File I/O, Exception Handling

514760
2025년 봄학기
5/29/2025
박경신

자바의 스트림

- 자바의 스트림
  - 데이터 스트림(stream)을 읽고(read) 쓰도록 (write) 구현
  - 자바 입출력 스트림 특징은 단방향 스트림, 선입선출 구조
  - 바이트(byte) 또는 문자 단위로 구분해서 처리
  - 텍스트 입출력을 할 때는 줄바꿈 문자에 대해서 알고 있어야 함
    - 운영체제에 따라서 줄바꿈 문자가 다르게 표현됨
  - 입력 스트림
    - 입력은 메모리, 다른 프로그램, 파일 등과 같은 출처(source)로부터 데이터를 읽음
  - 출력 스트림
    - 자바 프로그램에서 메모리, 다른 프로그램, 파일 등과 같은 출처(source)로 데이터를 출력

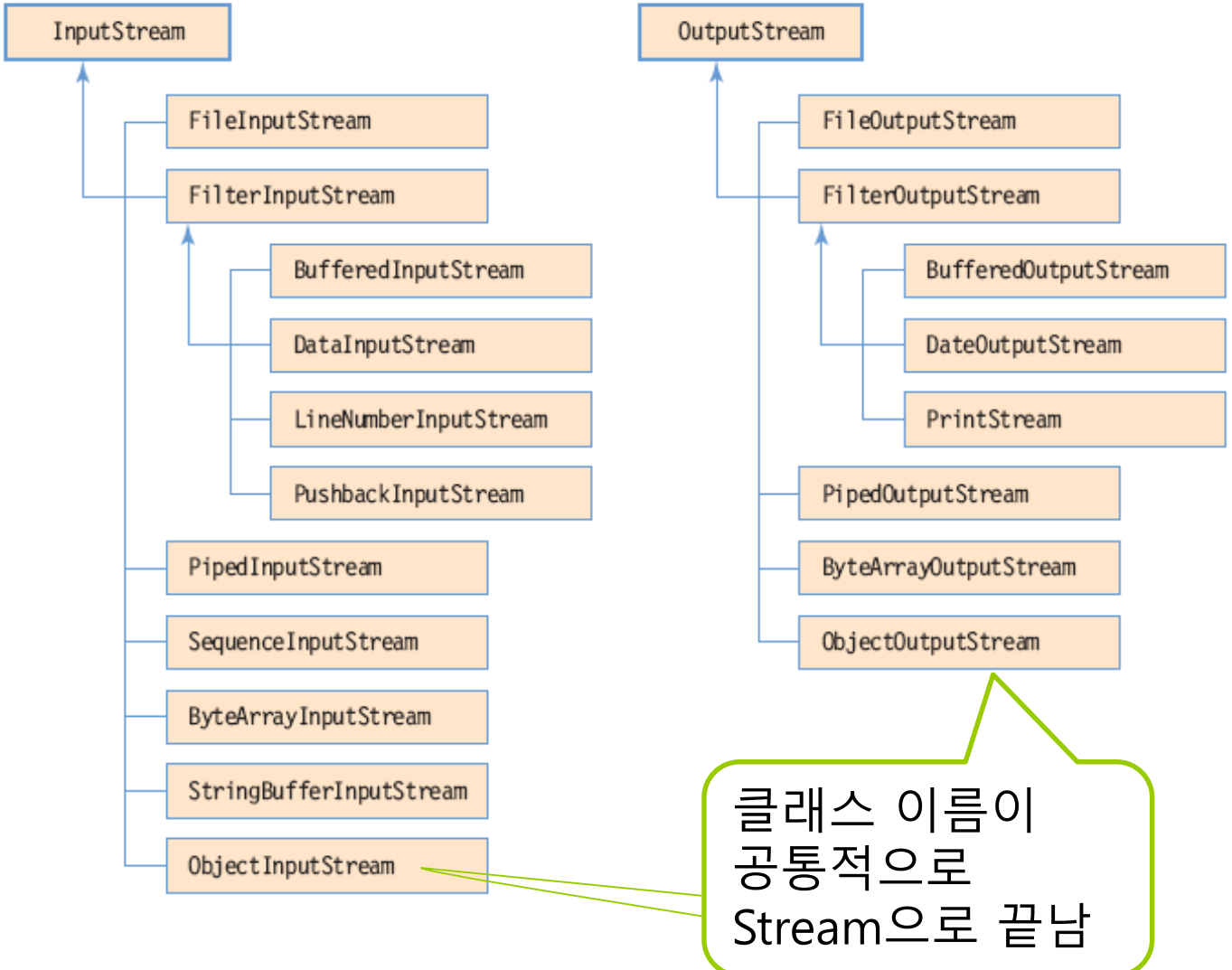
자바의 입출력 스트림 종류

- 자바의 입출력은 java.io 패키지에서 처리
- **바이트** 입출력 스트림과 **문자** 입출력 스트림
  - 바이트 입출력 스트림
    - 입출력 데이터를 단순 바이트의 스트림으로 처리
    - 바이트의 입출력은 8비트 단위로 사용됨
    - 예) 바이너리 파일을 읽는 입력 스트림
  - 문자 입출력 스트림
    - 문자만 입출력하는 스트림
    - 문자는 유니코드 방식으로 입출력
    - 문자가 아닌 바이너리 데이터는 스트림에서 처리하지 못함
    - 예) 텍스트 파일을 읽는 입력 스트림

구분	입력	출력
바이트	InputStream	OutputStream
문자	Reader	Writer

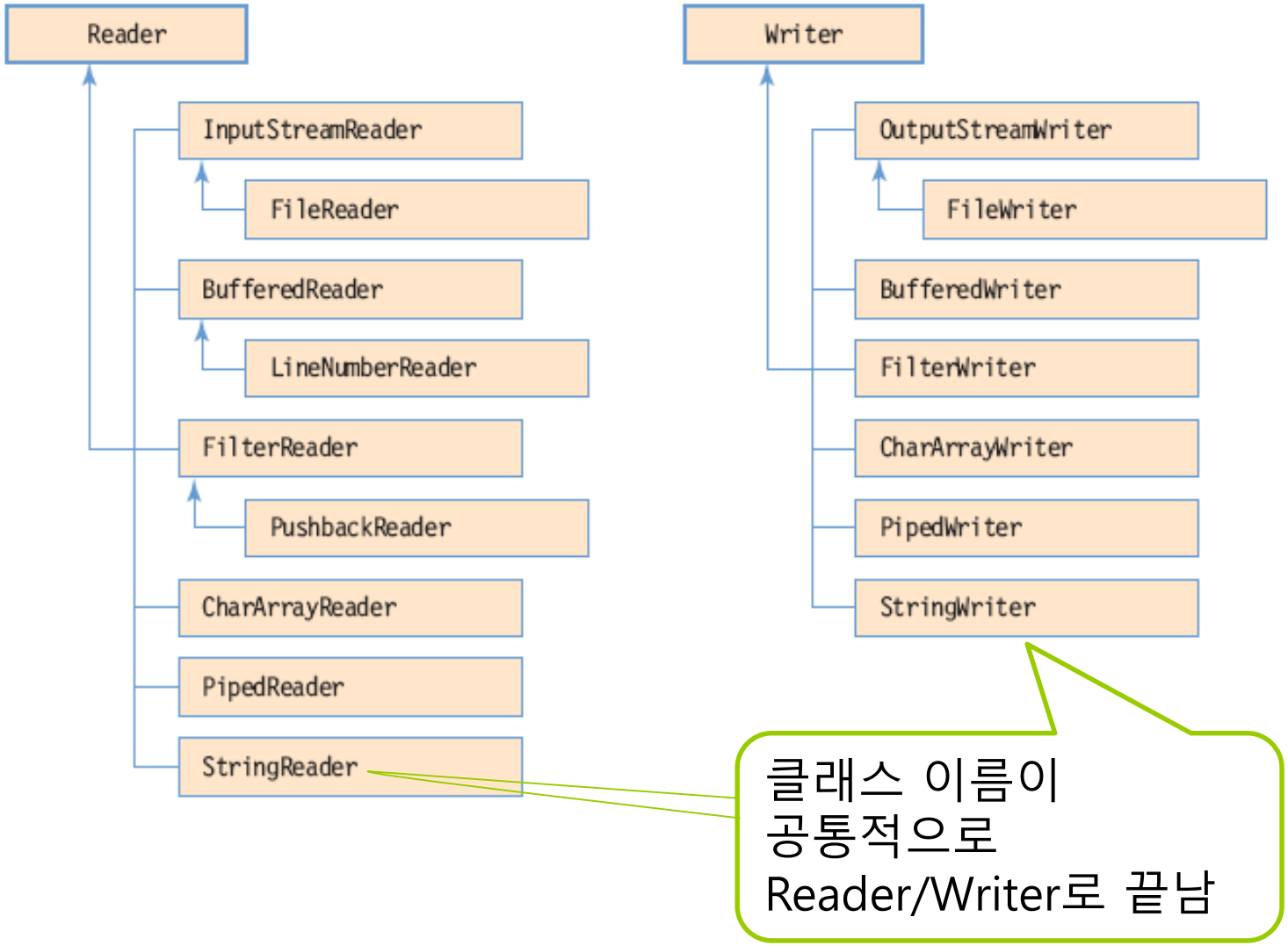
JDK의 바이트 스트림 클래스 계층 구조

- JDK는 입출력 스트림을 구현한 다양한 클래스 제공



JDK의 문자 스트림 클래스 계층 구조

- JDK는 입출력 스트림을 구현한 다양한 클래스 제공



바이트 스트림 클래스

- 바이트 스트림
  - 바이트 단위의 바이너리 값을 읽고 쓰는 스트림
- 바이트 스트림 클래스
  - InputStream/OutputStream 추상 클래스
    - 바이트 스트림을 다루는 모든 클래스의 슈퍼 클래스
    - 바이트 단위로 데이터 입력/출력하는 read()/write() 메소드
  - FileInputStream/FileOutputStream
    - 파일로부터 바이트 단위로 읽거나 저장하는 클래스
    - 바이너리 파일의 입출력 용도
  - DataInputStream/DataOutputStream
    - 자바의 기본 데이터 타입의 값(변수)을 바이너리 값 그대로 입출력
    - 문자열도 바이너리 형태로 입출력

FileInputStream을 이용한 파일 읽기

- 파일 전체를 읽어 화면에 출력하는 코드 샘플

```
FileInputStream fin = new FileInputStream("c:₩₩test.txt");

int c;

while((c = fin.read()) != -1) {

   System.out.print((char)c);

}

fin.close();
```

C:₩test.txt 파일을 열고 파일과 입력 바이트 스트림 객체 fin 연결

파일 끝까지 바이트씩 c에 읽어 들임. 파일의 끝을 만나면 read()는 -1 리턴

바이트 c를 문자로 변환하여 화면에 출력

스트림을 닫음. 파일도 닫힘. 스트림과 파일의 연결을 끊음. 더 이상 스트림으로부터 읽을 수 없음

예제 : 윈도우에 있는 system.ini 파일을 읽어 화면에 출력하기

FileInputStream을 이용하여 사용자 컴퓨터의 windows 디렉터리에 있는 system.ini 파일을 읽고 화면에 출력하라. system.ini 파일은 텍스트 파일이다.

```
import java.io.*;
public class FileInputStreamEx {
  public static void main(String[] args) {
    FileInputStream in = null;
    try {
      in = new FileInputStream("c:₩₩windows₩₩system.ini");
      int c;
      while ((c = in.read()) != -1) {
        System.out.print((char)c);
      }
      in.close();
    } catch (IOException e) {
        System.out.println("입출력 오류");
    }
  }
}
```

파일 끝을 만나면 -1 리턴

```
; for 16-bit app support
[386Enh]
woafont=dosapp.fon
EGA80WOA.FON=EGA80WOA.FON
EGA40WOA.FON=EGA40WOA.FON
CGA80WOA.FON=CGA80WOA.FON
CGA40WOA.FON=CGA40WOA.FON

[drivers]
wave=mmdrv.dll
timer=timer.drv

[mci]
```

FileOutputStream을 이용한 파일 쓰기

- 바이너리 값을 파일에 저장하는 바이트 스트림 코드

```
FileOutputStream fout = new FileOutputStream("c:₩₩test.out");

int num[]={1,4,-1,88,50};
byte b[]={7,51,3,4,1,24};

for(int i=0; i<num.length; i++)
  fout.write(num[i]);

fout.write(b);

fout.close();
```

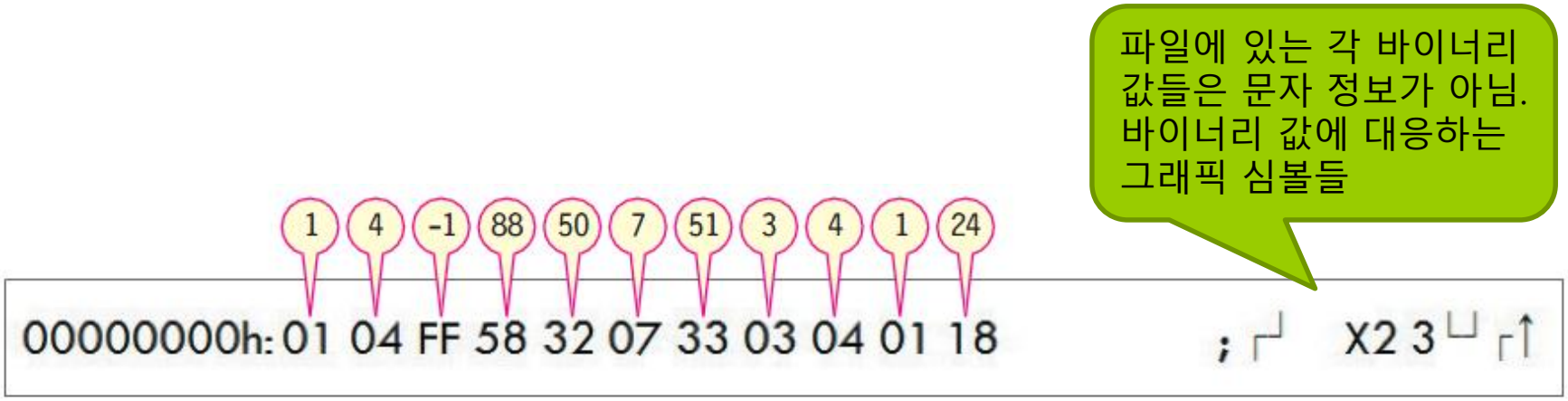
c:₩test.out 파일을 열고, 출력 바이트 스트림인 객체와 연결

파일에 배열 num[i]의 정수 값(바이너리)을 그대로 기록

파일에 바이트 배열(바이너리) 값을 그대로 기록

스트림을 닫음. 파일도 닫힘. 더 이상 스트림으로부터 읽을 수 없음

FileOutputStream을 이용한 파일 쓰기



test.out 파일의 내부

예제 : FileOutputStream을 이용한 파일 쓰기

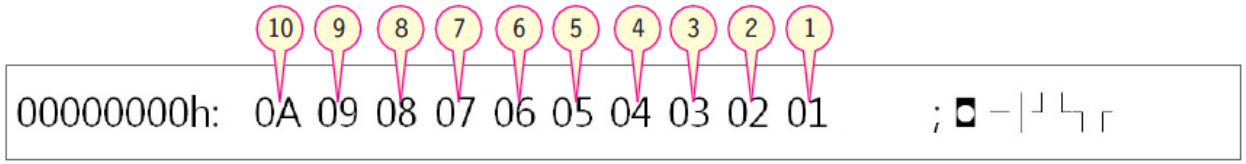
정수 타입의 결과 값을 FileOutputStream을 이용하여 파일에 저장한다. 다시 이 파일에서 정수형 변수로 읽고 이전에 계산된 결과 값과 같은지 확인하라.

```
import java.io.IOException;
public class FileOutputStreamEx {
  public static void main(String[] args) {
    try {
      FileOutputStream fout = new FileOutputStream("c:₩₩test.out");
      FileInputStream fin = null;

      for (int i=0; i<10; i++) {
        int n = 10-i; // 계산의 결과를 저장
        fout.write(n); // 파일에 결과값을 바이너리로 저장
      }
      fout.close(); //스트림을 닫는다.
```

예제 : FileOutputStream을 이용한 파일 쓰기

```
        fin = new FileInputStream("c:₩₩test.out");
        int c=0;
        while ((c = fin.read()) != -1) {
          System.out.print(c + " ");
        }
        fin.close();
      } catch (IOException e) {
        System.out.println("입출력 오류");
      }
    }
}
```

10 9 8 7 6 5 4 3 2 1



문자 스트림

- 문자 스트림
  - 유니 코드로 된 문자 데이터만 입출력
  - 문자로 표현되지 않는 데이터 (예를 들어, 이미지, 동영상과 같은 바이너리 데이터)는 입출력 할 수 없음
- 문자 스트림을 다루는 클래스
  - Reader/Writer 추상클래스
    - 문자 스트림을 다루는 모든 클래스의 슈퍼 클래스
    - 문자 단위로 입력/출력하는 read()/write() 메소드
  - InputStreamReader/OutputStreamWriter
    - 바이트 스트림과 문자 스트림을 연결시켜주는 다리 역할
    - 지정된 문자집합 이용
    - InputStreamReader : 바이트를 읽어 문자로 인코딩
    - OutputStreamWriter : 문자를 바이트로 디코딩하여 출력
  - **FileReader/FileWriter**
    - 텍스트 파일에서 문자 데이터 입출력

예제 : FileReader를 이용한 텍스트 파일 읽기 - system.ini 파일 읽기

FileReader를 이용하여 사용자 컴퓨터의 windows 디렉터리에 있는 system.ini 파일을 읽고 화면에 출력하라. system.ini 파일은 텍스트 파일이다.

```
import java.io.*;
public class FileReaderEx {
  public static void main(String[] args) {
    FileReader in = null;
    try {
      // 파일로부터 문자 입력 스트림 생성
      in = new FileReader("c:₩₩windows₩₩system.ini");
      int c;
      while ((c = in.read()) != -1) { // 한 문자씩 읽는다.
            System.out.print((char)c);
      }
      in.close();
    }
    catch (IOException e) {
      System.out.println("입출력 오류");
    }     }   }
```

파일의 끝을 만나면 read()는 -1 리턴

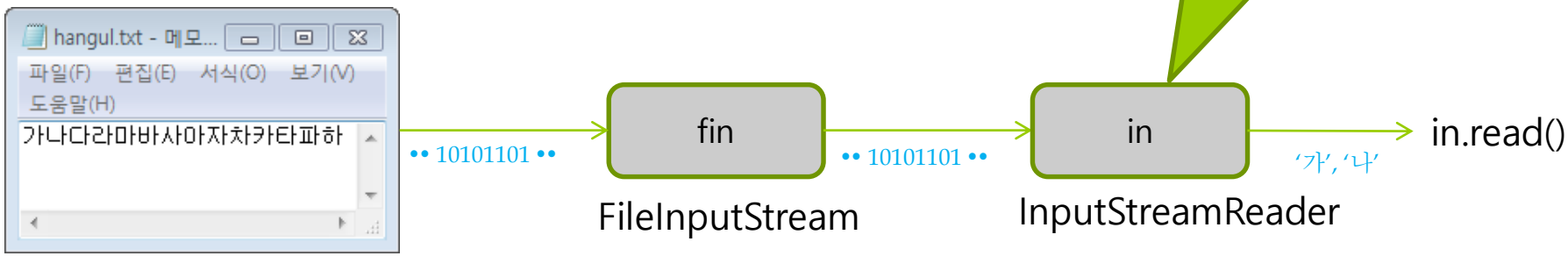
문자 집합과 InputStreamReader로 텍스트 파일 읽기

```
FileInputStream fin = new FileInputStream("c:₩₩tmp₩₩hangul.txt");
InputStreamReader in = new InputStreamReader(fin, "MS949");
```

```
while ((c = in.read()) != -1) {
  System.out.print((char)c);
}
```

한글 확장 완성형 문자 집합

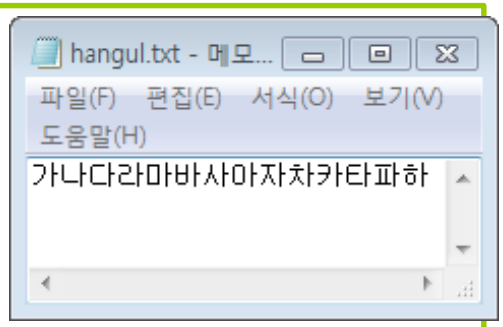
문자 집합 사용
(윈도우에서 MS949)



예제: 한글 텍스트 파일 읽기

InputStreamReader를 이용하여 MS949 문자 집합으로 한글 텍스트 파일을 읽고 출력하라.

```
import java.io.*;
public class FileReadHangulSuccess {
  public static void main(String[] args) {
    InputStreamReader in = null;
    FileInputStream fin = null;
    try {
      fin = new FileInputStream("c:₩₩tmp₩₩hangul.txt");
      in = new InputStreamReader(fin, "MS949");
      int c;
      System.out.println("인코딩 문자 집합은 " + in.getEncoding());
      while ((c = in.read()) != -1) {
        System.out.print((char)c);
      }
      in.close();
      fin.close();
    } catch (IOException e) { System.out.println("입출력 오류"); } } }
```



hangul.txt

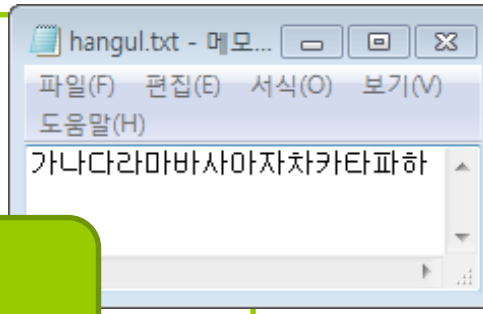
MS에서 만든 한글 확장 완성형 문자 집합

인코딩 문자 집합은 MS949
가나다라마바사아자차카타파하

예제: 문자 집합 지정이 잘못된 한글 텍스트 파일 읽기

InputStreamReader의 문자 집합을 US-ASCII로 지정하여 한글 파일을 읽고 출력하라.

```
import java.io.*;
public class FileReadHangulFail {
  public static void main(String[] args) {
    InputStreamReader in = null;
    FileInputStream fin = null;
    try {
      fin = new FileInputStream("c:₩₩tmp₩₩hangul.txt");
      in = new InputStreamReader(fin, "US-ASCII");
      int c;
      System.out.println("인코딩 문자 집합은 " + in.getEncoding());
      while ((c = in.read()) != -1) {
        System.out.print((char)c);
      }
      in.close();
      fin.close();
    } catch (IOException e) { System.out.println("입출력 오류"); } }}
```



문자 집합 지정이 잘못된 경우의 예를 보이기 위해 일부러 틀린 문자 집합 지정

인코딩 문자 집합은 ASCII
??????????????????????????

문자 집합 지정이 잘못되어 읽은 문자가 제대로 인식되지 못함.
출력 결과가 깨짐

FileWriter 사용 예

- c:₩test.txt 파일에 문자 출력 스트림을 생성하는 코드

```
FileWriter fout = new FileWriter("c:₩₩tmp₩₩test.txt");
```

- 파일에 문자 출력

```
FileWriter fout = new FileWriter("c:₩₩tmp₩₩test.txt");
fout.write('A'); // 문자 'A' 출력
fout.close();
```

예제: 키보드 입력을 파일로 저장하기

키보드 입력 데이터를 c:₩tmp₩test.txt 파일에 저장하는 코드를 작성하라.

```
import java.io.*;
public class FileWriterEx {
  public static void main(String[] args) {
    InputStreamReader in = new InputStreamReader(System.in);

    FileWriter fout = null;
    int c;
    try {
      fout = new FileWriter("c:₩₩tmp₩₩test.txt");
      while ((c = in.read()) != -1) {
        fout.write(c);
      }
      in.close();
      fout.close();
    } catch (IOException e) { System.out.println("입출력  오류");  }
  }
}
```

콘솔에서 abcdef 입력 후 <Enter> 키와 ctrl-z키 입력

실행 결과 test.txt 파일 생성

버퍼 입출력 스트림과 버퍼 입출력의 특징

- 버퍼 스트림
  - **버퍼를 가진 스트림**
  - 입출력 데이터를 일시적으로 저장하는 **버퍼**를 이용하여 입출력 효율 개선
  - 독자적으로 사용할 수 없고, 바이트 입출력 스트림 또는 문자 입출력 스트림과 연동해서 사용해야 함
- 버퍼 입출력의 목적
  - 입출력 시 운영체제의 API 호출 횟수를 줄여 입출력 성능 개선
    - 출력 시 여러 번 출력되는 데이터를 버퍼에 모아두고 한 번에 장치로 출력
    - 입력 시 입력 데이터를 버퍼에 모아두고 한번에 프로그램에게 전달

버퍼 스트림의 종류

- **바이트 버퍼 스트림**
  - **BufferedInputStream와 BufferedOutputStream**
  - 바이트 단위의 바이너리 데이터를 처리하는 버퍼 스트림
- **문자 버퍼 스트림**
  - **BufferedReader와 BufferedWriter**
  - 유니코드의 문자 데이터만 처리하는 버퍼 스트림
  - BufferedReader는 한 줄(line)씩 읽는 readLine() 메소드 제공

20바이트 버퍼를 가진 BufferedOutputStream

```
BufferedOutputStream bout = new BufferedOutputStream(System.out, 20);

FileReader fin = new FileReader("c:₩₩windows₩₩system.ini");

int c;
while ((c = fin.read()) != -1) {
  bout.write((char)c);
}
fin.close();
bout.close();
```

파일 전체를 읽어 화면에 출력

스트림 닫음

20바이트 크기의 버퍼 설정.
System.out 표준 스트림에 출력

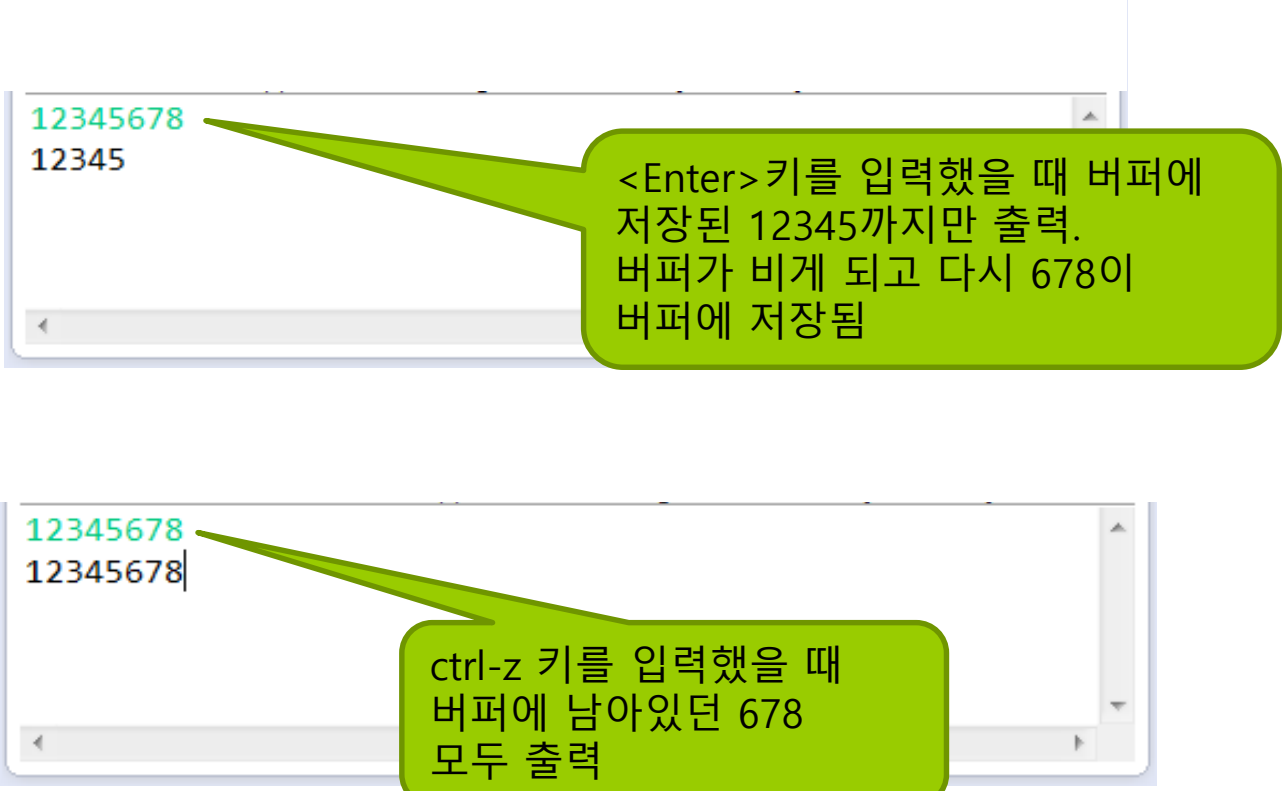
예제: 버퍼 스트림을 이용하는 출력 예제

```
import java.io.*;
public class BufferedIOEx {
  public static void main(String[] args) {
    InputStreamReader in =   new InputStreamReader(System.in);
    BufferedOutputStream out = new BufferedOutputStream(System.out, 5);
    try {
      int c;
      while ((c = in.read()) != -1)
          out.write(c);
      out.flush(); // 버퍼에 남아 있던 문자 출력
      if (in != null) {
        in.close();
        out.close();
      }
    }
    catch (IOException e) {
      System.out.println("입출력 오류");
    }
  }
}
```

ctrl-z가 입력될 때까지 반복

버퍼 크기를 5로 하고, 표준 출력 스트림과 연결된 버퍼 출력 스트림을 생성하라. 키보드에서 입력 받은 문자를 출력 스트림에 출력하고, 입력의 끝을 알리면(ctrl-z) 버퍼에 남아 있는 모든 문자를 출력하는 프로그램을 작성하라.

예제: 버퍼 스트림을 이용하는 출력 예제



텍스트 파일 읽기

- BufferedReader 클래스를 사용한 텍스트 파일 read

```
import java.io.*;
public class BufferedReaderExample {
    public static void main(String args[]) throws Exception{
        FileReader fr = new FileReader("C:/test.txt");
        BufferedReader br = new BufferedReader(fr);
        int ch;
        while((ch=br.read())!=-1) {
            System.out.print((char)ch);
        }
        br.close();
        fr.close();
    }
}
```

파일 전체를 읽어 화면에 출력

텍스트 파일 읽기

- BufferedReader 클래스를 사용한 텍스트 파일 readLine (파일을 한 줄씩 읽어서 lines 배열에 저장)

```
import java.io.*;
public class BufferedReaderExample2 {
    static List<String> lines = new ArrayList<>();
    public static void main(String args[]) throws Exception{
        BufferedReader br = new BufferedReader(new FileReader("C:/test.txt"));
        int i = 0;
        String line = "";
        while ((line=br.readLine())!= null){
            lines.add(line);
        }
        br.close();
        for (String l : lines) System.out.println(l); // lines 리스트 출력
    }
}
```

한 줄씩 읽어 lines 배열에 저장

File 클래스

- File 클래스
  - java.io.File
  - 운영체제 파일 시스템의 파일 또는 디렉토리를 다루기 위해서 만들어진 클래스
    - 파일과 디렉터리 경로명의 추상적 표현
  - 파일의 이름, 경로 등을 구하기 위해 사용할 수도 있고, 파일의 삭제, 이름 변경, 디렉토리 생성, 파일 크기, 파일인지 디렉토리인지 확인하는 기능 등을 제공
    - File 객체는 파일 읽고 쓰기 기능 없음

File 클래스 생성자와 주요 메소드

메소드	설명
File(File parent, String child)	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
File(String pathname)	pathname이 나타내는 File 객체 생성
File(String parent, String child)	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
File(URI uri)	file:URI를 추상 경로명으로 변환하여 File 객체 생성

메소드	설명
boolean mkdir()	새로운 디렉터리 생성
String[] list()	디렉터리 내의 파일과 디렉터리 이름의 문자열 배열 리턴
File[] listFiles()	디렉터리 내의 파일 이름의 File 배열 리턴
boolean renameTo(File dest)	dest가 지정하는 파일 이름 변경
boolean delete()	파일 또는 디렉터리 삭제
long length()	파일의 크기 리턴. 디렉터리나 장치 파일인 경우 0 리턴
String getPath()	파일 경로명 전체를 문자열로 변환하여 리턴
String getName()	파일 또는 디렉터리 이름을 문자열로 리턴
boolean isFile()	일반 파일이면 true 리턴
boolean isDirectory()	디렉터리이면 true 리턴
long lastModified()	파일이 마지막으로 변경된 시간 리턴
boolean exists()	파일 또는 디렉터리가 존재하면 true 리턴

File 클래스 사용 예

파일 객체 생성

```
File f = new File("c:₩₩test.txt");
```

파일인지
디렉터리인지
구분

```
File f = new File("c:₩₩windows₩₩system.ini");
String res;
if(f.isFile()) // 파일 타입이면
  res = "파일";
else // 디렉터리 타입이면
  res = "디렉터리";
System.out.println(f.getPath() + "은 " + res + "입니다.");
```

```
c:₩windows₩system.ini은 파일입니다.
```

서브 디렉터리
리스트 얻기

```
File f = new File("c:₩₩tmp₩₩java_sample");
String[] filenames = f.list(); // 파일명 리스트 얻기
for (int i=0; i<filenames.length; i++) {
  File sf = new File(f, filenames[i]);
  System.out.print(filenames[i]);
  System.out.print("₩t파일 크기: " + sf.length());
}
```

예제: File 클래스 활용한 파일 관리

File 클래스를 이용하여 파일의 타입을 알아내고, 디렉터리에 있는 파일들을 나열하며, 디렉터리 이름을 변경하는 프로그램을 작성해보자.

```
import java.io.File;
public class FileClassExample {
  // 디렉터리에 포함된 파일과 디렉터리의 이름,
  // 크기, 수정 시간을 출력하는 메소드
  public static void dir(File fd) {
    // 디렉터리에 포함된 파일 리스트 얻기
    String[] filenames = fd.list();
    for (String s : filenames) {
      File f = new File(fd, s);
      long t = f.lastModified(); // 마지막으로 수정된 시간
      System.out.print(s);
      System.out.print("₩t파일 크기: " + f.length()); // 파일 크기
      System.out.printf("₩t수정한 시간: %tb %td %ta %tT₩n",t, t, t, t);
    }
  }
```

예제: File 클래스 활용한 파일 관리

```
public static void main(String[] args) {
  File f1 = new File("c:₩₩windows₩₩system.ini");
  File f2 = new File("c:₩₩tmp₩₩java_sample");
  File f3 = new File("c:₩₩tmp");
  String res;
  if(f1.isFile()) // 파일 타입이면
    res = "파일";
  else // 디렉터리 타입이면
    res = "디렉터리";
  System.out.println(f1.getPath() + "은 " + res + "입니다.");
  if (!f2.exists()) { //f2가 나타내는 파일이 존재하는지 검사
    if (!f2.mkdir()) // 존재하지 않으면 디렉터리 생성
      System.out.println("디렉터리 생성 실패");
  }
```

예제: File 클래스 활용한 파일 관리

```
    if(f2.isFile()) // 파일 타입이면
      res = "파일";
    else // 디렉터리 타입이면
      res = "디렉터리";
    System.out.println(f2.getPath() + "은 " + res + "입니다.");
    dir(f3); // c:₩tmp에 있는 파일과 디렉터리 화면에 출력

    // 파일 이름 변경
    f2.renameTo(new File("c:₩₩tmp₩₩javasample"));
    dir(f3);
  }
}
```

C:₩tmp의 파일과 디렉터리 리스트

```
c:₩windows₩system.ini은 파일입니다.
c:₩tmp₩java_sample은 디렉터리입니다.
hangul.txt  파일 크기: 28          수정한 시간: 11월 29 일 21:04:46
Hello.java  파일 크기: 469 수정한 시간: 10월 06 수 13:23:59
Hello2010.java 파일 크기: 126 수정한 시간: 10월 06 수 10:01:56
HelloDoc.java          파일 크기: 669 수정한 시간: 10월 06 수 14:23:32
java_sample            파일 크기: 0           수정한 시간: 11월 14 일 16:46:27
hangul.txt  파일 크기: 28          수정한 시간: 11월 29 일 21:04:46
Hello.java  파일 크기: 469 수정한 시간: 10월 06 수 13:23:59
Hello2010.java 파일 크기: 126 수정한 시간: 10월 06 수 10:01:56
HelloDoc.java          파일 크기: 669 수정한 시간: 10월 06 수 14:23:32
javasample             파일 크기: 0           수정한 시간: 11월 14 일 16:46:27
```

예제: 바이너리 파일 복사

바이너리 파일을 복사하는 프로그램을 작성하라

```
import java.io.*;
public class CopyFile {
  public static void copyFile(BufferedInputStream src,
BufferedOutputStream dest) {
    try {
      int data;
      while ((data = src.read()) != -1) {
        dest.write(data);
      }
    } catch (IOException e) {
      System.out.println("파일 복사 오류");
    }
  }
```

예제: 바이너리 파일 복사

```
    public static void main(String[] args) {
      String srcFilename = “dog.jpg”;
      String destFilename = “output.jpg“;
      try {
          // BufferedInputStream과 BufferedOutputStream 사용
          BufferedInputStream src = new BufferedInputStream(new
FileInputStream(srcFilename));
          BufferedOutputStream dest = new BufferedOutputStream(new
FileOutputStream(destFilename));
          copyFile(src, dest);
          src.close();
          dest.close();
       } catch (IOException e) {
          System.out.println("파일 복사 오류");
       }
   }
}
```

예제: 텍스트 파일 복사

문자 스트림을 이용하여 텍스트 파일을 복사하는 프로그램을 작성하라

```
import java.io.*;
public class CopyFile2 {
  public static void copyFile(BufferedReader src, BufferedWriter dest) {
    try {
      int data;
      while ((data = src.read()) != -1) {
        dest.write(data);
      }
    } catch (IOException e) {
      System.out.println("파일 복사 오류");
    }
  }
```

예제: 텍스트 파일 복사

```
    public static void main(String[] args) {
      String srcFilename = “hello.txt”;
      String destFilename = “output.txt“;
      try {
          // BufferedReader와 BufferedWriter 사용
          BufferedReader src = new BufferedReader(new
FileReader(srcFilename));
          BufferedWriter dest = new BufferedWriter(new
FileWriter(destFilename));
          copyFile(src, dest);
          src.close();
          dest.close();
       } catch (IOException e) {
          System.out.println("파일 복사 오류");
       }
   }
}
```

오류(Error)와 예외(Exception)

- 오류의 종류
  - 에러(Error)
    - 시스템의 잘못된 동작 또는 고장으로 인한 오류
    - 에러가 발생되면 JVM실행에 문제가 있으므로 프로그램 종료
    - 정상 실행 상태로 돌아갈 수 없음
  - 예외(Exception)
    - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
    - 예외가 발생되면 프로그램 종료
      - 정수를 0으로 나눔
      - 파일에서 데이터를 읽어야 하는데 파일이 없음
      - 파일을 읽는 과정에서 파일에 문제가 있어서 읽기 오류 발생
    - "예외 처리" 추가하면 정상 실행 상태로 돌아갈 수 있음

오류(Error)와 예외(Exception)

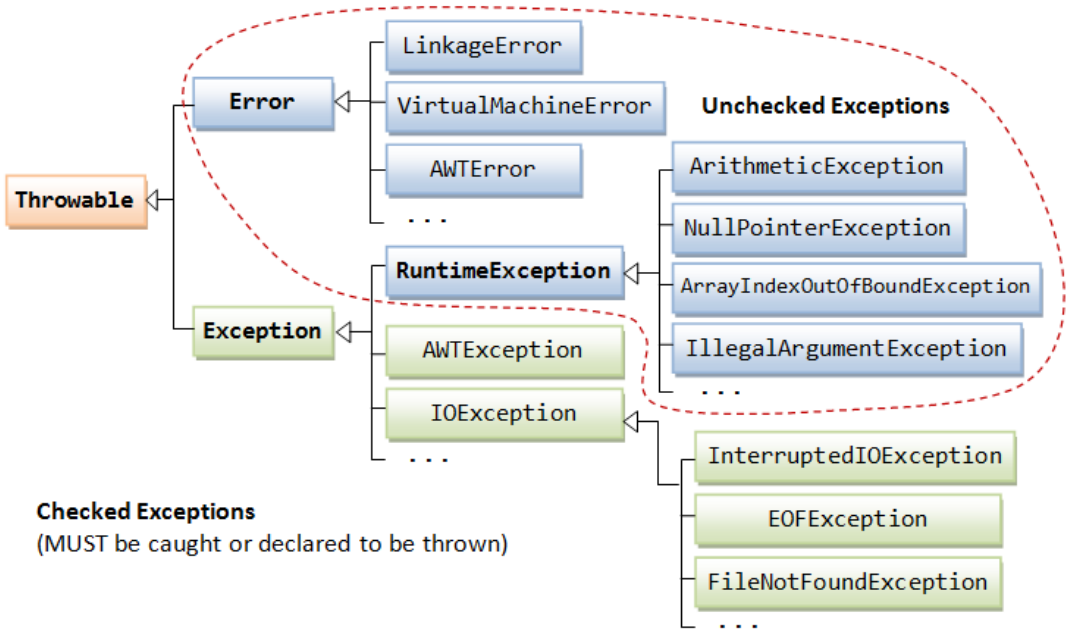
- 예외의 종류
  - 일반 예외(컴파일 체크 Exception)
    - 컴파일하는 과정에서 예외 처리 코드가 필요한지 검사
    - 예외 처리 코드 없으면 컴파일 오류 발생
  - **실행 예외(RuntimeException)**
    - 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
    - '경험'따라 예외 처리 코드 작성 필요

예외처리(Exception Handling)

- 예외 처리(Exception Handling)
  - 런타임 중에 발생할 수 있는 예외들을 처리하는 코드를 프로그래밍 하는 것
  - 자바에서는 예외 처리 코드가 없으면 컴파일러가 오류를 발생시키는 경우가 있음
    - 반대로 예외 처리 코드가 없어도 컴파일러가 오류를 발생시키지 않고, 예외가 발생하면 강제로 프로그램 실행을 중단시키는 경우도 있음

예외 클래스

- 예외 클래스
  - **Java는 예외(Exception)를 클래스로 관리**
  - 자바의 에러나 예외 클래스들은 Throwable의 자손
    - Throwable에는 printStackTrace() 가 있음
  - JVM이 프로그램을 실행하는 도중에 예외가 발생하면 해당 예외 클래스로 객체를 생성
    - 예외 처리코드에서 예외 객체를 이용



예외의 구분

구분	설명	예시
에러(Error)	프로그램에서 발생하는 오류	OutOfMemoryError
실행예외 (Unchecked Exception)	**런타임 예외(RuntimeException).** 발생할 것 같은 예외처리를 해주지 않아서 실행 시 발생하는 예외. 컴파일러가 예외 처리 코드 유무를 확인하지 않음.	NullPointerException
일반 예외 (Checked Exception)	런타임 예외를 제외한 나머지 **일반 예외.** 컴파일러가 예외 처리 코드 유무를 확인함.	IOException SQLException FileNotFoundException

일반 예외(Checked Exception)

- 대표적으로 파일 입출력은 아래와 같이 예외 처리 코드가 없으면 컴파일 오류를 발생 시킴

```
public static void main(String[] args) {
      FileReader fr = new FileReader("noname.txt"); // compile error
      fr.close();
}
```

- 예외 처리 코드 삽입

```
public static void main(String[] args) {
   try {
     FileReader fr = new FileReader("noname.txt");
     fr.close();
   }
   catch (IOException e) {
     System.out.println("noname.txt 파일이 존재하지 않습니다.");
   }
}
```

실행 예외(Unchecked Exception)

- 다음 예외들은 예외 처리 코드가 없어도 컴파일러에서 강요하지 않음

예외	설명
ArithmeticException	정수를 0으로 나눌 때 연산 예외 발생
ArrayIndexOutOfBoundsException	배열 인덱스 범위를 초과할 때 예외 발생
ClassCastException	변환할 수 없는 클래스로 형변환 할 때 예외 발생
NullPointerException	null 객체 참조할 때 예외 발생
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생
IllegalArgumentException	잘못된 인자(argument) 전달 시 발생
InputMismatchException	잘못된 입력일 때 발생

실행 예외(Runtime Exception)

- **NullPointerException**
  - 객체 참조가 없는 상태
    - null 값 갖는 참조변수로 객체 접근 연산자인 도트(.) 사용했을 때 발생

```
String data = null;
System.out.println(data.toString());
```

- **ArrayIndexOutOfBoundsException**
  - 배열에서 인덱스 범위 초과하여 사용할 경우 발생

```
public static void main(String[] args) {
   String data1 = args[0];
   String data2 = args[1];

   System.out.println("args[0]: " + data1);
   System.out.println("args[1]: " + data2);
}
```

실행시 매개값을 주지 않을 경우
예외 발생

실행 예외(RuntimeException)

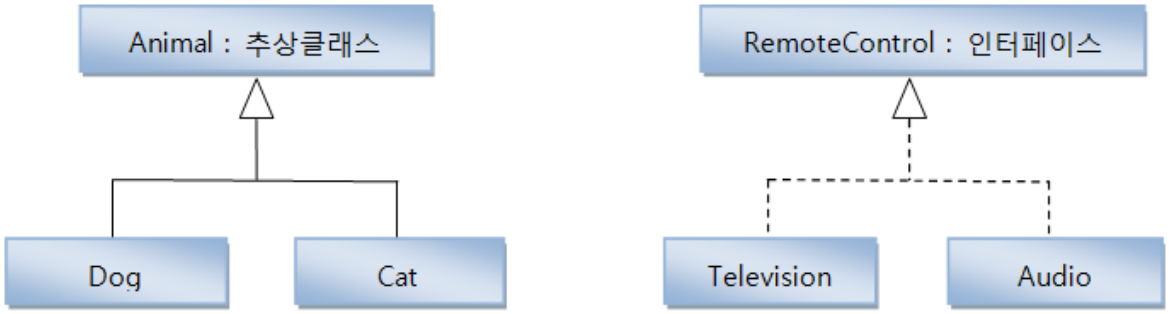
- **NumberFormatException**
  - 숫자로 변환될 수 없는 문자가 포함되어 있는 문자열을 숫자로 변경할 경우
    - Integer.parseInt(String s)
    - Double.parseDouble(String s)

```
public class NumberFormatExceptionExample {
	public static void main(String[] args) {
		String data1 = "100";
		String data2 = "a100";

		int value1 = Integer.parseInt(data1);
		int value2 = Integer.parseInt(data2);

		int result = value1 + value2;
		System.out.println(data1 + "+" + data2 + "=" + result);
	}
}
```

실행 예외(RuntimeException)

ClassCastException

- 타입 변환이 되지 않을 경우 발생



- 정상 코드

```
Animal animal = new Dog();
Dog dog = (Dog) animal;
```

```
RemoteControl rc = new Television();
Television tv = (Television) rc;
```

- 예외 발생 코드

```
Animal animal = new Dog();
Cat cat = (Cat) animal;
```

```
RemoteControl rc = new Television();
Audio audio = (Audio) rc;
```

실행 예외(RuntimeException)

- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생

```
public class NumberFormatExceptionExample {
	public static void main(String[] args) {
		Dog dog = new Dog();
		changeDog(dog);
		Cat cat = new Cat();
		changeDog(cat);
	}
	public static void changeDog(Animal animal) {
		//if(animal instanceof Dog) {
			Dog dog = (Dog) animal;	// ClassCastException 발생 가능
		//}
	}
}
class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}
```

예제 : ArithmeticException 예외 처리

try-catch문을 이용하여 정수를 0으로 나누려고 할 때 "0으로 나룰 수 없습니다."라는 경고 메시지를 출력하도록 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ExceptionExample2 {
   public static void main (String[] args) {
      Scanner rd = new Scanner(System.in);
      int divisor = 0;
      int dividend = 0;
      System.out.print("나뉨수를 입력하시오:");
      dividend = rd.nextInt();
      System.out.print("나눗수를 입력하시오:");
      divisor = rd.nextInt();
      try {
         System.out.println(dividend+"를 "+divisor+"로 나누면 몫은 "+
dividend/divisor+"입니다.");
      } catch (ArithmeticException e) {
         System.out.println("0으로 나눌 수 없습니다.");
      }
   }
}
```

ArithmeticException 예외 발생

```
나뉨수를 입력하시오:100
나눗수를 입력하시오:0
0으로 나눌 수 없습니다.
```

예제 : 범위를 벗어난 배열의 접근

배열의 인덱스가 범위를 벗어날 때 발생하는
ArrayIndexOutOfBoundsException을 처리하는 프로그램을 작성하시오.

```
public class ArrayException {
  public static void main (String[] args) {
    int[] intArray = new int[5];
    intArray[0] = 0;
    try {
      for (int i = 0; i < 5; i++) {
        intArray[i+1] = i+1 + intArray[i];
        System.out.println("intArray["+i+"]"+"="+intArray[i]);
      }
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("배열의 인덱스가 범위를 벗어났습니다.");
    }
  }
}
```

i가 4일 때
ArrayIndexOutOfBoundsException
예외 발생

```
intArray[0]=0
intArray[1]=1
intArray[2]=3
intArray[3]=6
배열의 인덱스가 범위를 벗어났습니다.
```

예제 : 정수가 아닌 문자열을 정수로 변환할 때 예외 발생

문자열을 정수로 변환할 때 발생하는 NumberFormatException을 처리하는 프로그램을 작성하라.

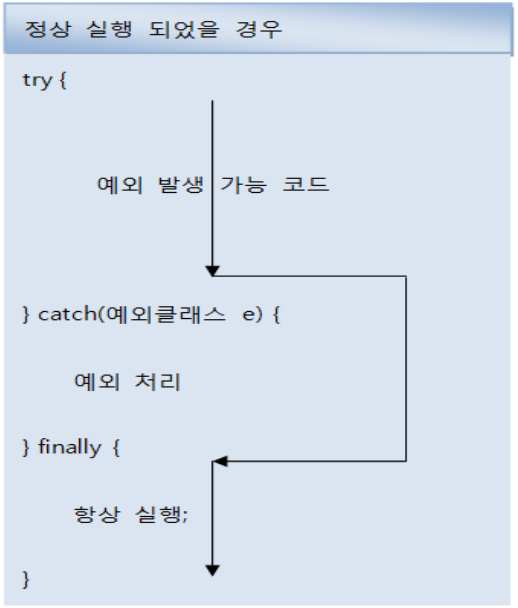
```
public class NumException {
   public static void main (String[] args) {
      String[] stringNumber = {"23", "12", "998", "3.141592"};
      try {
         for (int i = 0; i < stringNumber.length; i++) {
            int j = Integer.parseInt(stringNumber[i]);
            System.out.println("숫자로 변환된 값은 " + j);
         }
      }
      catch (NumberFormatException e) {
         System.out.println("정수로 변환할 수 없습니다.");
      }
   }
}
```

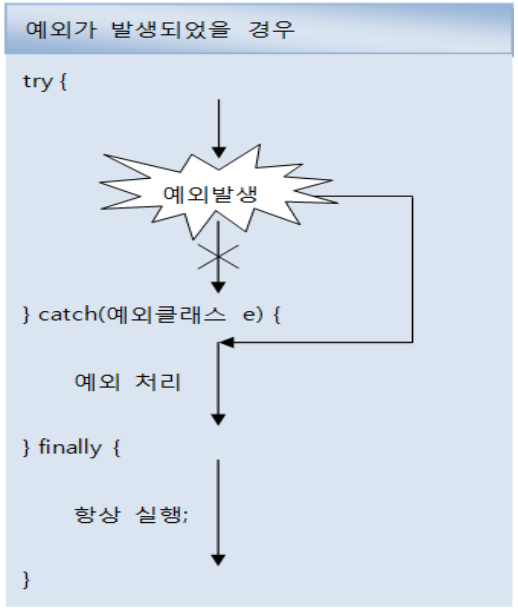
"3.141592"를 정수로 변환할 때 NumberFormatException 예외 발생

```
숫자로 변환된 값은 23
숫자로 변환된 값은 12
숫자로 변환된 값은 998
정수로 변환할 수 없습니다.
```

예외처리 코드 (try-catch-finally)

- 예외처리 코드
  - 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
    - 일반 예외: 반드시 작성해야 컴파일 가능
    - 실행 예외: 컴파일러가 체크해주지 않으며 개발자 경험 의해 작성
  - **try – catch – finally** 블록 이용해 예외 처리 코드 작성





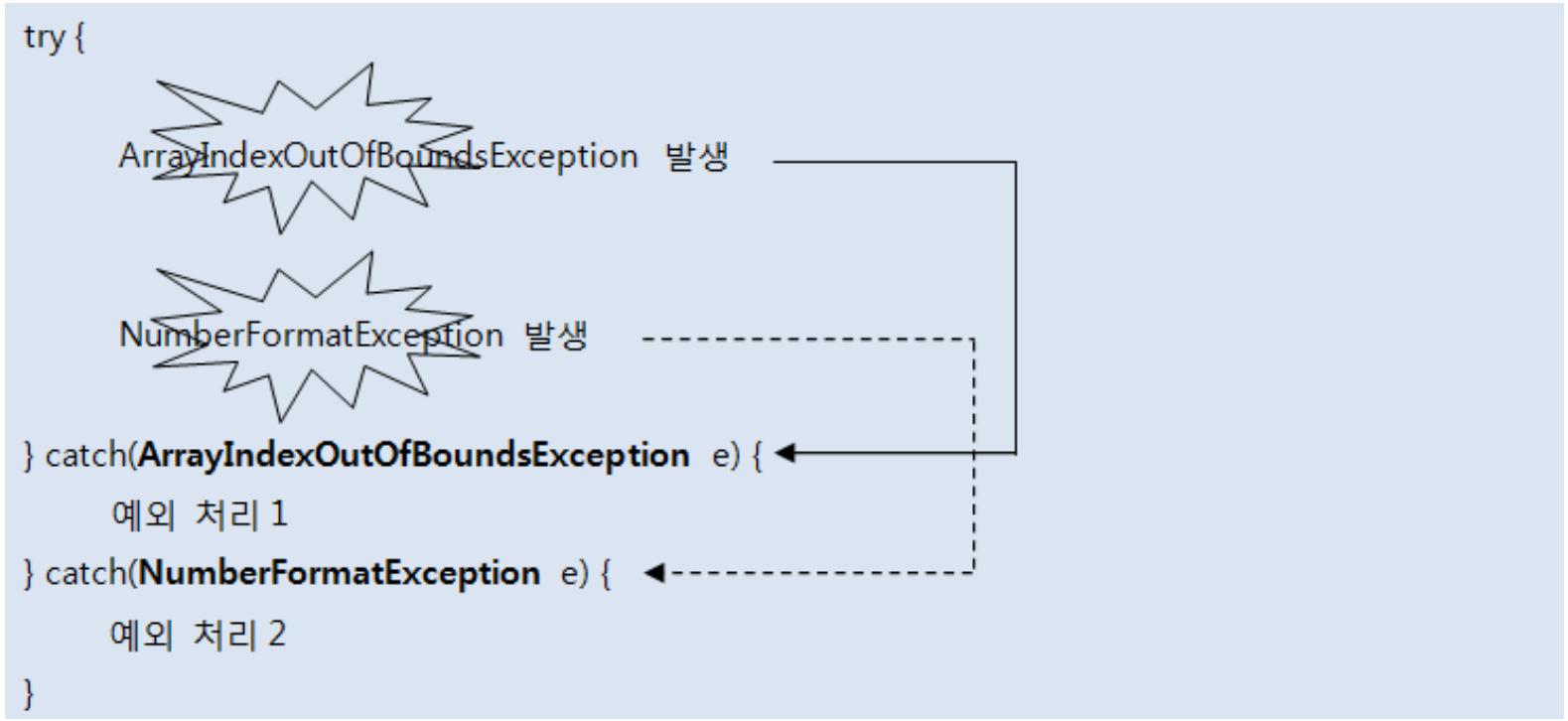
예외 처리 코드(try-catch-finally)

```
public class TryCatchFinallyRuntimeExceptionExample {
    public static void main(String[] args) {
        String data1 = null;
        String data2 = null;
        try {                                          실행시 매개값을 주지 않을 경우 예외 발생
            data1 = args[0];
            data2 = args[1];
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java TryCatchFinallyRuntimeExceptionExample  num1  num2");
            return;
        }
        try {
            int value1 = Integer.parseInt(data1);  실행시 매개값을 잘못 주었을 경우 예외 발생
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

예외 종류에 따른 처리 코드

다중 catch

- 하나의 try 블록 내부에서 다양한 종류의 예외 발생시
- 각 예외 별로 예외 처리 코드(catch 블록) 다르게 구현
- 단 하나의 catch 블록만 실행



예외 종류에 따른 처리 코드

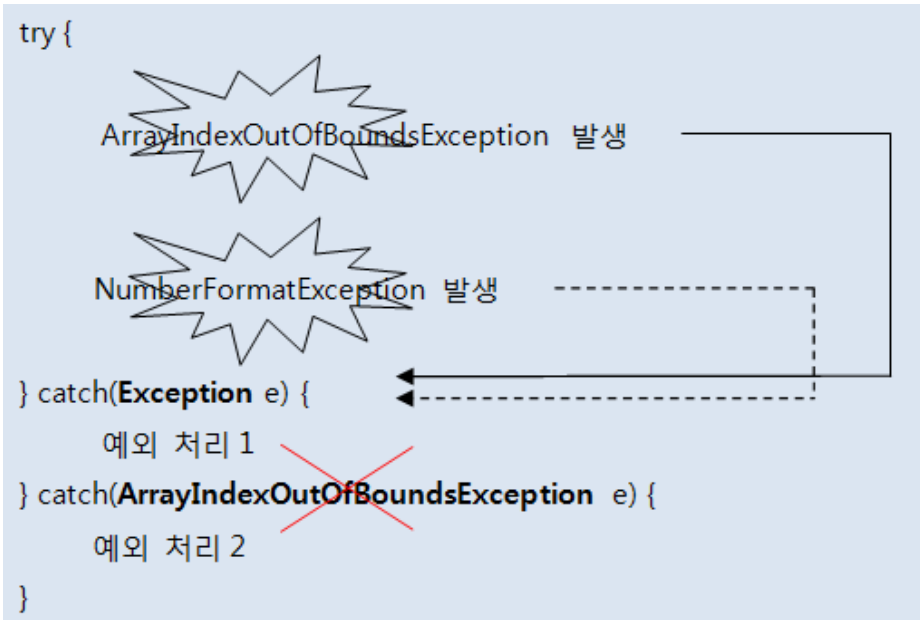
```
public class CatchByExceptionKindExample {
	public static void main(String[] args) {
		try {
			String data1 = args[0];
			String data2 = args[1];
			int value1 = Integer.parseInt(data1);
			int value2 = Integer.parseInt(data2);
			int result = value1 + value2;
			System.out.println(data1 + "+" + data2 + "=" + result);
		} catch(ArrayIndexOutOfBoundsException e) {
			System.out.println("실행 매개값의 수가 부족합니다.");
			System.out.println("[실행 방법]");
			System.out.println("java CatchByExceptionKindExample  num1  num2");
		} catch(NumberFormatException e) {
			System.out.println("숫자로 변환할 수 없습니다.");
		} finally {
			System.out.println("다시 실행하세요.");
		}
	}
}
```

실행시 매개값을 주지 않을 경우 예외 발생

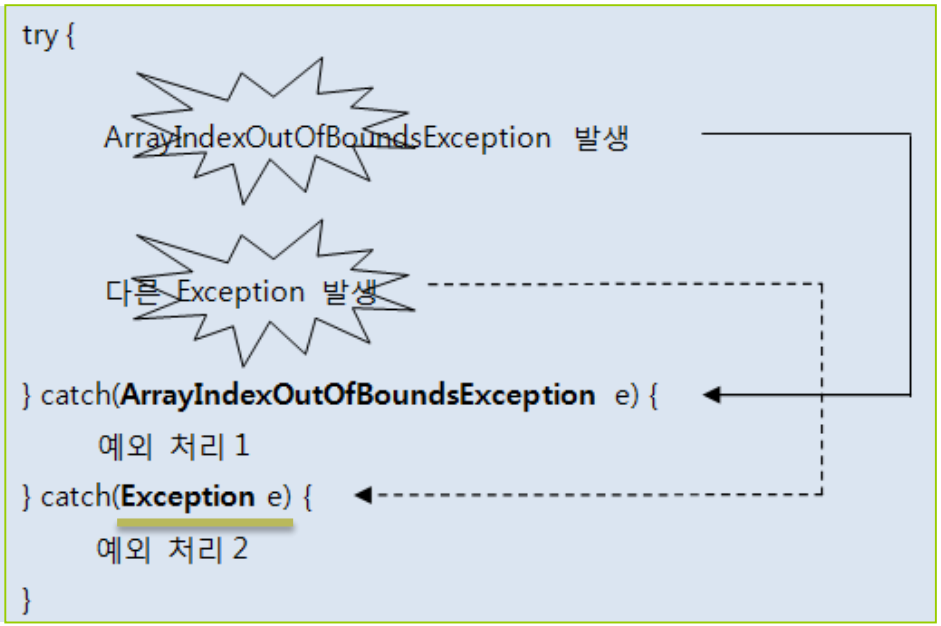
실행시 매개값을 잘못 주었을 경우 예외 발생

예외 종류에 따른 처리 코드

- 하위 예외는 상위 예외를 상속
  - 하위 예외는 상위 예외 타입도 됨
- **catch** 순서 – 상위 예외가 아래에 위치해야



상위 예외 Exception이 위에 있을 경우



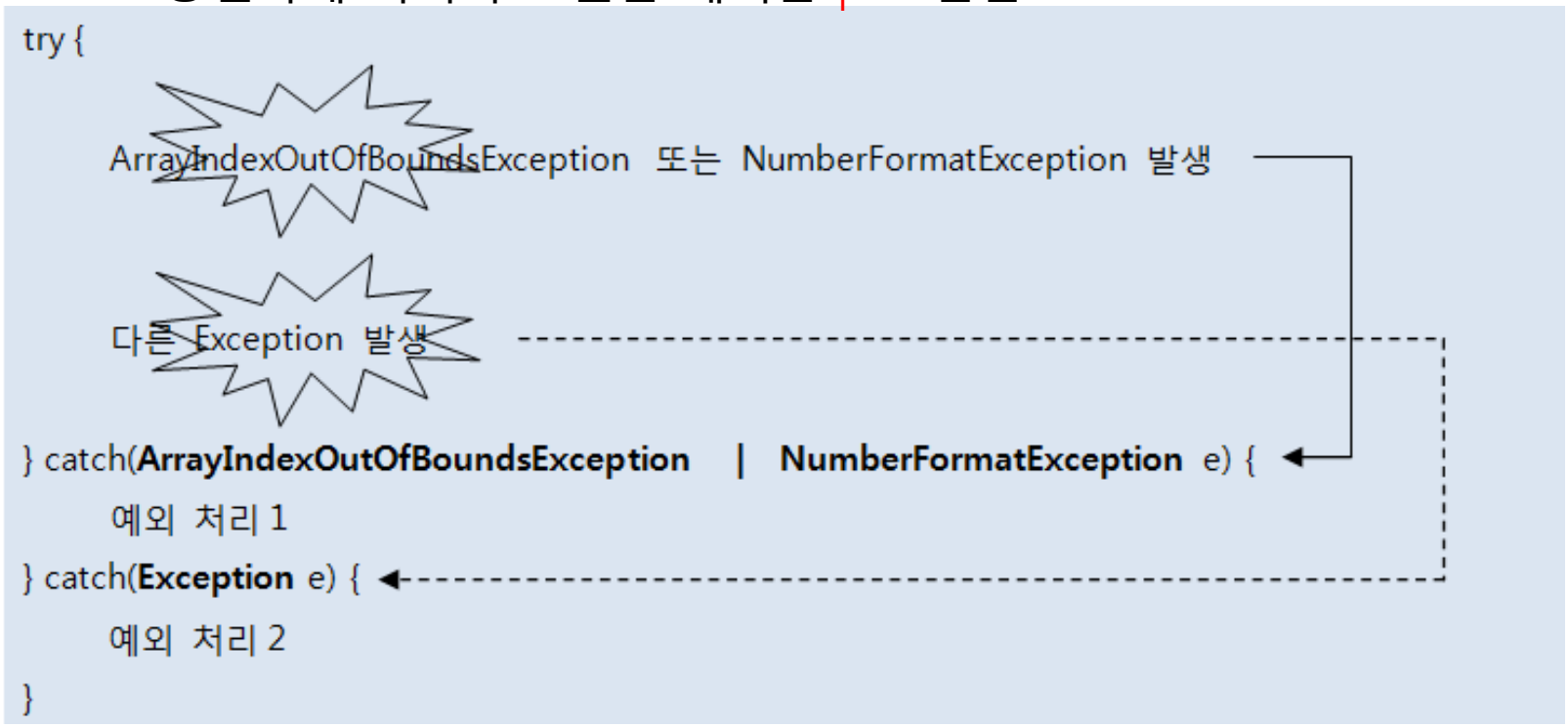
상위 예외 Exception이 아래에 있을 경우

예외 종류에 따른 처리 코드

```
public class CatchOrderExample {
	public static void main(String[] args) {
		try {
			String data1 = args[0];
			String data2 = args[1];
			int value1 = Integer.parseInt(data1);
			int value2 = Integer.parseInt(data2);
			int result = value1 + value2;
			System.out.println(data1 + "+" + data2 + "=" + result);
		} catch(ArrayIndexOutOfBoundsException e) {
			System.out.println("실행 매개값의 수가 부족합니다.");
		} catch(Exception e) {
			System.out.println("실행에 문제가 있습니다.");
		} finally {
			System.out.println("다시 실행하세요.");
		}
	}
}
```

예외 종류에 따른 처리 코드

- **멀티(multi) catch**
  - 자바 7이후 하나의 catch 블록에서 여러 개의 예외 처리 가능
    - 동일하게 처리하고 싶은 예외를 | 로 연결



예외 종류에 따른 처리 코드

```
public class MultiCatchExample {
	public static void main(String[] args) {
		try {
			String data1 = args[0];
			String data2 = args[1];
			int value1 = Integer.parseInt(data1);
			int value2 = Integer.parseInt(data2);
			int result = value1 + value2;
			System.out.println(data1 + "+" + data2 + "=" + result);
		} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
			System.out.println("실행 매개값의 수가 부족하거나 숫자로 변환할 수
없습니다..");
		} catch(Exception e) {
			System.out.println("알수 없은 예외 발생");
		} finally {
			System.out.println("다시 실행하세요.");
		}
	}
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

자동 리소스 닫기

- try-with-resources
  - 예외 발생 여부와 상관 없음
  - 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
  - 리소스 객체
    - 각종 입출력스트림, 서버소켓, 소켓, 각종 채널
    - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

자동 리소스 닫기

```
public static void main(String[] args) {
    FileInputStream fis = null;
    BufferedInputStream bis= null;
    // Try-with-resource
    try (is = new FileInputStream("test.txt");   bis = new BufferedInputStream(is)) {
        int data = -1;
        while((data = bis.read()) != -1) {
            System.out.println((char)data);
        }
    } catch (IOException e) {
        e.printStackTrack();
    }
    // try-with-resources는 try(...) 안에서 선언된 객체의 close() 메소드들을 호출
    // 그래서 finally에서 close()를 명시적으로 호출해줄 필요가 없음
    // Try-with-resources가 모든 객체의 close()를 호출해주지는 않음
    // AutoCloseable을 구현한 객체만 close()가 호출됨
}
```

예외 떠 넘기기(throws)

- **throws**
  - 메소드 선언부 끝에 작성
  - 메소드내에서 처리하지 않은 예외를 메소드 호출한 곳(calling method)으로 떠 넘기는 역할
- 예외를 위임
  - 예외를 발생시키는 코드

```
void printElement(int[] arr, int index) {
    System.out.print(arr[index]);
}
```

  - 예외를 위임시키는 코드

```
void printElement(int[] arr, int index) throws ArrayIndexOutOfBoundsException {
    System.out.print(arr[index]);
}
```

예외 떠 넘기기(throws)

- 예외를 변환해서 전달
  - 예외를 위임시키는데 다른 예외로 바꿔서 전달

```
void printElement2(int[] arr, int index) throws IllegalArgumentException {
    try {
        System.out.print(arr[index]);
    }
    catch (ArrayIndexOutOfBoundsException e) {
        // 새로운 예외로 변경해서 발생시킴
        throw new IllegalArgumentException();
    }
}
```

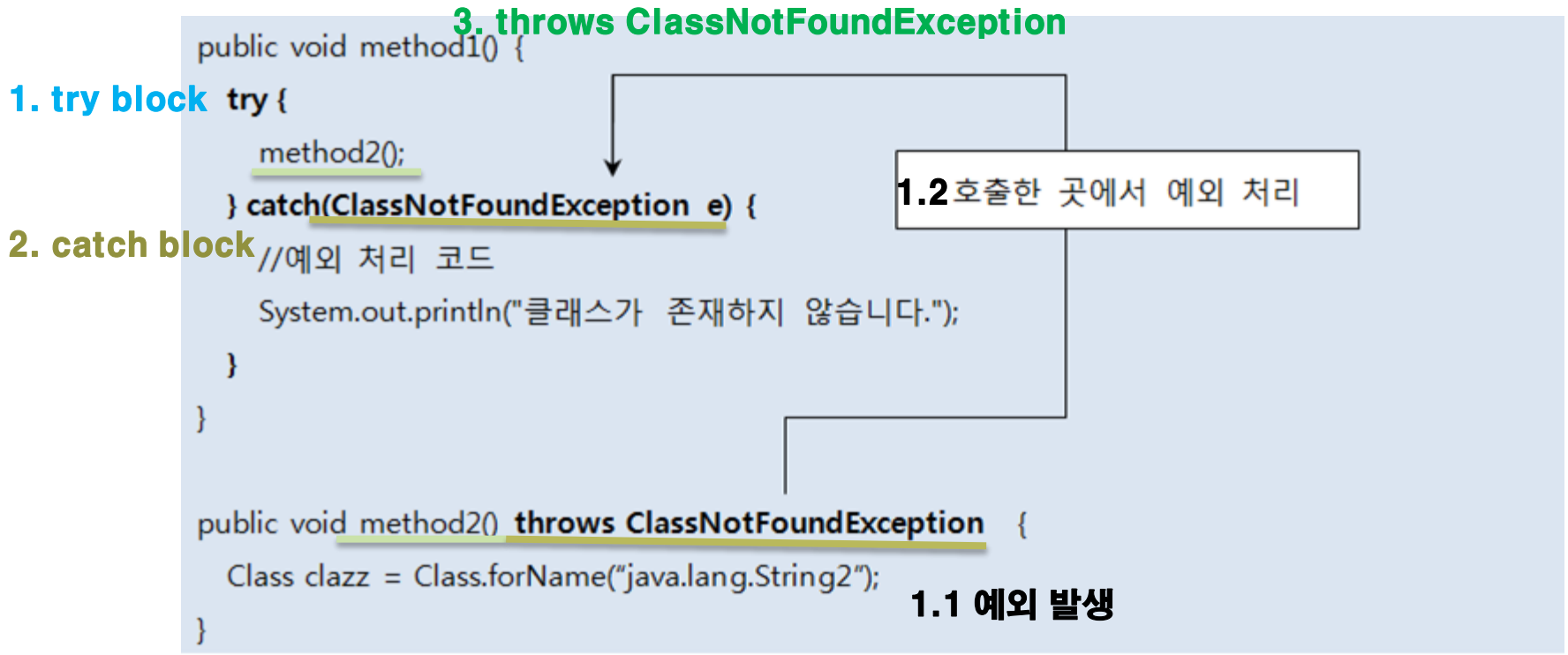
예외 떠 넘기기(throws)

- throws 선언된 메소드를 호출하는 메소드에서 try/catch 사용해야 함

```
public static void main(String[] args) {
    try {
        printElement(args, 5);
    }
    catch (ArrayIndexOutOfBoundsException e) {
        e.printStackTrace();
    }
   try {
        printElement2(args, 5);
    }
    catch (IllegalArgumentException e) {
        e.printStackTrace();
    }
}
```

예외 떠 넘기기(throws)

- throws 선언된 메소드를 호출하는 메소드(calling method)는
  1. 반드시 try 블록 내에서 호출
  2. catch 블록에서 떠 넘겨 받은 예외를 처리함
  3. try-catch 블록으로 예외처리를 하지 않고 throws 키워드로 자신도 다시 예외를 떠 넘길 수 있음



사용자 정의 예외와 예외 발생

- 사용자 정의 예외(user-defined exception) 클래스 선언
  - 자바 표준 API에서 제공하지 않는 예외
  - 애플리케이션 서비스와 관련된 예외, Application Exception
    - E.g. 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
  - 사용자 정의 예외 클래스 선언 방법
    1. 예외 클래스 상속
       - 일반 예외 : Exception class 상속
       - 실행 예외 : RuntimeException class 상속
    2. 생성자 정의
       - 매개변수 없는 기본 생성자, String 타입의 매개변수를 갖는 생성자

```
public class XXXException extends [ Exception | RuntimeException ] {
    public XXXException() { }
    public XXXException(String message) { super(message); }
}
```

사용자 정의 예외와 예외 발생

- 예외 발생 시키기(throw)
  - 코드에서 예외 발생시키는 법 - 예외 객체 생성

```
throw new XXXException()
throw new XXXException("메시지");
```

  - 호출한 곳에서 발생한 예외를 처리하도록

```
public void method() throws XXXException {
    throw new XXXException("메시지");
}
```

사용자 정의 예외와 예외 발생

```
public class BalanceInsufficientException extends Exception {
        public BalanceInsufficientException() { }
        public BalanceInsufficientException(String message) {
                super(message);
        }
}
```

1. Exception class 상속

2. constrctors 생성

```
public class Account {
        private long balance;

        public Account() { }

        public long getBalance() {
                return balance;
        }
        public void deposit(int money) {
                balance += money;
        }
        public void withdraw(int money) throws BalanceInsufficientException {
                if(balance < money) {
                        throw new BalanceInsufficientException("잔고부족:"+(money-balance)+" 모자람");
                }
                balance -= money;
        }
}
```

4. 호출한 곳에서 발생한 예외를 처리하도록 함

3. 사용자 정의 예외 발생 – 예외 객체 생성

사용자 정의 예외와 예외 발생

```
public class AccountExample {
	public static void main(String[] args) {

		Account account = new Account();

		//예금하기
		account.deposit(10000);
		System.out.println("예금액: " + account.getBalance());

		//출금하기
		try {
			account.withdraw(30000);
		} catch(BalanceInsufficientException e) {
			String message = e.getMessage();
			System.out.println(message);
			System.out.println();
			e.printStackTrace();
		}
	}
}
```

5. try block에서 메소드 호출

6. catch block에서 사용자정의 예외 처리

7. 예외 정보 얻기

예외 정보 얻기

- **getMessage()**
  - 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

```
throw new XXXException("예외  메시지");
```

  - 원인 세분화하기 위해 예외 코드 포함(예: 데이터베이스 예외 코드)

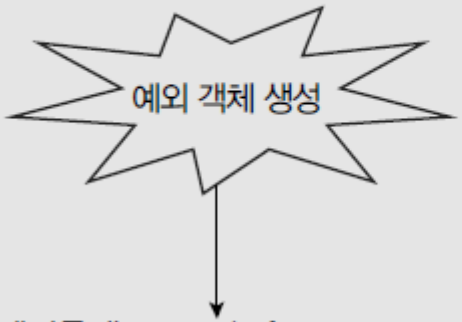
  - catch() 절에서 활용

```
} catch(Exception  e) {
    String message  = e.getMessage();
}
```

예외 정보 얻기

- **printStackTrace()**
  - 예외 발생 코드를 추적하여 모두 콘솔에 출력

```
try {
```




```
} catch(예외클래스    e) {
    //예외가 가지고 있는 Message 얻기
    String message = e.getMessage();

    //예외의 발생 경로를 추적
    e.printStackTrace();
}
```