가 동일한 serialVersionUID 상수값을 가지면 역직렬화 가능

```
public class Member implements
  Serializable {
  static final long
    serialVersionUID = 1;
  int field1;
  int field2;
}
```

```
public class Member implements
  Serializable {
  static final long
    serialVersionUID = 1;
  int field1;
  int field2;
  int field3;
}
```

- **File 클래스**
  - File 클래스로부터 File 객체를 생성

```
File file = new File("경로");
```

```
File file = new File("C:/Temp/file.txt");
File file = new File("C:\\Temp\\file.txt");
```

  - exists() 메소드가 false를 리턴할 경우, 다음 메소드로 파일 또는 폴더를 생성

리턴 타입	메소드	설명
boolean	createNewFile( )	새로운 파일을 생성
booelan	mkdir( )	새로운 디렉토리를 생성
boolean	mkdirs( )	경로상에 없는 모든 디렉토리를 생성

- **File 클래스**
  - exists() 메소드의 리턴값이 true라면 다음 메소드를 사용

리턴 타입	메소드	설명
boolean	delete( )	파일 또는 디렉토리 삭제
boolean	canExecute( )	실행할 수 있는 파일인지 여부
boolean	canRead( )	읽을 수 있는 파일인지 여부
boolean	canWrite( )	수정 및 저장할 수 있는 파일인지 여부
String	getName( )	파일의 이름을 리턴
String	getParent( )	부모 디렉토리를 리턴
File	getParentFile( )	부모 디렉토리를 File 객체로 생성 후 리턴
String	getPath( )	전체 경로를 리턴
boolean	isDirectory( )	디렉토리인지 여부
boolean	isFile( )	파일인지 여부
boolean	isHidden( )	숨김 파일인지 여부
long	lastModified( )	마지막 수정 날짜 및 시간을 리턴

- **File 클래스**
  - exists() 메소드의 리턴값이 true라면 다음 메소드를 사용

리턴 타입	메소드	설명
long	length( )	파일의 크기 리턴
String[]	list( )	디렉토리에 포함된 파일 및 서브 디렉토리 목록 전부를 String 배열로 리턴
String[]	list(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브 디렉토리 목록 중에 FilenameFilter에 맞는 것만 String 배열로 리턴
File[]	listFiles( )	디렉토리에 포함된 파일 및 서브 디렉토리 목록 전부를 File 배열로 리턴
File[]	listFiles(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브 디렉토리 목록 중에 FilenameFilter에 맞는 것만 File 배열로 리턴

- **FileExample.java**

```
package ch18.sec11;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;

public class FileExample {
	public static void main(String[] args) throws Exception {
		//File 객체 생성
		File dir = new File("C:/Temp/images");
		File file1 = new File("C:/Temp/file1.txt");
		File file2 = new File("C:/Temp/file2.txt");
		File file3 = new File("C:/Temp/file3.txt");

		//존재하지 않으면 디렉토리 또는 파일 생성
		if(dir.exists() == false) { dir.mkdirs(); }
		if(file1.exists() == false) { file1.createNewFile(); }
		if(file2.exists() == false) { file2.createNewFile(); }
		if(file3.exists() == false) { file3.createNewFile(); }
```

- **FileExample.java**

```
        //Temp 폴더의 내용을 출력
        File temp = new File("C:/Temp");
        File[] contents = temp.listFiles();

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd a HH:mm");
        for(File file : contents) {
            System.out.printf("%-25s", sdf.format(new Date(file.lastModified())));
            if(file.isDirectory()) {
                System.out.printf("%-10s%-20s", "<DIR>", file.getName());
            } else {
                System.out.printf("%-10s%-20s", file.length(), file.getName());
            }
            System.out.println();
        }
    }
}
```

```
2023-05-17 오후 16:15      <DIR>      2020
2024-01-19 오후 16:52      3683       chapter06.sql
2024-01-22 오전 11:08      0          file1.txt
2024-01-22 오전 11:08      0          file2.txt
2024-01-22 오전 11:08      0          file3.txt
2024-01-10 오후 13:21      415        Hello.class
2024-01-10 오후 13:18      108        Hello.java
2024-01-22 오전 11:08      <DIR>      images
2024-01-22 오전 11:04      201        object.dat
```

- **Files 클래스**
  - Files 클래스는 정적 메소드로 구성되어 있기 때문에 File 클래스처럼 객체로 만들 필요 없음
  - Files의 정적 메소드는 운영체제의 파일 시스템에게 파일 작업을 수행하도록 위임

기능	관련 메소드
복사	copy
생성	createDirectories, createDirectory, createFile, createLink, createSymbolicLink, createTempDirectory, createTempFile
이동	move
삭제	delete, deleteIfExists
존재, 검색, 비교	exists, notExists, find, mismatch
속성	getLastModifiedTime, getOwner, getPosixFilePermissions, isDirectory, isExecutable, isHidden, isReadable, isSymbolicLink, isWritable, size, setAttribute, setLastModifiedTime, setOwner, setPosixFilePermissions, probeContentType

- **Files 클래스**

기능	관련 메소드
속성	getLastModifiedTime, getOwner, getPosixFilePermissions, isDirectory, isExecutable, isHidden, isReadable, isSymbolicLink, isWritable, size, setAttribute, setLastModifiedTime, setOwner, setPosixFilePermissions, probeContentType
디렉토리 탐색	list, newDirectoryStream, walk
데이터 입출력	newInputStream, newOutputStream, newBufferedReader, newBufferedWriter, readAllBytes, lines, readAllLines, readString, readSymbolicLink, write, writeString

- **FilesExample.java**

```
package ch18.sec11;

import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FilesExample {
	public static void main(String[] args) {
		try {
			String data = "" +
					"id: winter\n" +
					"email: winter@mycompany.com\n" +
					"tel: 010-123-1234";

			//Path 객체 생성
			Path path = Paths.get("C:/Temp/user.txt");

			//파일 생성 및 데이터 저장
			Files.writeString(Paths.get("C:/Temp/user.txt"), data, Charset.forName("UTF-8"));
```

- **FilesExample.java**

```
            //파일 정보 얻기
            System.out.println("파일 유형: " + Files.probeContentType(path));
            System.out.println("파일 크기: " + Files.size(path) + " bytes");

            //파일 읽기
            String content = Files.readString(path, Charset.forName("UTF-8"));
            System.out.println(content);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
파일 유형: text/plain
파일 크기: 56 bytes
id: winter
email: winter@mycompany.com
tel: 010-123-1234
```