1. 스트림

- 자바 8부터 추가된 컬렉션(배열 포함)의 저장 요소를 하나씩 참조해서 람다식(함수적 스타일)으로 처리할 수 있도록 해주는 반복자이다.

2. 병렬처리 스트림

- 런타임 시 하나의 작업을 서브 작업으로 자동으로 나누고, 서브 작업의 결과를 자동으로 결합해서 최종 결과물을 생성해준다.

순차 처리 스트림
Stream<String> stream = list.stream(); stream.forEach(ParalleExample :: print);

병렬 처리 스트림
Stream<String> stream = list.paralleStream(); stream.forEach(ParalleExample :: print);

3. 스트림 종류

컬렉션으로 스트림 얻기
list.Stream();

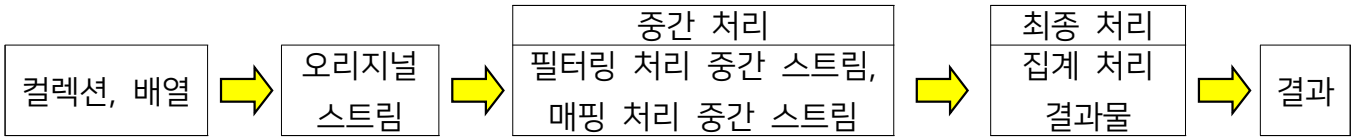
배열에서 스트림 얻기
Arrays.stream(arr);

범위(수) 스트림 얻기
IntStream.rangeClosed(1, 100);

파일에서 스트림 얻기
Files.lines(path, Charset.defaultCharset());

디렉터리에서 스트림 얻기
Files.list(path);

4. 스트림 처리 방식



5. 중간 처리 메소드와 최종 처리 메소드

종류		리턴타입	메소드(매개변수)	소속된 인터페이스
중간처리	필터링	Stream, IntStream, LongStream, DoubleStream	distinct()	공통
			filter(...)	공통
	매핑		flatMap(...)	공통
			flatMapToDouble(...)	Stream
			flatMapToInt(...)	Stream
			flatMapToLong(...)	Stream
			map(...)	공통
			mapToDouble(...)	Stream, IntStream, LongStream
			mapToInt(...)	Stream, LongStream, DoubleStream
			mapToLong(...)	Stream, IntSTream, DoubleStream
			mapToObj(...)	IntStream, LongStream, DoubleStream
			asDoubleStream()	IntStream, LongStream
			asLongStream()	IntStream
			boxed()	IntStream, LongStream, DoubleStream
	정렬		sorted(...)	공통
	루핑		peek(...)	공통
최종처리	매칭	boolean	allMatch(...)	공통
		boolean	anyMatch(...)	공통
		boolean	noneMatch(...)	공통
	집계	long	count()	공통
		OptionalXXX	findFirst(...)	공통
		OptionalXXX	max(...)	공통
		OptionalXXX	min(...)	공통
		OptionalDouble	average()	IntStream, LongStream, DoubleStream
		OptionalXXX	reduce(...)	공통
		int, long, double	sum()	IntStream, LongStream, DoubleStream
	루핑	void	forEach(...)	공통
	수집	R	collect(...)	공통

6. 필터링 - distinct()

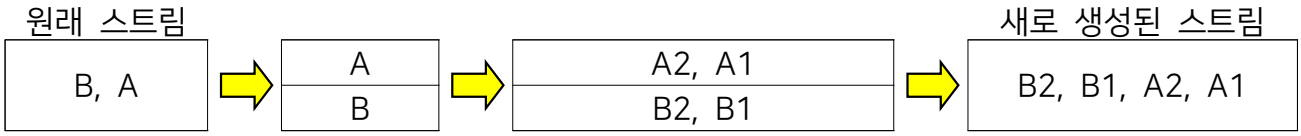
- 중복 제거를 위해 사용.
- Stream의 경우 Object.equals가 true이면 동일한 객체로 판단하고 중복을 제거한다.
- IntStream, LongStream, DoubleStream은 동일값일 경우 중복을 제거한다.

7. 필터링 - filter()

- 매개값으로 주어진 Predicate가 true를 리턴하는 요소만 필터링한다.

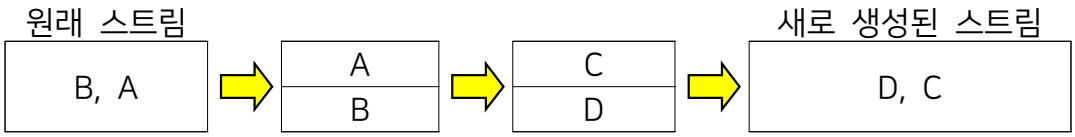
8. 매핑 - flatMapXXX()

- 요소를 대체하는 복수 개의 요소들로 구성된 새로운 스트림을 리턴한다.



9. 매핑 - mapXXX()

- 요소를 대체하는 요소로 구성된 새로운 스트림을 리턴한다.



10. 매핑 - asDoubleStream()

- IntStream의 int 요소 또는 LongStream의 long 요소를 double 요소로 타입 변환해서 DoubleStream을 생성한다.

11. 매핑 - asLongStream()

- IntStream의 int 요소를 long 요소로 타입변환해서 LongStream을 생성한다.

12. 매핑 - boxed()

- int, long, double 요소를 Integer, Long, Double 요소로 박싱해서 Stream을 생성한다.

13. 정렬 - sorted()

- 요소가 최종 처리되기 전에 중간 단계에서 요소를 정렬해서 최종 처리 순서를 변경한다.

리턴 타입	메소드(매개변수)	설명
Stream<T>	sorted()	객체를 Comparable 구현 방법에 따라 정렬
Stream<T>	sorted(Comparator<T>)	객체를 주어진 Comparator에 따라 정렬
DoubleStream	sorted()	double 요소를 오름차순으로 정렬
IntStream	sorted()	int 요소를 오름차순으로 정렬
LongStream	sorted()	long 요소를 오름차순으로 정렬

14. 루핑 - peek()

- 중간처리 메소드이다.
- 중간 처리 단계에서 전체 요소를 루핑하면서 추가적인 작업을 하기 위해 사용한다.

15. 루핑 - forEach()

- 최종처리 메소드이다.
- 최종 처리 단계에서 전체 요소를 루핑하면서 추가적인 작업을 하기 위해 사용한다.

16. 매칭 - allMatch()

- 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하는지 boolean으로 리턴한다.

17. 매칭 - anyMatch()

- 최소한의 한 개의 요소가 매개값 Predicate의 조건을 만족하는지 boolean으로 리턴한다.

18. 매칭 - noneMatch()

- 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하지 않는지 boolean으로 리턴한다.

19-1. 기본집계 - sum(), count(), average(), max(), min()

리턴 타입	메소드(매개변수)	설명
long	count()	요소 개수
OptionalXXX	findFirst()	첫 번째 요소
Optional<T> OptionalXXX	max(Comparator<T>) max()	최대 요소
Optional<T> OptionalXXX	min(Comparator<T>) min()	최소 요소
OptionalDouble	average()	요소 평균
int, long, double	sum	요소 총합

19-2. Optional 클래스

- OptionalXXX는 자바 8에서 추가한 java.util 패키지의 Optional, OptionalDouble, OptionalInt, OptionalLong 클래스 타입을 말한다. 이들은 값을 저장하는 값 기반 클래스(value-based class)들이다. 이 객체에서 값을 얻기 위해서는 get(), getAsDouble(), getAsInt(), getAsLong()을 호출하면 된다.
- Optional 클래스는 단순히 집계 값만 저장하는 것이 아니라, 집계 값이 존재하지 않을 경우 디폴트 값을 설정할 수도 있고, 집계 값을 처리하는 Consumer도 등록할 수 있다.

리턴 타입	메소드(매개변수)	설명
boolean	isPresent()	값이 저장되어 있는지 여부
T double int long	orElse(T) orElse(double) orElse(int) orElse(long)	값이 저장되어 있지 않을 경우 디폴트 값 지정
void	ifPresent(Consumer) ifPresent(DoubleConsumer) ifPresent(IntConsumer) ifPresent(LongConsumer)	값이 지정되어 있을 경우 Consumer에서 처리

20. 커스텀 집계 - reduce()

- 스트림은 기본 집계 메소드인 sum(), average(), count(), max(), min()을 제공하지만, 프로그램화해서 다양한 집계 결과물을 만들 수 있도록 reduce()를 제공한다.

21. 수집 - collect()

- 요소들의 최종 처리 메소드이다.
- 필요한 요소만 컬렉션으로 담을 수 있고, 요소들을 그룹핑한 후 집계(리덕션)할 수 있다.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.stream.IntStream;

public class Main {

	public static void main(String[] args) throws Exception{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

		String s = br.readLine();

		int[] intArr = IntStream.generate(() -> -1).limit(26).toArray();
		IntStream charsStream = s.chars();

		int[] index = {0};
		charsStream.forEach((i) -> {
			if(intArr[i - 'a'] == -1) {
				intArr[i - 'a'] = index[0];
			}
			index[0]++;
		});

		for(int i = 0; i < intArr.length; i++) {
			bw.write(String.valueOf(intArr[i]));
			if(i != intArr.length - 1) {
				bw.write(" ");
			}
		}

		bw.flush();
	}
}
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

public class Main {

	public static void main(String[] args) throws Exception{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));



		Map<Character, String> map = new HashMap<>();

		map.put('0', "000");
		map.put('1', "001");
		map.put('2', "010");
		map.put('3', "011");
		map.put('4', "100");
		map.put('5', "101");
		map.put('6', "110");
		map.put('7', "111");

		String s = br.readLine()
				.chars()
				.mapToObj(c -> map.get((char)c))
				.collect(Collectors.joining());

		if(s.equals("000")) {
			s = "0";
		}else {
			int start = -1;
			for(int i = 0; i < s.length(); i++) {
				char c = s.charAt(i);
				if(start == -1 && c == '1') {
					start = i;
				}
			}

			if(start != -1) {
				s = s.substring(start);
			}
		}
```

```
                bw.write(s);
                bw.flush();
        }
}
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class Main {

	public static void main(String[] args) throws Exception{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

		int cnt = Integer.parseInt(br.readLine());
		for(int i = 0; i < cnt; i++) {
			Map<Character, Integer> map = new HashMap<>();
			String answer = "";

			br.readLine().chars().forEach(n -> {
				char c = (char)n;
				if(c != ' ') {
					map.put(c, map.getOrDefault(c, 0) + 1);
				}
			});

			Entry<Character, Integer> entry1 = Collections.max(map.entrySet(), (entryA, entryB) -> entryA.getValue() - entryB.getValue());
			map.remove(entry1.getKey());
			Entry<Character, Integer> entry2 = null;
			if(map.size() > 0) {
				entry2 = Collections.max(map.entrySet(), (entryA, entryB) -> entryA.getValue() - entryB.getValue());
			}

			answer = entry1.getKey() + "₩n";
			if(entry2 != null) {
				if(entry1.getValue() == entry2.getValue()) {
					answer = "?₩n";
				}
			}


			bw.write(answer);
		}
```

```
                bw.flush();
                bw.close();
                br.close();
        }
}
```