옛 java에서 java8로

생각의 전환

Java 8이 출시되었습니다. 만...

“어차피 사용하려면 한참 있어야하잖아.”하고 생각하고 있진 않나요?

확실히, 아직 이르긴 하죠.

하지만, 혹시 지금

Java6나 Java7을 열심히 공부하고 있진 않나요?

시작해야만 알 수 있을때도 있죠.

자 그럼, 언제 해야하나요?

from old java
to modern java

인프라본부 웹개발1팀 신윤섭 책임연구원

그럼, 바로 코드로.

```
Map<Dept, Long> groupByDeptAndFilter(List<Emp> list){
        return list.stream()
                        .collect(Collectors.groupingBy(emp -> emp.dept))
                        .entrySet()
                        .stream()
                        .collect(
                                Collectors.toMap(entry -> entry.getKey(),
                                                entry -> entry.getValue().stream()
                                                                .filter(emp -> emp.sal > 1000L)
                                                                .count()
                                )
                        );
}
```

이게 뭐야..ㅋ 뭔 소릴 하는거야.

잘 모르겠어. for문이랑 map으로 다시 써 줄래?

```
Map<Dept, Long> groupByDeptClassic(List<Emp> list){
        Map<Dept, Long> result = new HashMap<Dept, Long>();
        for(Emp emp : list){
                if( !result.containsKey(emp.dept) ){
                        result.put(emp.dept, 0L);
                }
                if(emp.sal > 1000L){
                        Long count = result.get(emp.dept);
                        count++;
                        result.put(emp.dept, count);
                }
        }
        return result;
}
```

아.. 이제 좀 뭔지 알겠네...

Welcome to

사고전환

from old java
to modern java
생각의 전환을 위한 Java8 입문

Lesson 1

새로운 파일 조작을 이해하자

복습:

finally에서 close하는 정석.

```
List<String> readFileSe6(String filename){
        List<String> lines = new ArrayList<String>();
        BufferedReader reader = null;
        try{
                reader = new BufferedReader( new FileReader(filename) );
                String line;
                while( (line = reader.readLine()) != null ){
                        lines.add("<"+line+">");
                }
                reader.close();
        }catch(IOException ex){
                throw new RuntimeException(ex);
        }
        return lines;
}
```

이러면 안되죠잉

```
List<String> readFileSe6(String filename){
        List<String> lines = new ArrayList<String>();
        BufferedReader reader = null;
        try{
                reader = new BufferedReader( new FileReader(filename) );
                String line;
                while( (line = reader.readLine()) != null ){
                        lines.add("<"+line+">");
                }
        }catch(IOException ex){
                throw new RuntimeException(ex);
        }finally{
                try{
                        if(reader != null ){
                                reader.close();
                        }
                }catch(Exception e){
                }
        }
        return lines;
}
```

finally에서 close하는게 정석이죠

finally 에서 close하는것은 낡은 정석
(~Java6)

try-with-resources 를 적용하면
(Java7~)

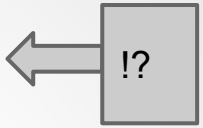
```
List<String> readFileSe7_1(String filename){
    List<String> lines = new ArrayList<>();
    try(   FileReader in = new FileReader(filename);
           BufferedReader reader = new BufferedReader( in ) ){
           String line;
           while( (line = reader.readLine()) != null ){
                  lines.add("<"+line+">");
           }
    }catch(IOException ex){
           throw new RuntimeException(ex);
    }
    return lines;
}
```

Diamond interface에 의한 타입 추론

이게 바로
try-with-resources

근데, 좀 이상한 부분이 있네요.

```
List<String> readFileSe7_1(String filename){
    List<String> lines = new ArrayList<>();
    try(   FileReader in = new FileReader(filename);
           BufferedReader reader = new BufferedReader( in )  ){
           String line;
           while( (line = reader.readLine()) != null ){
                  lines.add("<"+line+">");
           }
    }catch(IOException ex){
           throw new RuntimeException(ex);
    }
    return lines;
}
```



```
List<String> readFileSe7_2(String filename){
	List<String> lines = new ArrayList<>();
	Path path = Paths.get(filename);
	try(   BufferedReader reader = Files.newBufferedReader( path )  ){
		String line;
		while( (line = reader.readLine()) != null ){
			lines.add("<"+line+">");
		}
	}catch(IOException ex){
		throw new RuntimeException(ex);
	}
	return lines;
}
```

새로운정석 1 :

try-with-resources과 Files와Path로 조작한다.

(Java7~)

새로운정석 2 :

파일을 한번에 읽으려면 Files.readAllLines !

(Java7~)

```
List<String> readFileSe7_3(String filename){
	try{
		List<String> lines = Files.readAllLines(Paths.get(filename));
		int lineIdx = 0;
		for (Iterator<String> it = lines.iterator(); it.hasNext();) {
			String line = it.next();
			lines.set(lineIdx++, "<"+line+">");
		}
		return lines;
	}catch(IOException ex){
		throw new RuntimeException(ex);
	}
}
```

한번에 읽어서 편리하긴하지만 Heap 메모리도 소모하게되고, 매 라인마다 데이타 변경이 필요한 경우 for문을 순환할 필요가 있음.

Java8 에서는?

```
List<String> readFileSe8_1(String filename){
        List<String> lines = new ArrayList<>();
        try{
                Files.lines(Paths.get(filename))
                        .forEach(s -> lines.add("<"+s+">"));
        }catch(IOException ex){
                throw new UncheckedIOException(ex);
        }
        return lines;
}
```

여전히 결과 리턴을 위한 collection 인스턴스를 초기화 하고 있음.
이 초기화 조차도 stream api와 람다식을 통해 제거 가능..

새로운정석 3 :

Files.lines 로 처리한다.

(Java8~)

새로운정석 4 :

for와 while을 보게되면 Stream API 로 치환을 생각하자.

```
List<String> readFileSe8_2(String filename){
        try{
                return Files.lines(Paths.get(filename))
                                .map(s -> "<"+s+">")
                                .collect(Collectors.toList());
        }catch(IOException ex){
                throw new UncheckedIOException(ex);
        }
}
```

앞선 예제의
List<String> lines = new ArrayList<>();
도 map과 collect를 통해 제거.

잠깐 이쯤에서 퀴즈.

다음과 같은 코드는...

```
void readFileSe8_3(String filename,List<String> lines){
        Path path = Paths.get(filename);
        try(BufferedWriter writer = Files.newBufferedWriter(path ) ){
                lines.forEach( s -> writer.write("<"+s+">"));
        }catch(IOException ex){
                throw new UncheckedIOException(ex);
        }
}
```

1.잘 동작하는 코드인것 같다.

2.행의 결과 순서가 뒤죽박죽이 될것 같다.

3.컴파일 에러가 발생할것 같다.

4.실행시 예외가 발생할것 같다.

```
void readFileSe8_3(String filename,List<String> lines){
        Path path = Paths.get(filename);
        try(BufferedWriter writer = Files.newBufferedWriter(path ) ){
                lines.forEach( s -> writer.write("<"+s+">"));
        }catch(IOException ex){
                throw new UncheckedIOException(ex);
        }
}
```

writer.write() 는 IOException 예외를 던집니다.

~~1.잘 동작하는 코드인것 같다.~~

~~2.행의 결과 순서가 뒤죽박죽이 될것 같다.~~

3.컴파일 에러가 발생한다.

~~4.실행시 예외가 발생한다.~~

```
void readFileSe8_3(String filename,List<String> lines){
        Path path = Paths.get(filename);
        try(BufferedWriter writer = Files.newBufferedWriter(path ) ){
                lines.forEach( s -> {
                        try{
                                writer.write("<"+s+">");
                        }catch(IOException ex){
                                throw new UncheckedIOException(ex);
                        }
                });
        }catch(IOException ex){
                throw new UncheckedIOException(ex);
        }
}
```

어찌됐든 Lambda로 처리하는게 편한것만은 아니네요.

Lesson 2

문자열 조작은 어떻게 바뀌는가?

```
String joinSE7(List<String> lines){
        StringBuilder builder = new StringBuilder();
        for(String line : lines){
                builder.append(line).append(",");
        }
        return builder.substring(0, (builder.length()>1)?builder.length()-1:0).toString();
}
```

모두가 잘아는 문자열 결합 방법이네요.

```
String joinCommons(List<String> lines){
        return StringUtils.join(lines, ”,”);
}
```

Commons Lang 을 사용하면 이렇구요..

```
String joinSE8(List<String> lines){
        return String.join(”,”, lines);
}
```

새로운정석 5 :

String.join 반가워~

(Java8~)

```
String joinPrefixSE7(List<String> lines){
        StringBuilder builder = new StringBuilder();
        for(String line : lines){
                builder.append("<")
                        .append(line)
                        .append(">")
                        .append(",");
        }
        return builder.substring(0, (builder.length()>1)?builder.length()-1:0).toString();
}
```

이런 패턴은 어쩔...

```
String joinCommons(List<String> lines){
        return “<” + StringUtils.join(lines, ”>,<”) + “>”;
}
```

도와줘 Java8

```
String joinPrefixSE8(List<String> lines){
        return lines.stream()
                    .map(s -> "<"+s+">")
                    .collect(Collectors.joining(","));
}
```

새로운정석 6 :
String.join 잘가~
(Java8~)

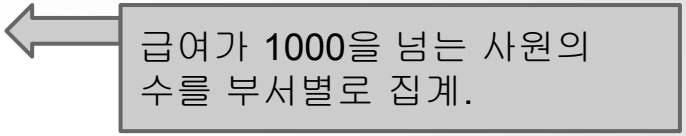
결국, 새로운정석 4 :

for와 while을 보게되면 Stream API 로 치환을 생각하자.

Lesson 3

for랑 while과 if등이 있는 이런저런 처리를 하는 복합적인 경우.

```
Map<Dept, Long> groupByDeptClassic(List<Emp> list){
        Map<Dept, Long> result = new HashMap<Dept, Long>();
        for(Emp emp : list){
                if( !result.containsKey(emp.dept) ){
                        result.put(emp.dept, 0L);
                }
                if(emp.sal > 1000L){
                        Long count = result.get(emp.dept);
                        count++;
                        result.put(emp.dept, count);
                }
        }
        return result;
}
```



Java8에서는 처음엔, 이렇게 작성하지 않을까?

```
Map<Dept, Long> groupByDeptClassic(List<Emp> list){
        Map<Dept, Long> result = new HashMap<Dept, Long>();
        for(Emp emp : list){
                result.putIfAbsent(emp.dept, 0L);
                if(emp.sal > 1000L){
                        Long count = result.get(emp.dept);
                        count++;
                        result.put(emp.dept, count);
                }
        }
        return result;
}
```

새로운 편리한 API

하지만, 새로운정석 4 :

“for와 while을 보게되면 Stream API 로 치환을 생각하자.”를 떠올리자.

```
Map<Dept, Long> groupByDeptAndFilter(List<Emp> list) {
        return list.stream()
                        .collect(Collectors.groupingBy(emp -> emp.dept))
                        .entrySet()
                        .stream()
                        .collect(
                                Collectors.toMap(entry -> entry.getKey(),
                                                entry -> entry.getValue().stream()
                                                                .filter(emp -> emp.sal > 1000L)
                                                                .count()
                                )
                        );
}
```

코드를 어떻게 쓰는거야?
코드를 어떻게 읽는거야?

```
select
  DEPT_ID, COUNT(EMP_ID)
from
  EMP
where
  EMP.SAL > 1000
group by
  EMP.DEPT_ID
```

SQL과 마찬가지.
공부로 눈에 익히고,
코드를 작성해 몸에 익혀
비로소 사용할수 있게 됩니다.

input : List<Emp>

output : Map<Dept, Emp의 수>

다만, Emp의 급여가 1000초과인.

1. 일단, List<Emp>를 Map<Dept, List<Emp>>로 그루핑.

2. Map의 값인 List<Emp>를 1000으로 필터링.

3. 필터링 이후의 List<Emp>를 카운트.

```
Map<Dept, Long> groupByDeptAndFilterLambda(List<Emp> list){
        Map<Dept, List<Emp>> groupByDept = list.stream()
                        .collect(Collectors.groupingBy(emp -> emp.dept));
        Map<Dept, List<? super Emp>> filtered = groupByDept.entrySet()
                        .stream()
                        .collect(Collectors.toMap(entry -> entry.getKey(),
                                        entry -> entry.getValue().stream()
                                        .filter(emp -> emp.sal > 1000)
                                        .collect(Collectors.toList())
                                        )
                        );
        Map<Dept, Long> counted = filtered.entrySet()
                        .stream()
                        .collect(Collectors.toMap(entry -> entry.getKey(),
                                        entry -> entry.getValue()
                                        .stream()
                                        .count()
                                        )
                        );
        return counted;
}
```

```
Map<Dept, Long> groupByDeptAndFilter(List<Emp> list){
        return list.stream()
                        .collect(Collectors.groupingBy(emp -> emp.dept))
                        .entrySet()
                        .stream()
                        .collect(
                                Collectors.toMap(entry -> entry.getKey(),
                                                entry -> entry.getValue().stream()
                                                                .filter(emp -> emp.sal > 1000L)
                                                                .count()
                                )
                        );
}
```

정리

for와 while을 보게되면 Stream API 로 치환을 생각하자.

코드리뷰에서 for와 while이 눈에 띄면 Lambda식으로 변경 시키자.

코드리뷰에서 100라인짜리 Lambda가 나온다면..

...

어떻게하면 읽기쉬운 코드를 작성할 수 있을지는 팀원과 생각해서 자신들만의 표준을 정해봅니다.

References

from old java to modern java. 谷本 心 (tanimoto sin, @cero_t)

가장 빨리 만나는 자바8. 카이 호스트만

Functional Programming in Java 8. 벤컷 수브라마니암