

제14장

# 자바 입출력

구디아카데미 ▷ 민경태 강사

```
each: function(e, t, n) {  
    n, i = 0,  
    o = e.length,  
    a = M(e);  
    if (n) {  
        if (a) {  
            for (; o > i; i++)  
                if (r = t.apply(e[i], n), r ===  
            ) else  
                for (i in e)  
                    if (r = t.apply(e[i], n), r ===  
        ) else if (a) {  
            for (; o > i; i++)  
                if (r = t.call(e[i], i, e[i])  
            ) else  
                for (i in e)  
                    if (r = t.call(e[i], i, e[i])  
        ) return e  
    },  
    trim: b && !b.call("\uffff\u00a0") ?  
        return null == e ? "" : b.call(  
    ) : function(e) {  
        return null == e ? "" : (e +  
    ),  
    makeArray: function(e, t) {  
        var n = t || [];  
        return null != e && (M(Obj  
    ),  
    isArray: function(e, t, n) {  
        var r;  
        if (t) {  
            if (n) return m.c  
            for (n = t.length  
                if (n in t  
        )  
    }
```

# 학습목표

1. File 클래스에 대해서 알 수 있다.
2. 입출력 스트림의 개념에 대해서 이해할 수 있다.
3. 바이트 입출력 스트림에 대해서 알 수 있다.
4. 문자 입출력 스트림에 대해서 알 수 있다.

```
each: function(e, t, n) {  
    n, i = 0,  
    o = e.length,  
    a = M(e);  
    if (n) {  
        if (a) {  
            for (; o > i; i++)  
                if (r = t.apply(e[i], n), r ===  
        } else  
            for (i in e)  
                if (r = t.apply(e[i], n), r ===  
    } else if (a) {  
        for (; o > i; i++)  
            if (r = t.call(e[i], i, e[i])  
    } else  
        for (i in e)  
            if (r = t.call(e[i], i, e[i])  
    return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
} : function(e) {  
    return null == e ? "" : (e + "  
},  
makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
}),  
isArray: function(e, t, n) {  
    var r;  
    if (t) {  
        if (n) return m.c  
        for (n = t.length  
            if (n in t  
    }  
}
```

# 목차

1. File 클래스
2. 입출력 스트림의 개념
3. 바이트 출력 스트림
  - 1) OutputStream
  - 2) FileOutputStream
  - 3) BufferedOutputStream
4. 바이트 입력 스트림
  - 1) InputStream
  - 2) FileInputStream
  - 3) BufferedInputStream

```
each: function(e, t, n) {  
    n, i = 0,  
    o = e.length,  
    a = M(e);  
    if (n) {  
        if (a) {  
            for (; o > i; i++)  
                if (r = t.apply(e[i], n), r ===  
        } else  
            for (i in e)  
                if (r = t.apply(e[i], n), r ===  
    } else if (a) {  
        for (; o > i; i++)  
            if (r = t.call(e[i], i, e[i])  
    } else  
        for (i in e)  
            if (r = t.call(e[i], i, e[i])  
    return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
} : function(e) {  
    return null == e ? "" : (e + "  
},  
makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
}),  
isArray: function(e, t, n) {  
    var r;  
    if (t) {  
        if (n) return n.c  
        for (n = t.length  
            if (n in t  
    }  
}
```

# 목차

## 5. 문자 출력 스트림

- 1) Writer
- 2) FileWriter
- 3) BufferedWriter
- 4) PrintWriter

## 6. 문자 입력 스트림

- 1) Reader
- 2) FileReader
- 3) BufferedReader
- 4) InputStreamReader

```

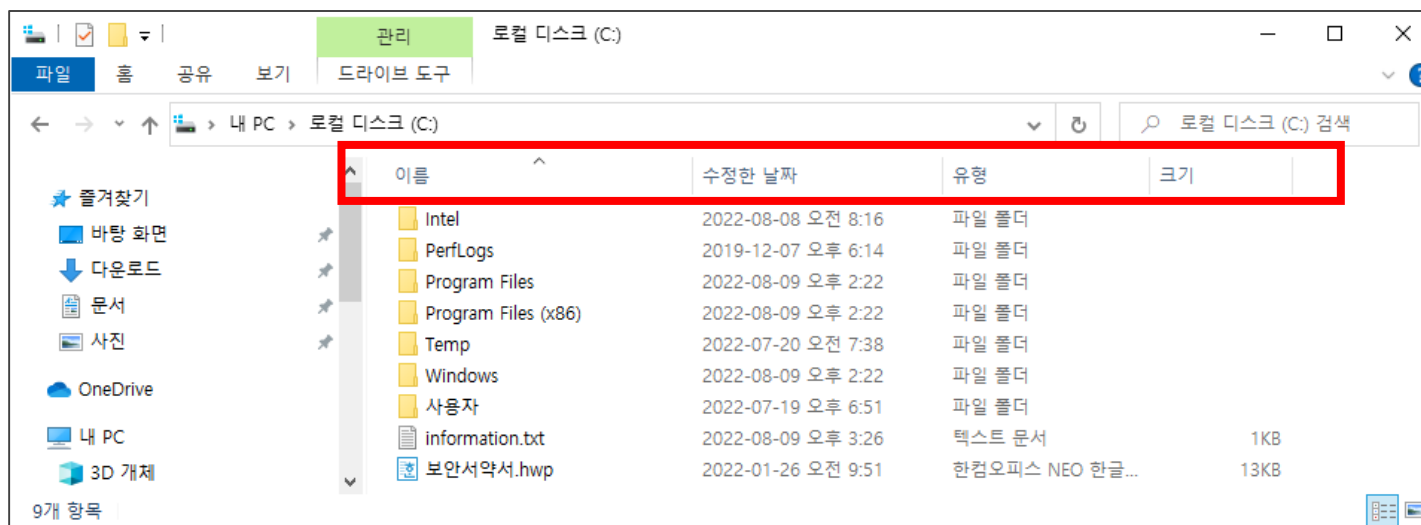
each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        a = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
    } else if (a) {
        for (; o > i; i++)
            if (r = t.call(e[i], e[i]), r === !1) break;
    } else
        for (i in e)
            if (r = t.call(e[i], e[i]), r === !1) break;
    return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e);
} : function(e) {
    return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string"
    ),
inArray: function(e, t, n) {
    var r;
    if (t) {
        if (m) return m.call(t, e, n);
        for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r in t && t[r] === e) return r;
    }
}

```

# 1. File 클래스

# File 클래스

- 파일과 디렉터리를 관리할 수 있는 클래스
- 패키지 : java.io
- 파일과 디렉터리의 정보 확인이 가능함
  - 파일명
  - 경로
  - 수정한 날짜
  - 파일 크기 등



# File 클래스

- File 클래스를 이용해 파일과 디렉터리를 조작할 수 있음
  - 파일/디렉터리 생성
  - 파일/디렉터리 삭제
- 파일 조작
  - 파일을 생성하는 경우 크기가 0인 파일을 생성하게 됨
  - 파일의 내용까지 작성할 수는 없으므로 파일의 내용 작성을 위해서는 출력 스트림을 함께 사용해야 함

# File 클래스 - 생성자

## ■ File 클래스 주요 생성자

생성자	의미
File(File parent, String child)	디렉터리 정보를 가지고 있는 File parent와 파일 String child를 이용해서 File 객체 생성
File(String pathname)	파일의 전체 경로를 가지고 있는 String pathname을 이용해서 File 객체 생성
File(String parent, String child)	디렉터리 String parent와 파일 String child를 이용해서 File 객체 생성



# File 클래스 - 메소드

## ■ File 클래스 주요 메소드

메소드	역할
<code>boolean isDirectory()</code>	File 객체가 디렉터리이면 true 아니면 false 반환
<code>boolean isFile()</code>	File 객체가 파일이면 true 아니면 false 반환
<code>String getName()</code>	파일의 이름 반환
<code>String getParent()</code>	파일이 저장된 디렉터리의 이름 반환
<code>String getPath()</code>	디렉터리의 이름과 파일의 이름을 합친 전체 경로를 반환
<code>String[] list()</code>	경로 내의 디렉터리와 파일 이름을 합친 전체 경로를 가진 <code>String[]</code> 배열을 반환
<code>File[] listFiles()</code>	경로 내의 디렉터리와 파일을 File 객체로 변환한 뒤 <code>File[]</code> 배열에 저장해서 반환
<code>boolean mkdirs()</code>	모든 하위 디렉터리 생성. 성공하면 true 실패하면 false 반환
<code>boolean createNewFile()</code>	파일 크기가 0인 새로운 파일을 생성. 성공하면 true 실패하면 false 반환
<code>boolean delete()</code>	곧바로 디렉터리나 파일을 삭제. 성공하면 true 실패하면 false 반환
<code>boolean deleteOnExit()</code>	JVM이 종료되면 디렉터리나 파일을 삭제. 성공하면 true 실패하면 false 반환
<code>boolean exists()</code>	디렉터리나 파일이 존재하면 true 아니면 false 반환

# File 클래스 - 디렉터리 조작

## ■ 디렉터리 생성

```
File dir = new File("testDir");  
if(!dir.exists()) {  
    dir.mkdirs();  
}
```

## ■ 디렉터리 삭제(비어 있는 디렉터리만 삭제 가능함)

```
File dir = new File("testDir");  
if(dir.exists()) {  
    dir.delete();  
}
```

# File 클래스 - 파일 조작 및 확인

## ■ 파일 생성

```
// testDir 디렉터리 생성
File dir = new File("testDir");
if(!dir.exists()) {
    dir.mkdirs();
}

// testDir 디렉터리에 test.txt 파일 생성
File file = new File(dir, "test.txt");
try {
    file.createNewFile();
} catch(IOException e) {
    e.printStackTrace();
}
```

## ■ 파일 정보 확인

```
System.out.println(file.isFile());
System.out.println(file.isDirectory());
System.out.println(file.length());
System.out.println(file.getParent());
System.out.println(file.getName());
System.out.println(file.getPath());
```

### 실행결과

```
true (파일인가?)
false (디렉터리인가?)
0 (파일 크기)
"testDir" (디렉터리명)
"test.txt" (파일명)
"textDir\test.txt" (전체경로)
```

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r in t && t[r] === e) return r;
  }
}

```

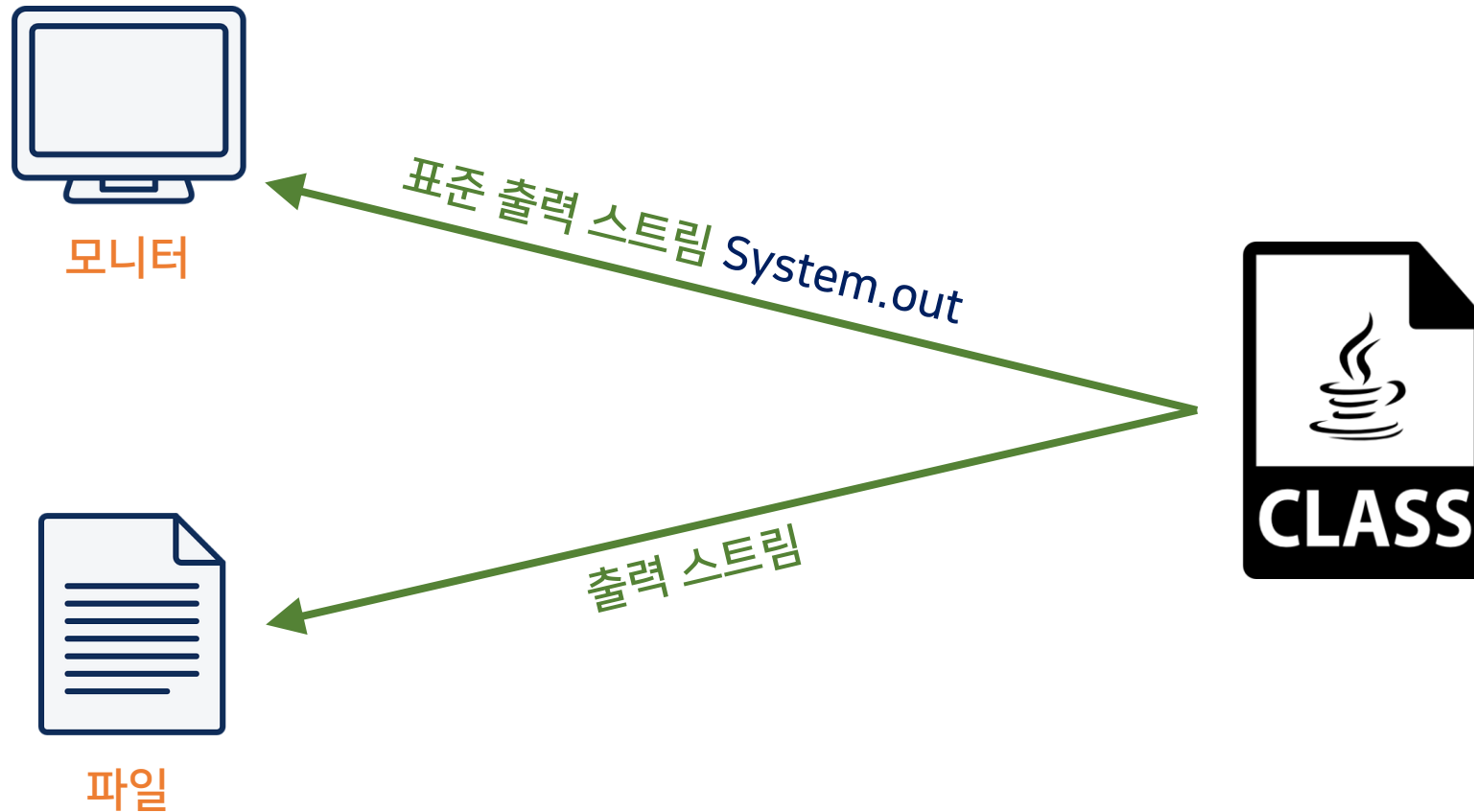
## 2. 입출력 스트림의 개념

# 입출력 스트림의 개념

- IO Stream
  - 입출력이 이루어지는 연결 통로
- 자바 프로그램은 키보드, 모니터, 파일, 네트워크 등과 입출력 스트림을 통해서 데이터를 주고 받을 수 있음
- 입력만 수행하는 입력 스트림과 출력만 수행하는 출력 스트림으로 구분해서 사용함

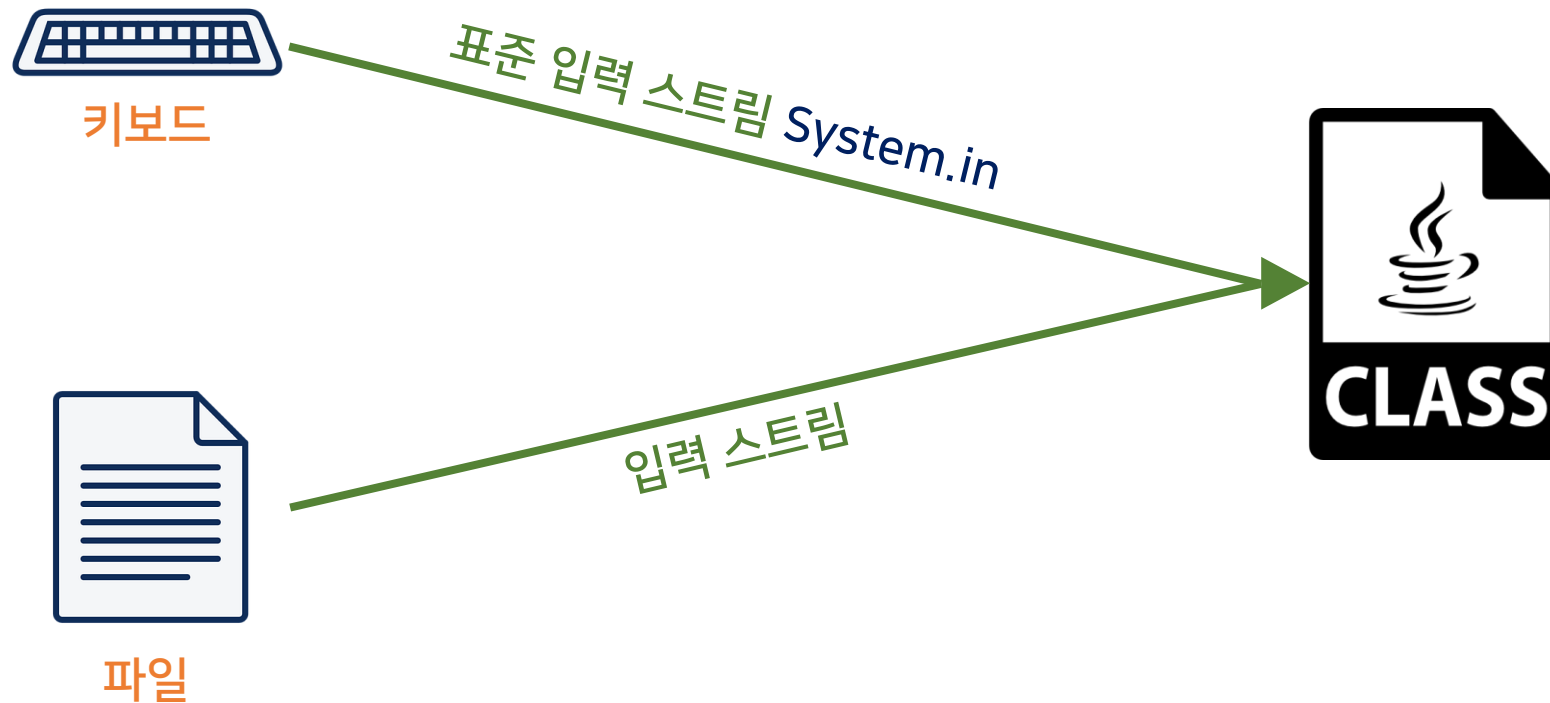
# 출력 스트림

- 모니터, 파일 등으로 정보를 내보내는 스트림



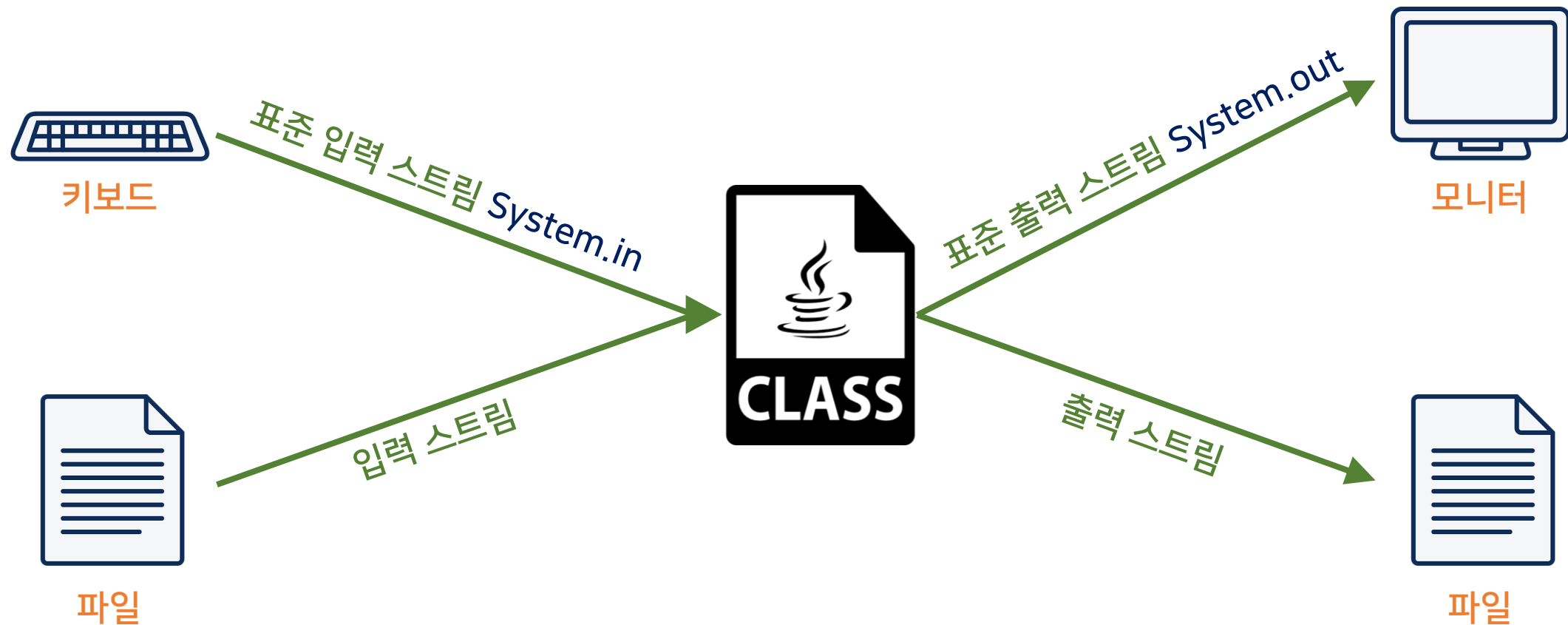
# 입력 스트림

- 키보드, 파일 등의 정보를 읽어 들이는 스트림



# 입출력 스트림

- 데이터 전송을 위한 입출력 스트림의 흐름





# java.io 패키지

- 파일 처리 및 입출력 스트림에 관련된 클래스를 제공하는 패키지
- 파일 시스템을 관리하는 클래스와 입출력 스트림을 이용하는 클래스를 제공함
  - File 클래스
  - OutputStream 클래스
  - InputStream 클래스
  - Writer 클래스
  - Reader 클래스

# 바이트 기반 스트림

- 바이트(byte) 단위로 데이터를 주고 받는 스트림
- 그림, 영상 등 바이너리 데이터(binary data, 2진 데이터)를 주고 받을 수 있음
- 자바의 바이트 출력 스트림
  - OutputStream 클래스와 이를 상속 받는 모든 클래스
  - 모든 클래스 이름이 OutputStream으로 끝남
- 자바의 바이트 입력 스트림
  - InputStream 클래스와 이를 상속 받는 모든 클래스
  - 모든 클래스 이름이 InputStream으로 끝남

# 문자 기반 스트림

- 문자(character) 단위로 데이터를 주고 받는 스트림
- 한글을 포함한 모든 문자 데이터를 주고 받을 수 있음
- 자바의 문자 출력 스트림
  - Writer 클래스와 이를 상속 받는 모든 클래스
  - 모든 클래스 이름이 Writer로 끝남
- 자바의 문자 입력 스트림
  - Reader 클래스와 이를 상속 받는 모든 클래스
  - 모든 클래스 이름이 Reader로 끝남

```

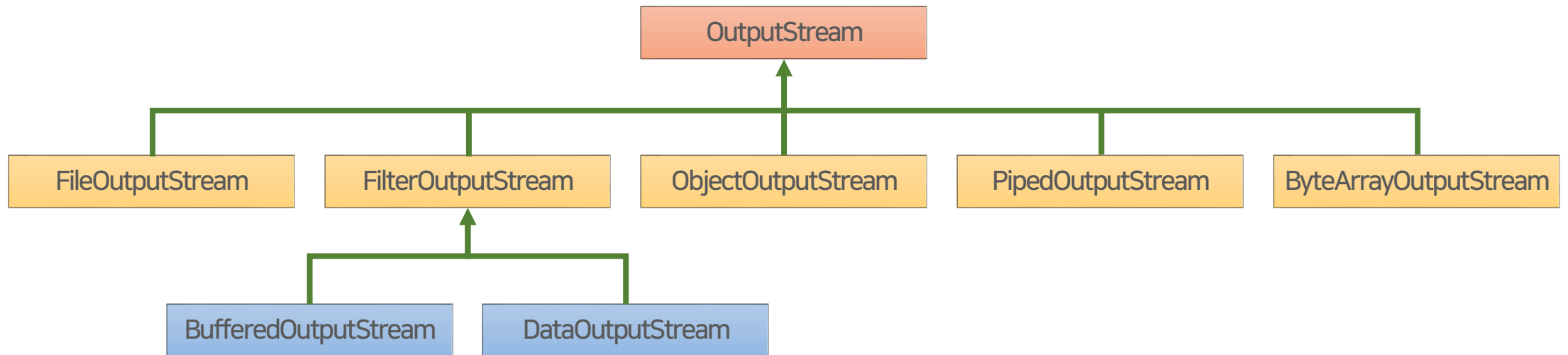
each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffeff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; n in t && t[n] === e) return n;
  }
}

```

### 3. 바이트 출력 스트림

# OutputStream

- 모든 바이트 출력 스트림의 최상위 슈퍼 클래스
- 주요 OutputStream 구조도



# OutputStream 클래스 - 메소드

- 모든 바이트 출력 스트림은 OutputStream 클래스의 메소드를 사용할 수 있음
- OutputStream 클래스의 주요 메소드

메소드	역할
<code>void close()</code> <code>throws IOException</code>	출력 스트림을 닫고 이 스트림과 관련된 모든 시스템 리소스 해제
<code>void flush()</code> <code>throws IOException</code>	출력 스트림을 플러시(flush)하고 버퍼링된 출력 바이트를 강제로 기록
<code>abstract void write(int b)</code> <code>throws IOException</code>	전달된 int b의 데이터를 현재 출력 스트림에 쓰기
<code>void write(byte[] b)</code> <code>throws IOException</code>	전달된 byte[] b의 전체 내용을 현재 출력 스트림에 쓰기
<code>void write(byte[] b, int off, int len)</code> <code>throws IOException</code>	전달된 byte[] b의 인덱스 off부터 len개 요소를 현재 출력 스트림에 쓰기

# FileOutputStream 클래스

- 이미지, 동영상 등과 같은 바이너리 파일을 생성하는 바이트 출력 스트림
- 텍스트 파일의 생성은 FileOutputStream 클래스 대신 FileWriter 클래스 사용을 권장함
- 2가지 파일 생성 모드가 있음
  - 생성 모드 : 항상 새로운 파일을 만듦
  - 추가 모드 : 기존에 지정된 파일이 있으면 해당 파일의 끝에 추가함

# FileOutputStream 클래스 - 생성자

## ■ FileOutputStream 클래스 주요 생성자

생성자	역할
<code>FileOutputStream(File file)</code> <code>throws FileNotFoundException</code>	전달된 File 객체가 나타내는 파일에 쓰기 위한 파일 출력 스트림 생성
<code>FileOutputStream(File file, boolean append)</code> <code>throws FileNotFoundException</code>	전달된 File 객체가 나타내는 파일에 쓰기 위한 파일 출력 스트림 생성 boolean append 값이 true이면 추가 모드로 파일 출력 스트림 생성
<code>FileOutputStream(String name)</code> <code>throws FileNotFoundException</code>	이름이 name인 파일에 쓰기 위한 파일 출력 스트림 생성
<code>FileOutputStream(String name, boolean append)</code> <code>throws FileNotFoundException</code>	이름이 name인 파일에 쓰기 위한 파일 출력 스트림 생성 boolean append 값이 true이면 추가 모드로 파일 출력 스트림 생성



# FileOutputStream 클래스 - test1.dat 생성

## ■ test1.dat 파일을 새로 만들기

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileOutputStream out = new FileOutputStream("test1.dat");  
            String str = "hello";  
            byte[] b = str.getBytes();  
            out.write(b);  
            out.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과



test1.dat  
DAT 파일 (크기 5바이트)  
5바이트

# FileOutputStream 클래스 - test1.dat 추가

## ■ test1.dat 파일에 데이터 추가하기

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileOutputStream out = new FileOutputStream("test1.dat", true);  
            String str = "world";  
            byte[] b = str.getBytes();  
            out.write(b);  
            out.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 데이터 추가 결과



test1.dat  
DAT 파일 (크기가 10바이트로 증가)  
10바이트

# 버퍼 입출력

## ■ 버퍼 입출력의 동작 원리

- 입출력 횟수를 줄여서 입출력 성능(속도)을 개선할 수 있도록 버퍼를 사용함



# BufferedOutputStream 클래스

- 버퍼(Buffer)를 제공하는 보조 출력 스트림
  - 단독 사용 불가
  - 다른 OutputStream 클래스와 함께 사용해야 함
- 출력 데이터를 저장할 수 있는 내부 버퍼를 가지고 있음
  - `protected byte[] buf` 필드를 가지고 있음
- BufferedOutputStream 클래스를 이용하면 출력 속도를 향상시킬 수 있음

# BufferedOutputStream 클래스 - 생성자

## ■ BufferedOutputStream 클래스 생성자


생성자	역할
BufferedOutputStream(OutputStream out)	전달된 기본 출력 스트림 out에 데이터를 쓰기 위해서 기본 버퍼 크기의 버퍼링된 새 출력 스트림을 생성함
BufferedOutputStream(OutputStream out, int size)	지정된 버퍼 크기 size를 사용하여 전달된 기본 출력 스트림 out에 데이터를 쓰기 위해 버퍼링된 새 출력 스트림을 생성함

# BufferedOutputStream 클래스 - test2.dat 생성

## ■ test2.dat 파일을 새로 만들기

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream("test2.dat"));  
            String str = "hello";  
            byte[] b = str.getBytes();  
            out.write(b);  
            out.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과

 test2.dat  
DAT 파일 (크기 5바이트)  
5바이트

# BufferedOutputStream 클래스 - test2.dat 추가

## ■ test2.dat 파일에 데이터 추가하기

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream("test2.dat", true));  
            String str = "world";  
            byte[] b = str.getBytes();  
            out.write(b);  
            out.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

데이터 추가 결과



test2.dat  
DAT 파일 (크기가 10바이트로 증가)  
10바이트

# 버퍼링 유무에 따른 속도 차이 확인

- 버퍼링의 성능 확인을 위한 시나리오
  - 버퍼를 사용하지 않는 스트림으로 파일 생성
    - FileOutputStream 클래스를 이용해서 test3.dat 파일 생성
  - 버퍼를 사용하는 스트림으로 파일 생성
    - BufferedOutputStream 클래스를 이용해서 test4.dat 파일 생성
  - test3.dat 파일과 test4.dat 파일에 모두 "hello"라는 데이터를 기록함
  - "hello"를 파일에 기록하는 시간을 기록해서 차이를 비교함



# FileOutputStream 속도 확인

- test3.dat 파일 생성에 걸리는 시간 측정

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileOutputStream out = new FileOutputStream("test3.dat");  
            String str = "hello";  
            byte[] b = str.getBytes();  
            long startTime = System.nanoTime();  
            out.write(b); // 쓰기 작업의 시간을 측정  
            System.out.println(System.nanoTime() - startTime);  
            out.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과 (실행 결과는 다를 수 있다.)

76500

# BufferedOutputStream 속도 확인

- test4.dat 파일 생성에 걸리는 시간 측정

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream("test4.dat"));  
            String str = "hello";  
            byte[] b = str.getBytes();  
            long startTime = System.nanoTime();  
            out.write(b); // 쓰기 작업의 시간을 측정  
            System.out.println(System.nanoTime() - startTime);  
            out.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과 (실행 결과는 다를 수 있다.)

16800

FileOutputStream 보다 약 4.5배 빠르게 측정되었다.

```

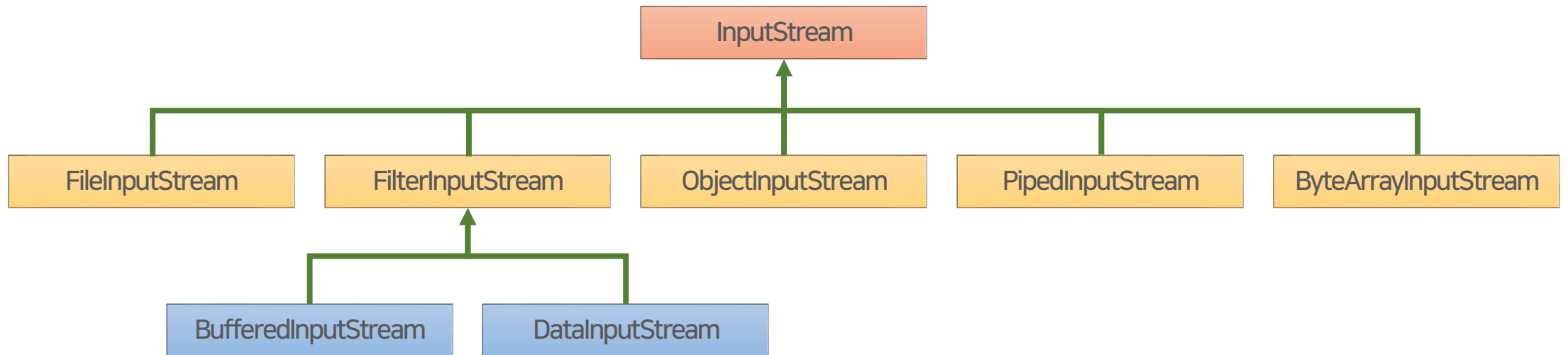
each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n : 0; r in t && t[r] === e) return r;
  }
}

```

## 4. 바이트 입력 스트림

# InputStream

- 모든 바이트 입력 스트림의 최상위 슈퍼 클래스
- 주요 InputStream 구조도



# InputStream 클래스 - 메소드

- 모든 바이트 입력 스트림은 InputStream 클래스의 메소드를 사용할 수 있음
- InputStream 클래스의 주요 메소드

메소드	역할
<code>void close()</code> <code>throws IOException</code>	입력 스트림을 닫고 이 스트림과 관련된 모든 시스템 리소스 해제
<code>abstract int read()</code> <code>throws IOException</code>	입력 스트림에서 데이터의 다음 바이트 읽어서 반환 읽은 데이터가 없으면 -1 반환
<code>int read(byte[] b)</code> <code>throws IOException</code>	입력 스트림에서 일부 바이트를 읽고 byte[] b 배열에 저장함 입력 스트림에서 읽은 바이트 수를 반환, 읽은 데이터가 없으면 -1 반환
<code>int read(byte[] b, int off, int len)</code> <code>throws IOException</code>	입력 스트림에서 최대 len 바이트를 읽고 byte[] b 배열의 인덱스 off부터 저장함 입력 스트림에서 읽은 바이트 수를 반환, 읽은 데이터가 없으면 -1 반환
<code>long transferTo(OutputStream out)</code> <code>throws IOException</code>	입력 스트림에서 모든 바이트를 읽고 이를 지정된 출력 스트림에 쓰기 입력 스트림에서 출력 스트림으로 보낸 바이트 수를 반환

# FileInputStream 클래스

- 이미지, 동영상 등과 같은 바이너리 파일을 읽는 바이트 입력 스트림
- 텍스트 파일의 읽기는 FileInputStream 클래스 대신 FileReader 클래스 사용을 권장함

# FileInputStream 클래스 - 생성자

## ■ FileInputStream 클래스 주요 생성자

생성자	역할
<code>FileInputStream(File file)</code> <code>throws FileNotFoundException</code>	전달된 File 객체가 나타내는 파일을 읽기 위한 파일 입력 스트림 생성
<code>FileInputStream(String name)</code> <code>throws FileNotFoundException</code>	이름이 name인 파일을 읽기 위한 파일 입력 스트림 생성

# FileInputStream 클래스 - test1.dat 읽기1

## ■ public int read() 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileInputStream in = new FileInputStream("test1.dat");  
            int c = 0;  
            byte[] result = new byte[10]; // test1.dat 파일이 10바이트라는 것을 알고 있어서 설정한 크기  
            int i = 0;  
            while((c = in.read()) != -1) {  
                result[i++] = (byte)c;  
            }  
            System.out.println(new String(result));  
            in.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

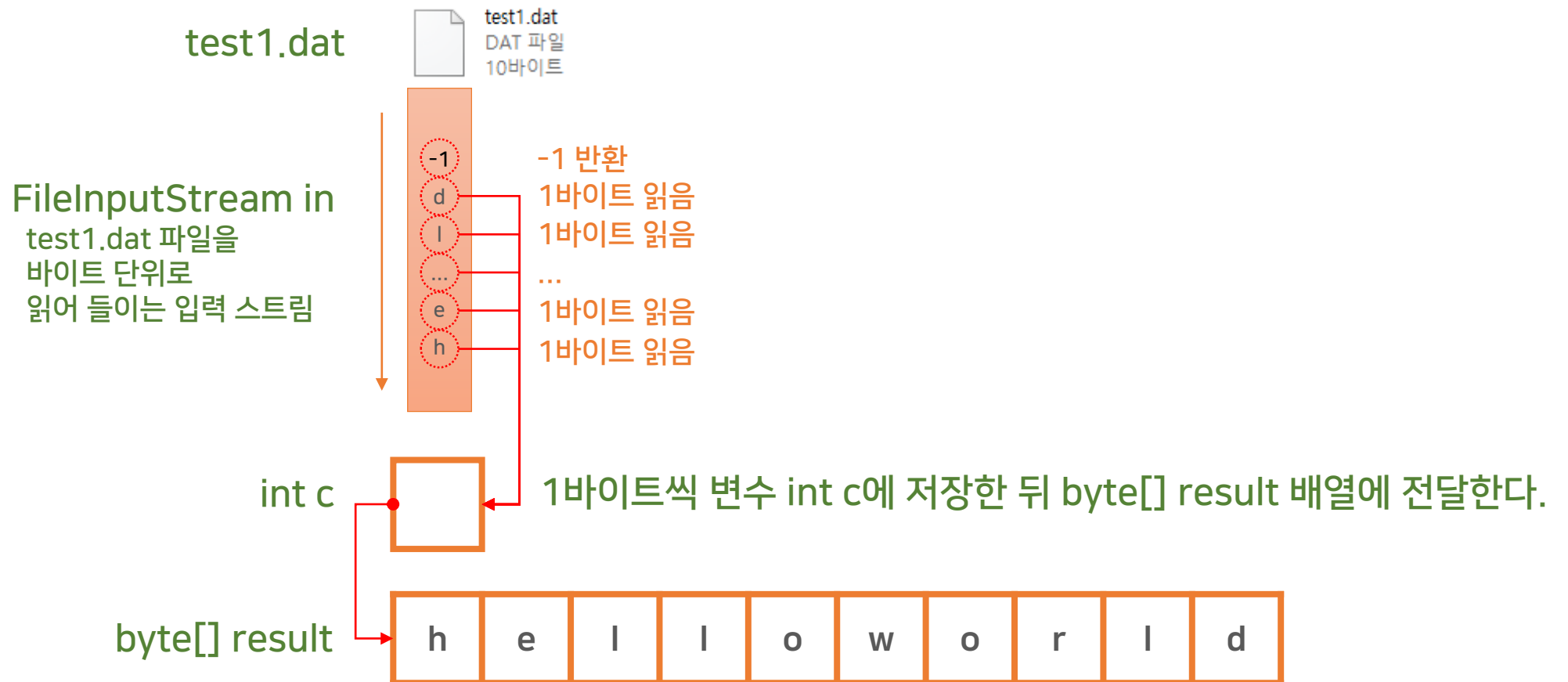
실행결과

"helloworld"



# FileInputStream 클래스 - test1.dat 읽기1

## ■ 코드 해석



# FileInputStream 클래스 - test1.dat 읽기2

## ■ public int read(byte[] b) 메소드

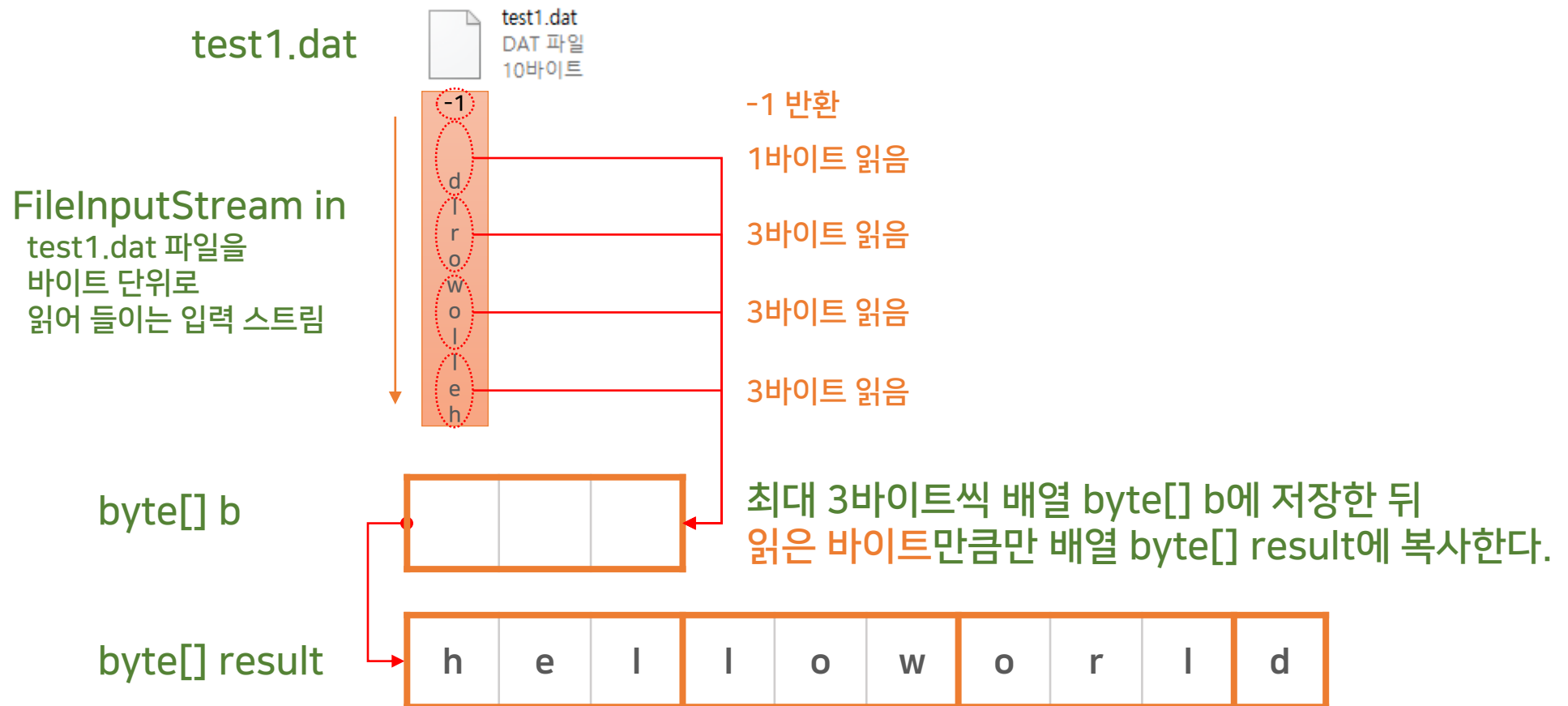
```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileInputStream in = new FileInputStream("test1.dat");  
            byte[] b = new byte[3];  
            byte[] result = new byte[10]; // test1.dat 파일이 10바이트라는 것을 알고 있어서 설정한 크기  
            int i = 0;  
            int readByte = 0;  
            while((readByte = in.read(b)) != -1) {  
                System.arraycopy(b, 0, result, i, readByte);  
                i += readByte;  
            }  
            System.out.println(new String(result));  
            in.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"helloworld"

# FileInputStream 클래스 - test1.dat 읽기2

## ■ 코드 해석



# BufferedInputStream 클래스

- 버퍼(Buffer)를 제공하는 보조 입력 스트림
  - 단독 사용 불가
  - 다른 InputStream 클래스와 함께 사용해야 함
- 입력 데이터를 저장할 수 있는 내부 버퍼를 가지고 있음
  - protected byte[] buf 필드를 가지고 있음
- BufferedInputStream 클래스를 이용하면 입력 속도를 향상시킬 수 있음

# BufferedInputStream 클래스 - test2.dat 읽기1

## ■ public int read() 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedInputStream in = new BufferedInputStream(new FileInputStream("test2.dat"));  
            int c = 0;  
            byte[] result = new byte[10]; // test2.dat 파일이 10바이트라는 것을 알고 있어서 설정한 크기  
            int i = 0;  
            while((c = in.read()) != -1) {  
                result[i++] = (byte)c;  
            }  
            System.out.println(new String(result));  
            in.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"helloworld"

# BufferedInputStream 클래스 - test2.dat 읽기2

## ■ public int read(byte[] b) 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedInputStream in = new BufferedInputStream(new FileInputStream("test2.dat"));  
            byte[] b = new byte[3];  
            byte[] result = new byte[10]; // test2.dat 파일이 10바이트라는 것을 알고 있어서 설정한 크기  
            int i = 0;  
            int readByte = 0;  
            while((readByte = in.read(b)) != -1) {  
                System.arraycopy(b, 0, result, i, readByte);  
                i += readByte;  
            }  
            System.out.println(new String(result));  
            in.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"helloworld"

```

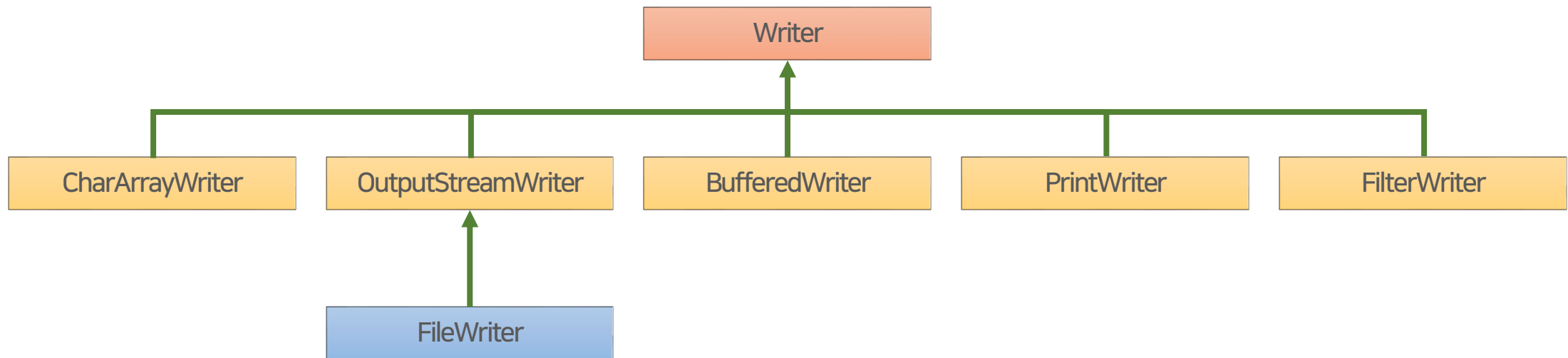
each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; r in t && t[r] === e) return r;
  }
}

```

## 5. 문자 출력 스트림

# Writer

- 모든 문자 출력 스트림의 최상위 슈퍼 클래스
- 주요 Writer 구조도





# Writer 클래스 - 메소드

- 모든 문자 출력 스트림은 Writer 클래스의 메소드를 사용할 수 있음
- Writer 클래스의 주요 메소드

메소드	역할
<code>void close()</code> <code>throws IOException</code>	출력 스트림을 닫고 플러시(flush)함
<code>void flush()</code> <code>throws IOException</code>	출력 스트림을 플러시(flush)함
<code>void write(int c)</code> <code>throws IOException</code>	int c로 전달된 한 글자를 현재 출력 스트림에 쓰기
<code>void write(char[] cbuf)</code> <code>throws IOException</code>	char[] buf로 전달된 전체 요소를 현재 출력 스트림에 쓰기
<code>abstract void write(char[] cbuf, int off, int len)</code> <code>throws IOException</code>	char[] buf의 인덱스 off부터 len개 요소를 현재 출력 스트림에 쓰기
<code>void write(String str)</code> <code>throws IOException</code>	String str로 전달된 문자열을 현재 출력 스트림에 쓰기
<code>void write(String str, int off, int len)</code> <code>throws IOException</code>	String str의 인덱스 off부터 len개 문자열을 현재 출력 스트림에 쓰기

# FileWriter 클래스

- 텍스트 파일을 생성하는 문자 출력 스트림
- 파일을 생성할 때 사용할 문자셋(Charset)을 지정할 수 있음
- 2가지 파일 생성 모드가 있음
  - 생성 모드 : 항상 새로운 파일을 만듦
  - 추가 모드 : 기존에 지정된 파일이 있으면 해당 파일의 끝에 추가함

# FileWriter 클래스 - 생성자

## ■ FileWriter 클래스 주요 생성자

생성자	역할
FileWriter(File file) <b>throws</b> IOException	전달된 File 객체가 나타내는 파일에 쓰기 위한 파일 출력 스트림 생성
FileWriter(File file, boolean append) <b>throws</b> IOException	전달된 File 객체가 나타내는 파일에 쓰기 위한 파일 출력 스트림 생성 boolean append 값이 true이면 추가 모드로 파일 출력 스트림 생성
FileWriter(File file, Charset charset) <b>throws</b> IOException	전달된 File 객체가 나타내는 파일에 쓰기 위한 파일 출력 스트림 생성 Charset charset 인코딩 방식을 사용
FileWriter(String fileName) <b>throws</b> IOException	이름이 fileName인 파일에 쓰기 위한 파일 출력 스트림 생성
FileWriter(String fileName, boolean append) <b>throws</b> IOException	이름이 fileName인 파일에 쓰기 위한 파일 출력 스트림 생성 boolean append 값이 true이면 추가 모드로 파일 출력 스트림 생성
FileWriter(String fileName, Charset charset) <b>throws</b> IOException	이름이 fileName인 파일에 쓰기 위한 파일 출력 스트림 생성 Charset charset 인코딩 방식을 사용

# FileWriter 클래스 - test1.txt 생성

## ■ test1.txt 파일을 새로 만들기

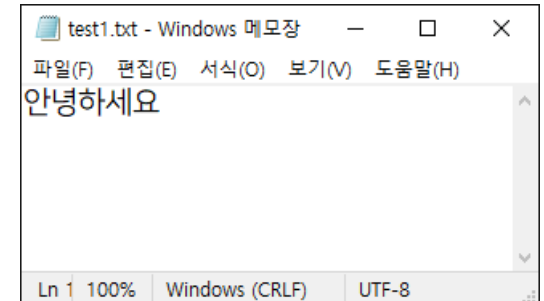
```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileWriter writer = new FileWriter("test1.txt");  
            String str = "안녕하세요";  
            writer.write(str);  
            writer.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과



test1.txt  
텍스트 문서  
15바이트

(크기 15바이트)

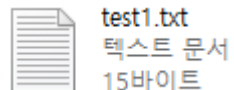


# FileWriter 클래스 - test1.txt 추가

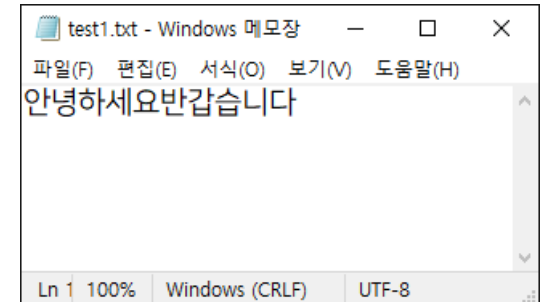
## ■ test1.txt 파일에 데이터 추가하기

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileWriter writer = new FileWriter("test1.txt", true);  
            String str = "반갑습니다";  
            writer.write(str);  
            writer.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과



(크기가 30바이트로 증가)



# BufferedWriter 클래스

- 버퍼(Buffer)를 제공하는 보조 출력 스트림
  - 단독 사용 불가
  - 다른 Writer 클래스와 함께 사용해야 함
- 출력 데이터를 저장할 수 있는 내부 버퍼를 가지고 있음
  - `private char cb[]` 필드를 가지고 있음
- BufferedWriter 클래스를 이용하면 출력 속도를 향상시킬 수 있음

# BufferedWriter 클래스 - 생성자

## ■ BufferedWriter 클래스 생성자

생성자	역할
BufferedWriter(Writer out)	기본 크기의 출력 버퍼를 사용하는 버퍼링된 문자 출력 스트림 생성 기본 크기는 충분히 큰 크기(defaultCharBufferSize=8192)를 제공함
BufferedWriter(Writer out, int sz)	int sz 크기의 출력 버퍼를 사용하는 버퍼링된 문자 출력 스트림 생성 int sz 값이 0 이하이면 IllegalArgumentException 예외 발생

# BufferedWriter 클래스 - 메소드

- BufferedWriter 클래스 메소드 (Writer 클래스에는 없는 메소드)

메소드	역할
<code>void newLine()</code> <code>throws IOException</code>	줄 바꿈 기호를 삽입함 플랫폼에 따라 줄 바꿈 기호가 "w\n"이 아닐 수도 있으나 사용자는 신경 쓸 필요가 없음



# BufferedWriter 클래스 - test2.txt 생성

## ■ test2.txt 파일을 새로 만들기

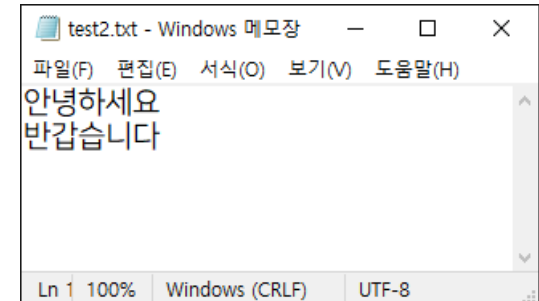
```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedWriter writer = new BufferedWriter(new FileWriter("test2.txt"));  
            String str1 = "안녕하세요";  
            String str2 = "반갑습니다";  
            writer.write(str1);  
            writer.newLine();  
            writer.write(str2);  
            writer.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과



test2.txt  
텍스트 문서  
32바이트

(크기 32바이트)



# PrintWriter 클래스

- `PrintStream` 클래스의 모든 `print` 관련 메소드를 구현한 문자 출력 스트림
  - `print()`
  - `println()`
  - `printf()`
- 바이트 기반 출력 스트림 `OutputStream` 클래스 객체를 문자 기반 출력 스트림 `PrintWriter`로 곧바로 바꿀 수도 있음

# PrintWriter 클래스 - 생성자

## ■ PrintWriter 클래스 주요 생성자

생성자	역할
<code>PrintWriter(File file)</code>	전달된 File 객체가 나타내는 파일에 쓰기 위한 PrintWriter 출력 스트림 생성
<code>PrintWriter(File file, Charset charset)</code>	전달된 File 객체가 나타내는 파일에 쓰기 위한 PrintWriter 출력 스트림 생성 Charset charset 인코딩 방식을 사용
<code>PrintWriter(OutputStream out)</code>	전달된 OutputStream out에서 새로운 PrintWriter 출력 스트림 생성
<code>PrintWriter(OutputStream out, boolean autoFlush)</code>	전달된 OutputStream out에서 새로운 PrintWriter 출력 스트림 생성 boolean autoFlush 값이 true이면 <code>printf()</code> , <code>println()</code> 메소드가 자동 플러시 동작을 함
<code>PrintWriter(Writer out)</code>	전달된 Writer out에서 새로운 PrintWriter 출력 스트림 생성
<code>PrintWriter(Writer out, boolean autoFlush)</code>	전달된 Writer out에서 새로운 PrintWriter 출력 스트림 생성 boolean autoFlush 값이 true이면 <code>printf()</code> , <code>println()</code> 메소드가 자동 플러시 동작을 함
<code>PrintWriter(String filename)</code>	이름이 filename인 파일에 쓰기 위한 PrintWriter 출력 스트림 생성
<code>PrintWriter(String filename, Charset charset)</code>	이름이 filename인 파일에 쓰기 위한 PrintWriter 출력 스트림 생성 Charset charset 인코딩 방식을 사용

# PrintWriter 클래스 - 메소드

## ■ PrintWriter 클래스 주요 메소드 (Writer 클래스에는 없는 메소드)

메소드	역할
void print(Type a)	전달된 Type a를 현재 PrintWriter 출력 스트림에 쓰기
void println()	줄 바꿈을 현재 PrintWriter 출력 스트림에 쓰기
void println(Type a)	전달된 Type a를 현재 PrintWriter 출력 스트림에 쓰고 줄 바꿈 처리
PrintWriter printf(String format, Object... args)	지정된 형식 문자열 String format과 전달된 Object... args 값을 사용하여 현재 PrintWriter 출력 스트림에 쓰기
* Type : boolean, char, int, long, float, double, char[], String, Object 가능	

# PrintWriter 클래스 - test3.txt 생성

## ■ test2.txt 파일을 새로 만들기

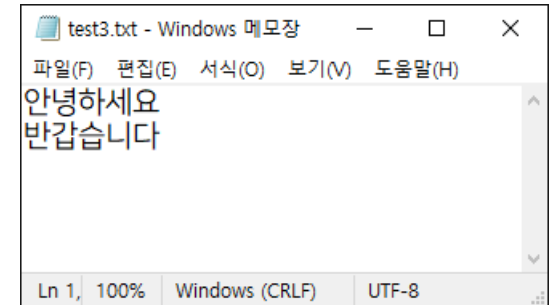
```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            PrintWriter writer = new PrintWriter("test3.txt");  
            String str1 = "안녕하세요";  
            String str2 = "반갑습니다";  
            writer.println(str1);  
            writer.println(str2);  
            writer.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### 파일 생성 결과



test3.txt  
텍스트 문서  
34바이트

(크기 34바이트)



```

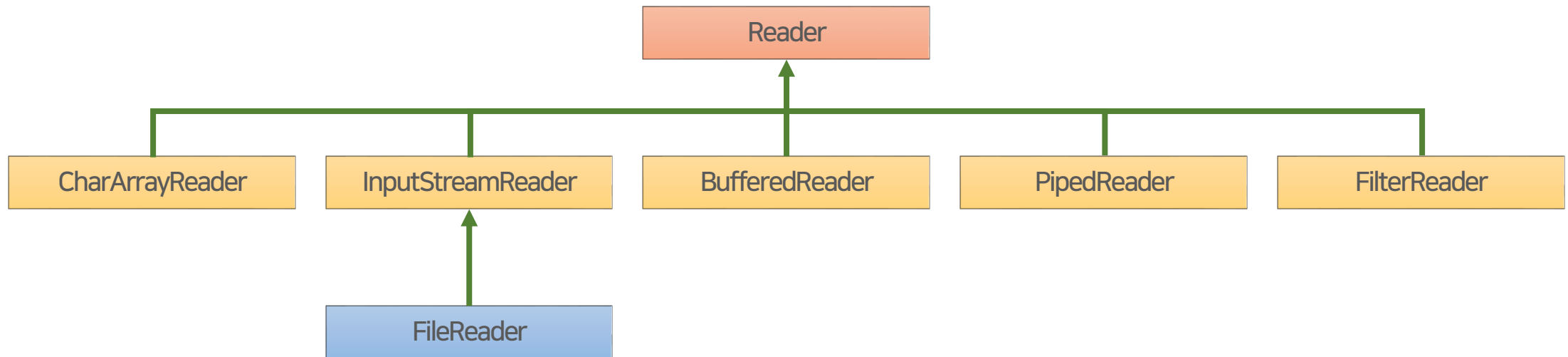
each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) :
      if (n in t && t[n] === e) return n;
  }
}

```

## 6. 문자 입력 스트림

# Reader

- 모든 문자 입력 스트림의 최상위 슈퍼 클래스
- 주요 Reader 구조도



# Reader 클래스 - 메소드

- 모든 문자 입력 스트림은 Reader 클래스의 메소드를 사용할 수 있음
- Reader 클래스의 주요 메소드

메소드	역할
<code>void close()</code> <code>throws IOException</code>	입력 스트림을 닫고 이 스트림과 관련된 모든 시스템 리소스 해제
<code>int read()</code> <code>throws IOException</code>	입력 스트림에서 한 글자를 읽어서 반환 읽은 글자가 없으면 -1 반환
<code>int read(char[] cbuf)</code> <code>throws IOException</code>	입력 스트림에서 일부 글자들을 읽고 <code>char[] cbuf</code> 배열에 저장함 입력 스트림에서 읽은 글자 수를 반환, 읽은 글자가 없으면 -1 반환
<code>int read(char[] cbuf, int off, int len)</code> <code>throws IOException</code>	입력 스트림에서 최대 <code>len</code> 글자를 읽고 <code>char[] cbuf</code> 배열의 인덱스 <code>off</code> 부터 저장 입력 스트림에서 읽은 글자 수를 반환, 읽은 글자가 없으면 -1 반환
<code>long transferTo(Writer out)</code> <code>throws IOException</code>	입력 스트림에서 모든 글자를 읽고 이를 지정된 출력 스트림 <code>Writer out</code> 에 쓰기 입력 스트림에서 출력 스트림으로 보낸 글자 수를 반환



# FileReader 클래스

- 텍스트 파일을 읽는 문자 입력 스트림
- 파일을 읽을 때 사용할 문자셋을 지정할 수 있음
- 파일을 찾을 수 없는 경우 FileNotFoundException 예외가 발생함

# FileReader 클래스 - 생성자

## ■ FileReader 클래스 주요 생성자

생성자	역할
<code>FileReader(File file)</code> <code>throws FileNotFoundException</code>	전달된 File 객체가 나타내는 파일을 읽기 위한 파일 입력 스트림 생성
<code>FileReader(File file, Charset charset)</code> <code>throws FileNotFoundException</code>	전달된 File 객체가 나타내는 파일을 읽기 위한 파일 입력 스트림 생성 Charset charset 인코딩 방식을 사용
<code>FileReader(String fileName)</code> <code>throws FileNotFoundException</code>	이름이 fileName인 파일을 읽기 위한 파일 입력 스트림 생성
<code>FileReader(String fileName, Charset charset)</code> <code>throws FileNotFoundException</code>	이름이 fileName인 파일을 읽기 위한 파일 입력 스트림 생성 Charset charset 인코딩 방식을 사용

# FileReader 클래스 - test1.txt 읽기1

## ■ public int read() 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileReader reader = new FileReader("test1.txt");  
            int c = 0;  
            StringBuilder sb = new StringBuilder();  
            while((c = reader.read()) != -1) {  
                sb.append((char)c);  
            }  
            System.out.println(sb.toString());  
            reader.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"안녕하세요반갑습니다"

# FileReader 클래스 - test1.txt 읽기2

## ■ public int read(byte[] b) 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            FileReader reader = new FileReader("test1.txt");  
            char[] cbuf = new char[3];  
            StringBuilder sb = new StringBuilder();  
            int readChar = 0;  
            while((readChar = reader.read(cbuf)) != -1) {  
                sb.append(cbuf, 0, readChar);  
            }  
            System.out.println(sb.toString());  
            reader.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"안녕하세요반갑습니다"

# BufferedReader 클래스

- 버퍼(Buffer)를 제공하는 보조 입력 스트림
  - 단독 사용 불가
  - 다른 Reader 클래스와 함께 사용해야 함
- 입력 데이터를 저장할 수 있는 내부 버퍼를 가지고 있음
  - `private char cb[]` 필드를 가지고 있음
- BufferedReader 클래스를 이용하면 입력 속도를 향상시킬 수 있음

# BufferedReader 클래스 - 생성자

## ■ BufferedReader 클래스 생성자

생성자	역할
BufferedReader(Reader in)	기본 크기의 입력 버퍼를 사용하는 버퍼링된 문자 입력 스트림 생성 기본 크기는 충분히 큰 크기(defaultCharBufferSize=8192)를 제공함
BufferedReader(Reader in, int sz)	int sz 크기의 입력 버퍼를 사용하는 버퍼링된 문자 입력 스트림 생성 int sz 값이 0 이하이면 IllegalArgumentException 예외 발생

# BufferedReader 클래스 - 메소드

- BufferedReader 클래스 메소드 (Reader 클래스에는 없는 메소드)

메소드	역할
String readLine() <b>throws</b> IOException	줄(Line) 단위로 읽어서 반환 문서 끝에 도달하면(EOF) null 값을 반환

# BufferedReader 클래스 - test2.txt 읽기

## ■ public int readLine() 메소드

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedReader reader = new BufferedReader(new FileReader("test2.txt"));  
            String line = null;  
            StringBuilder sb = new StringBuilder();  
            while((line = reader.readLine()) != null) {  
                sb.append(line).append("\n");  
            }  
            sb.deleteCharAt(sb.length() - 1);  
            System.out.println(sb.toString());  
            reader.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

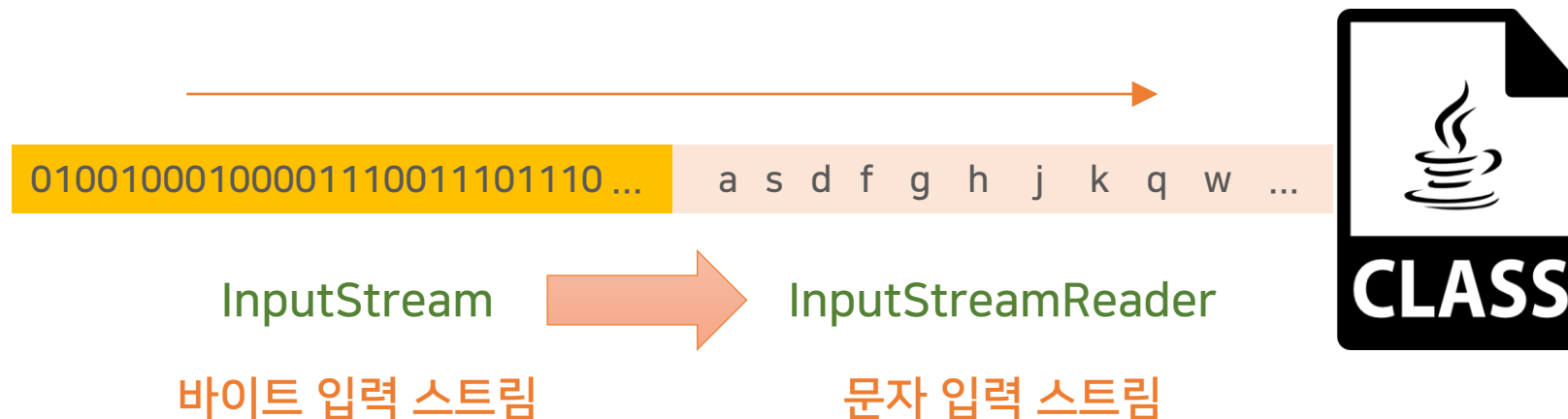
실행결과

"안녕하세요"  
"반갑습니다"



# 바이트 입력 스트림을 문자 입력 스트림으로 변환하기

- 바이트 입력 스트림으로 문자 데이터를 읽어 들이는 경우 문자가 깨질 우려가 있음
- 이런 경우에는 바이트 입력 스트림을 문자 입력 스트림으로 바꾼 뒤 문자 데이터를 읽어 들이게 처리할 수 있음



# InputStreamReader 클래스

- 바이트 기반 입력 스트림인 InputStream 클래스를 문자 기반 입력 스트림인 Reader 클래스로 변환하는 클래스
- 읽기 성능 향상을 위해서 BufferedReader 클래스와 함께 사용할 수 있음

# InputStreamReader 클래스 - 생성자

## ■ InputStreamReader 클래스 주요 생성자

생성자	역할
<code>InputStreamReader(InputStream in)</code>	InputStream in으로 InputStreamReader 입력 스트림 생성 기본 문자셋을 사용
<code>InputStreamReader(InputStream in, String charsetName)</code> <code>throws</code> <code>UnsupportedEncodingException</code>	InputStream in으로 InputStreamReader 입력 스트림 생성 String charset 문자셋을 사용
<code>InputStreamReader(InputStream in, Charset cs)</code>	InputStream in으로 InputStreamReader 입력 스트림 생성 Charset cs 문자셋을 사용

# InputStream 클래스 - test3.txt 읽기

- 텍스트 파일을 바이트 입력 스트림으로 읽은 경우 한글 깨짐 확인

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            InputStream in = new FileInputStream("test3.txt");  
            int c = 0;  
            StringBuilder sb = new StringBuilder();  
            while((c = in.read()) != -1) {  
                sb.append((char)c);  
            }  
            System.out.println(sb.toString());  
            in.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

```
"ìëíì,ì"  
"ë°ê°ìµëëα"
```

# InputStreamReader 클래스 - test3.txt 읽기

- InputStreamReader 클래스 사용으로 깨짐 없이 읽기 성공

```
public class MainClass {  
    public static void main(String[] args) {  
        try {  
            BufferedReader reader  
                = new BufferedReader(new InputStreamReader(new FileInputStream("test3.txt")));  
            int c = 0;  
            StringBuilder sb = new StringBuilder();  
            while((c = reader.read()) != -1) {  
                sb.append((char)c);  
            }  
            System.out.println(sb.toString());  
            reader.close();  
        } catch(IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

실행결과

"안녕하세요"  
"반갑습니다"