11
자료구조 2

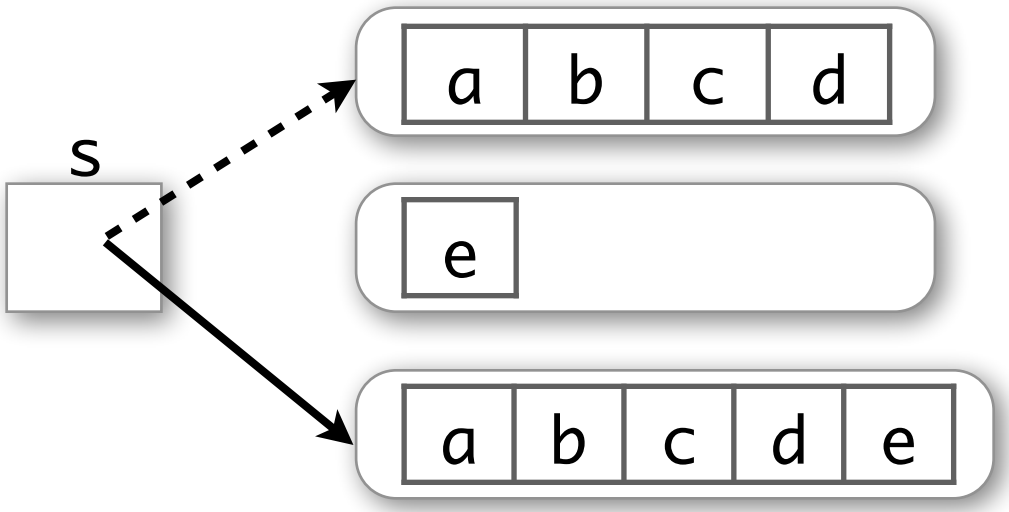
강의 내용

- 데이터를 저장 및 관리하기 위한 다양한 자료구조
  - 문자열, 문서, 파일
- 예외 처리

문자열(String)은 변경불가

- 문자열은 한 번 만들어지면 변경할 수 없다.
- 연산을 적용하면 새로운 문자열을 만드는 것이다.
- 변경을 원한다면 java.lang.StringBuffer 를 사용하라.

```
String s = "abcd";
s = s + "e";
```



문자열 장난 첫 번째, 토큰으로 나누기

- 많은 경우 문자열을 의미 있는 것들로 나누고 싶을 때가 많다.
  - 예, "910359,이욱세,남"

    ➜ "910359" "이욱세" "남"

- 토큰 (token) 또는 어휘 (token)
  - 의미 있는 단어, 즉, 문자열
  - 분리자(delimiter)로 토큰이 나열된 자료가 많다.

java.util.StringTokenizer

- 문자열 토큰화 라이브러리

```
String s = "910359,이욱세,남";
StringTokenizer t = new StringTokenizer(s, ",");
t.nextToken();  // "910359"
t.nextToken();  // "이욱세"
t.nextToken();  // "남"
t.nextToken();  // NoSuchElementException
```

분리자

- 분리자를 여러 개 지정 가능

```
String s = "$13.46";
StringTokenizer t = new StringTokenizer(s, "$.");
t.nextToken();  // "13"
t.nextToken();  // "46"
t.nextToken();  // NoSuchElementException
```

다른 유용한 메소드

class StringTokenizer	
생성자	
new StringTokenizer (String text, String delim)	text로부터 분리자들 delim으로 문자열 토큰화기 생성
메소드	
nextToken(): String	문자열을 보고 분리자들을 지우고, 분리자가 포함되지 않은 길이가 0초과인 가장 긴 문자열을 찾아 지우고 결과로 반환
nextToken(String new_delimiters): String	nextToken()과 같으나 분리자를 새로 지정
hasMoreTokens(): boolean	토큰이 남았는지 여부 반환
countTokens(): int	토큰이 몇 개 남았는지 반환

파일 (Files)

- 파일(file): 컴퓨터의 보조 기억장치에 저장된 기호의 나열
  - 문자 파일 (character file): 문자들의 나열
  - 바이너리 파일 (binary file): 0과 1의 나열

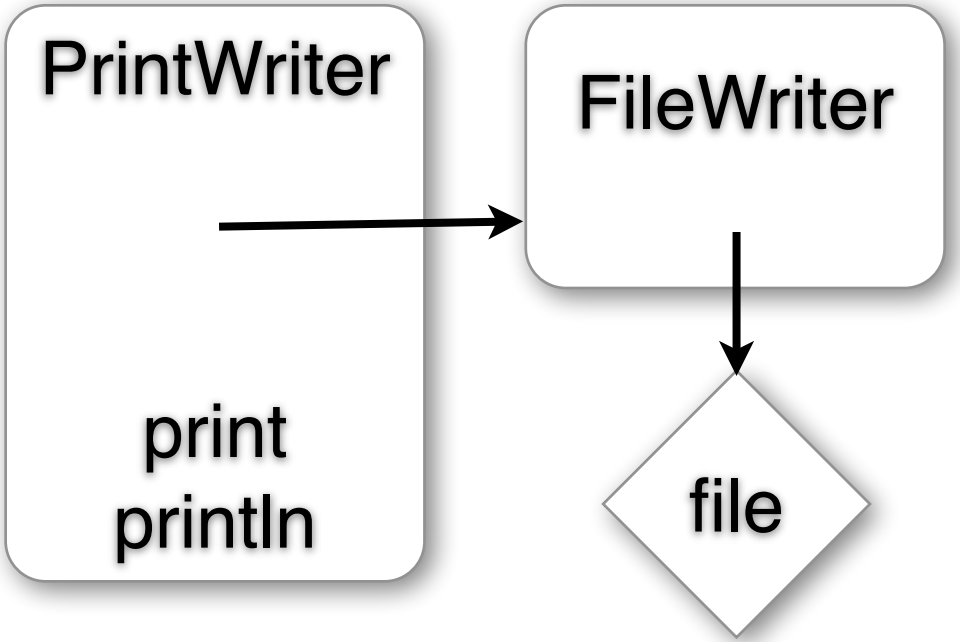
- 파일을 사용하려면
  - 열어서(open) 보거나 쓰고 다 사용했으면 닫아야(close)한다.

- 파일을 열 때 보기만 할 건지 쓰기만 할 건지 다 할 건지 지정해야 한다.
  - 입력 (input) 파일: 보기만 하는 파일
  - 출력 (output) 파일: 쓰기만 하는 파일

파일 출력 객체 만들기

- FileWriter 객체
  - 파일의 주소 보유
  - 파일에 쓰기 메소드 보유
  - 생성시 파일이 쓰기 모드로 열린다. 없으면 생성.
- PrintWriter 객체
  - FileWriter 객체 보유
  - print, println을 포함해 여러 메소드 제공

```
PrintWriter ofile =
  new PrintWriter
    (new FileWriter("file.txt"));
```



예제, 간단한 출력

IO 라이브러리 사용

IOException이 발생할 수 있다.

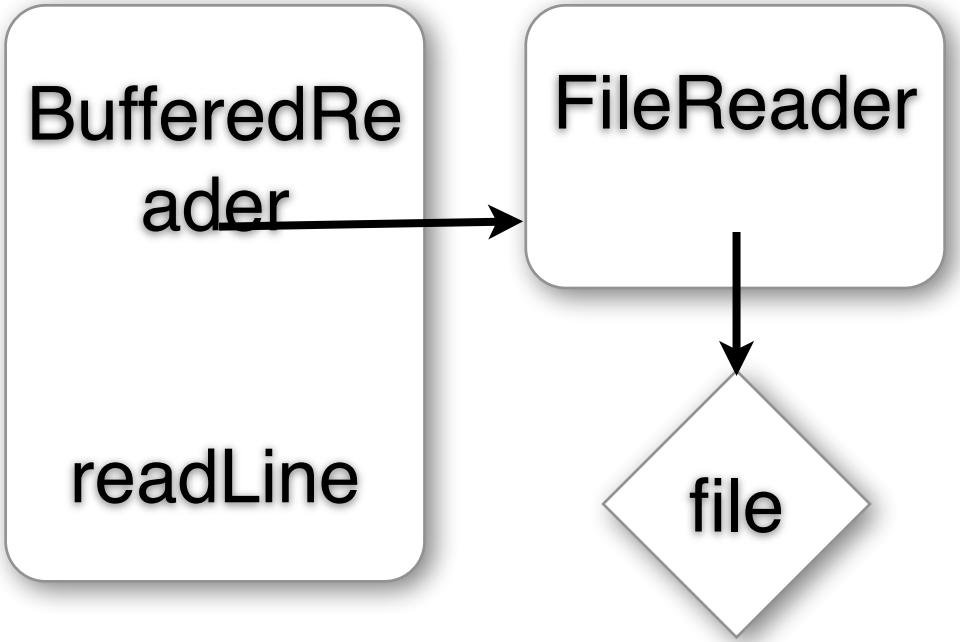
```
import java.io.*;

public class Output1 {
    public static void main(String[] args) throws IOException {
        PrintWriter outfile =
      new PrintWriter(new FileWriter("test.txt"));
        outfile.println("Hello to you!");
        outfile.print("How are");
        outfile.println(" you?");
        outfile.println(47+2);
        outfile.close();
    }
}
```

파일 입력 객체 만들기

- FileReader 객체
  - 파일의 주소 보유
  - 파일 읽기 메소드 보유
  - 생성시 파일이 읽기 모드로 열린다. 없으면 예외 발생.
- BufferedReader 객체
  - FileReader 객체 보유
  - readLine을 포함해 여러 메소드 제공

```
BufferedReader ifile =
  new BufferedReader
    (new FileReader("file.txt"));
```

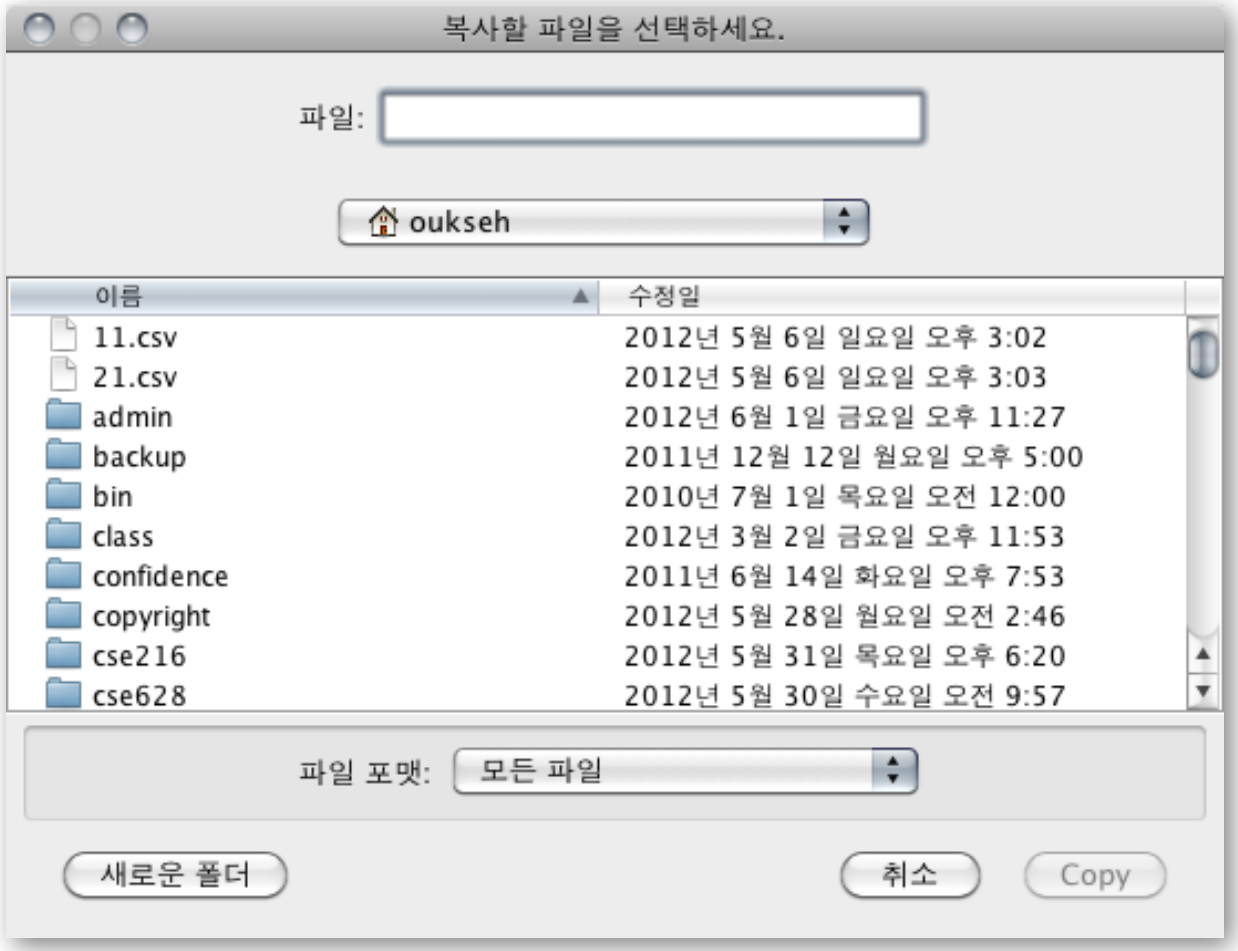


예제, 파일 복사

```
import java.io.*; import javax.swing.*;

public class CopyFile {
    public static void main(String[] args) throws IOException {
        String f = JOptionPane.showInputDialog("Input filename, please: ");
        BufferedReader infile = new BufferedReader(new FileReader(f));
        PrintWriter outfile = new PrintWriter(new FileWriter(f + ".out"));
        while (infile.ready()) {
            outfile.println(infile.readLine());
        }
        infile.close();
        outfile.close();
    }
}
```

파일 선택 GUI


복사할 파일을 선택하세요.
파일:
oukseh
이름
수정일
11.csv
2012년 5월 6일 일요일 오후 3:02
21.csv
2012년 5월 6일 일요일 오후 3:03
admin
2012년 6월 1일 금요일 오후 11:27
backup
2011년 12월 12일 월요일 오후 5:00
bin
2010년 7월 1일 목요일 오전 12:00
class
2012년 3월 2일 금요일 오후 11:53
confidence
2011년 6월 14일 화요일 오후 7:53
copyright
2012년 5월 28일 월요일 오전 2:46
cse216
2012년 5월 31일 목요일 오후 6:20
cse628
2012년 5월 30일 수요일 오전 9:57
파일 포맷:
모든 파일
새로운 폴더
취소
Copy

예, 파일 선택해서 복사

```
import java.io.*; import javax.swing.*;

public class CopyFile {
    public static void main(String[] args) throws IOException {
        JFileChooser chooser = new JFileChooser();
        chooser.setDialogTitle("복사할 파일을 선택하세요.");
        int result = chooser.showDialog(null, "Copy");
        if(result != JFileChooser.APPROVE_OPTION)
            System.exit(0);
        String f = chooser.getSelectedFile().toString();
        BufferedReader infile = new BufferedReader(new FileReader(f));
        PrintWriter outfile = new PrintWriter(new FileWriter(f + ".out"));
        while (infile.ready()) {
            outfile.println(infile.readLine());
        }
        infile.close();
        outfile.close();
    }
}
```

특별한 녀석들: 표준 입출력

- System.out: 표준 출력
  - System.out.println
- System.in: 표준입력
  - 불행히도 System.in은 InputStream 클래스로 한 번에 한 문자만 입력받는 메소드 정도 밖에 없다.
  - System.in을 편하게 쓰려면 BufferedReader로 변환 필요

```
BufferedReader keyboard =
  new BufferedReader(new InputStreamReader(System.in));
String s = keyboard.readLine();
```

예제, 급여 처리

```
이순신|31|20250
이욱세|42|24500
홍길동|18|18000
!
```

```
이순신|627750
이욱세|1029000
홍길동|324000
!
```

- 입력 파일을 읽는 PayrollReader 클래스를 작성하라.
  - 이름, 일한시간, 시급
- 출력 파일을 쓰는 PayrollWriter 클래스를 작성하라.
  - 이름, 급여
- 입력 파일로부터 출력파일을 만들어라.

명세

class PayrollReader	
Methods	
getNextRecord(): boolean	파일에서 다음 레코드를 읽는다. 제대로 읽었으면 참을 아니면 거짓을 반환한다.
nameOf(): String	현재 레코드의 고용인 이름을 반환한다.
hoursOf(): int	현재 레코드의 고용인이 일한 시간을 반환한다.
payrateOf(): double	현재 레코드의 고용인의 시급을 반환한다.
close()	파일을 닫는다.

구현

```
import java.io.*; import java.util.*;

public class PayrollReader {
    private BufferedReader infile;
    private String EOF = "!";
    private String name;
    private int hours, payrate;

    public PayrollReader(String file_name) {
        infile = new BufferedReader(new FileReader(file_name));
    }
    public String nameOf() { return name; }
    public int hoursOf() { return hours; }
    public int payrateOf() { return payrate; }
    public void close() { infile.close(); }
```

구현

```
    public boolean getNextRecord() {
        if(!infile.ready()) return false;
        String line = infile.readLine();
        StringTokenizer t = new StringTokenizer(line, "|");
        String s = t.nextToken().trim();
        if(s.equals(EOF) || t.countTokens() != 2) return false;
        name = s;
        hours = new Integer(t.nextToken().trim()).intValue();
        payrate = new Integer(t.nextToken().trim()).intValue();
        return true;
    }
}
```

제어기

```
import javax.swing.*;

public class Payroll {
    private static void processPayroll(String in, String out) {
        PayrollReader reader = new PayrollReader(in);
        PayrollWriter writer = new PayrollWriter(out);
        while(reader.getNextRecord()) {
            int pay = reader.hoursOf() * reader.payrateOf();
            writer.printCheck(reader.nameOf(), pay);
        }
        reader.close(); writer.close();
    }
    public static void main(String[] args) {
        String in_name =
      JOptionPane.showInputDialog("Plead type input payroll name: ");
        String out_name =
      JOptionPane.showInputDialog("Plead type output payroll name: ");
        if(in_name != null && out_name != null)
            processPayroll(in_name, out_name);
    }

}
```

안전한 코딩

- getNextRecord에서 문제가 발생하였다면?
  - 파일을 읽지 못한다면?
    - 파일이 열렸는데 시스템 문제로 인해 계속 읽을 수 없을 수도 있다.
  - 레코드가 원하는 형식이 아니면?
    - 급여 파일이 깨졌을 수도 있다.

- 예상되는 위기 대처법
  - 파일에 문제가 있다면 false를 반환
  - 레코드가 원하는 형식이 아니면 다음 레코드를 읽음
  - 어쨌든 사용자가 알 수 있도록 로그(log)는 남겨야 한다!

오류를 어떻게 알 수 있는가?

- infile.readLine() 에서 못 읽으면 IOException 예외 발생
  - 프로그램을 죽일 순 없다.
  - 예외를 잡아서 로그를 남기고 false를 반환해야 한다.

- t.countToken() != 2 이면 레코드 형식 오류
  - 로그를 남기고 다음 줄로 다시 시도

예외 잡기

- try 몸체에서 실행하다가 예외가 발생하면,
  - 그 아래 문장들은 실행되지 않고 밖으로 탈출
- catch 구문에서 예외를 잡는다.
  - 단 제한된 타입의 예외만
  - catch 구문을 실행하고 정상적인 수행을 계속

```
try {
    명령문;
    ...
}
catch (타입 변수) {
    명령문;
    ...
}
```

전혀 안전하지 않은 코딩

```
    public boolean getNextRecord() {
        if(!infile.ready()) return false;
        String line = infile.readLine();
        StringTokenizer t = new StringTokenizer(line, "|");
        String s = t.nextToken();
        if(s.equals(EOF) || t.countTokens() != 2) return false;
        name = s;
        hours = Integer.parseInt(t.nextToken());
        payrate = Integer.parseInt(t.nextToken());
        return true;
    }
}
```

파일을 읽다가 오류난 경우를 잡자

```
public boolean getNextRecord() {
  boolean result = false;
    try {
        if(infile.ready()) return false;
        String line = infile.readLine();
    ...
    if(!s.equals(EOF) && t.countTokens() == 2) {
      ...
      result = true;
    }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
  return result;
}
```

형식이 이상한 경우를 잡자

```
public boolean getNextRecord() {
  boolean result = false;
    try {
        if(infile.ready()) return false;
        String line = infile.readLine();
    ...
    if(!s.equals(EOF)) {
      if(t.countTokens() == 2) {
        ...
        result = true;
      }
      else {
            System.out.println("PayrollReader: bad record format: "
                    + line + "  Skipping record");
            result = getNextRecord();
      }
    }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
  return result;
```

오류 처리를 한 곳에 모으자

```
public boolean getNextRecord() {
  boolean result = false;
    try {
    ...
    if(!s.equals(EOF)) {
      if(t.countTokens() == 2) {
        ...
        result = true;
      }
      else throw new RuntimeException(line);
    }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
    catch (RuntimeException e) {
        System.out.println("PayrollReader error: bad record format: "
                + e.getMessage() + "  Skipping record");
        result = getNextRecord();
    }
  return result;
}
```

예외도 객체다

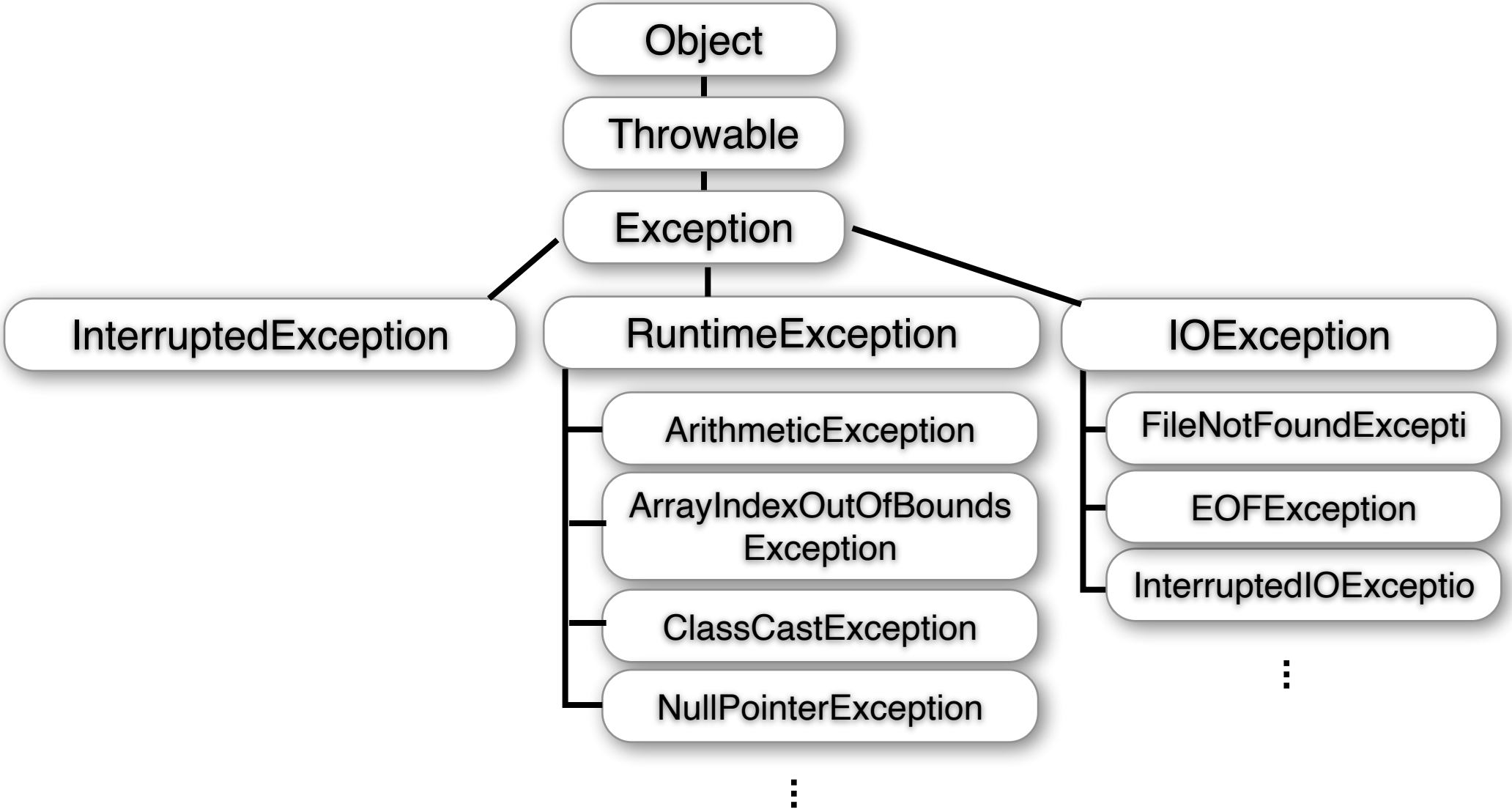
- 예외 객체는 오류에 대한 정보를 가진 객체다.
  - 오류에 대한 설명
  - 오류가 발생한 위치, 추가로 탈출 경로

```
Exception in thread "main" java.io.FileNotFoundException: /Volumes/Pamela/
Users/oukseh/11.csv (Permission denied)
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(FileInputStream.java:120)
    at java.io.FileInputStream.<init>(FileInputStream.java:79)
    at java.io.FileReader.<init>(FileReader.java:41)
    at CopyFile.main(CopyFile.java:11)
```

예외의 메소드

`getMessage(): String`	오류에 대한 정보
`toString(): String`	예외 객체를 문자열로 변환. 일반적으로 예외 클래스 이름과 오류에 대한 정보를 포함
`printStackTrace()`	예외가 언제 발생했고, 어디를 따라 탈출했는지 정보를 콘솔에 출력

예외도 클래스 구조도를 가지고 있다



catch에서 지정하는 예외

- catch(<타입> e) { ... }는
  - <타입>의 하위 타입만 잡는다.
  - Exception으로 하면 모든 예외
  - RuntimeException으로 하면 모든 실행시간 예외
  - IOException으로 하면 파일처리 예외

예, 잘할때까지 정수 입력 받기

```
private int readAnIntFrom(BufferedReader view) throws IOException {
    int num;
    try {
        System.out.print("Type an int: ");
        String s = view.readLine();
        num = Integer.parseInt(s);
    }
    catch (Exception e) {
        System.out.println("Error: " + e.getMessage()
      + " not an integer; try again.");
        num = readAnIntFrom(view); // restart return num;
    }
}
```

Exception 보다는 RuntimeException

RuntimeException보다는 NumberFormatException

throws 절

- 예외도 프로그램을 죽인다.
  - 오류 발생보다는 좋지만 프로그램을 비정상 종료한다는 점에서는 역시 위험한 녀석이다.

- Java의 해결책
  - RuntimeException을 제외한 모든 예외에 대해
  - 모든 메소드마다
  - 발생가능한 예외를 명시해야 한다.
  - 사용자가 인지할 수 있도록.

예외도 잘 써야 보약

```
public class ExceptionExample {
    public ExceptionExample() { }
    public void f() {
        try { g(); }
        catch (RuntimeException e) { System.out.println("caught at f"); }
        System.out.println("f completes");
    }
    public void g()
    {   try {
            PrintWriter outfile = new PrintWriter(new FileWriter("text.out"));
            try { outfile.println( h() ); }
            catch (NullPointerException e)
        { System.out.println("null pointer caught at g"); }
        }
    catch (IOException e)
            { System.out.println("io error caught at g"); }
    System.out.println("g completes");
    }
    private int h() {
        int[] r = new int[2];
        return r[3]; // ArrayIndexOutOfBoundsException!
    }
```

f를 호출하면 무슨 일이 일어날까?

여러 예외를 구분짓고 싶다

- 계산A, 계산B를 하다가 각각 오류가 발생했을 때 처리 루틴을 별도로 작성하고 싶다.
  - `try { ... } catch (?) { ... }`

- 방법1: 예외 정보로 구분
  - `throw new RuntimeException("A");`
  - `throw new RuntimeException("B");`
  - `try { ... } catch (RuntimeException e) {`
    `String m = e.getMessage();`
    `if(m.equals("A")) ... if(m.equals("B")) ...`

여러 예외를 구분짓고 싶다

- 방법2: 사용자 정의 예외 사용
  - `class MyExceptionA extends Exception {}`
  - `class MyExceptionB extends Exception {}`
  - `throw new MyExceptionA();`
  - `throw new MyExceptionB();`
  - ```
    try { ... }
    catch (MyExceptionA e) { ... }
    catch (MyExceptionB e) { ...}
    ```
    ```