1️⃣

Chapter 1

❓ **궁금증**

이 장에서는 자바가 거듭 변화하는 이유, 컴퓨팅 환경의 변화, 자바에 부여되는 시대적 변화의 요구, 자바 8,9의 새로운 핵심 기능을 소개한다.

변화하는 자바

여러가지 요구사항들이 생기면서, 자바는 이 변화에 대응해 여러가지 변화를 거쳐 왔다.

그 중에서도, 자바 8에서는 획기적인 변화를 거쳐 자바의 유행에 힘을 보탰다.

이 책에서는 자바 8이후의 변화에 대해 집중적으로 알아 볼 예정이다.

1.1 역사적인 흐름은 어떤가?

앞서 말했듯, 자바 8에서는 큰 변화가 생겼다.

말로만 하면 이해하기가 힘드니, 두개의 코드로 비교하면서 보도록 하자.

```
Collection.sort(inventory, new Comparator<Apple>() {
  public int compare(Apple a1, Apple a2) {
    return a1.getWeight().compareTo(a2.getWeight());
  }
}
```

이러한 코드가 자바 8로 작성하면

```
inventory.sort(comparing(Apple::getWeight));
```

이와 같이 굉장히 **자연어와 가까운 형식**으로 표현할 수 있다.

`자바 8 이전`의 자바 프로그램 에서는, **코어 하나만** 사용했다. (나머지 코어는 유휴 idle 상태나, OS 바이러스 검사 프로그램과 프로세스 파워를 나눠 사용)

나머지 코어를 활용하기 위해 스레드를 사용했는데, 이는 관리하기 어렵고 많은 문제를 야기했다.

자바의 병렬 실행 환경을 쉽게 관리, 에러를 제거하기위한 노력

- **Java 1.0** : 스레드와 락, 메모리 모델 지원
- **Java 5** : 스레드 풀(thread pool), 병렬 실행 컬렉션 (concurrent collection) 등
- **Java 7 :** 포크/조인 프레임워크 제공

하지만 이런 노력에도, 개발자가 활용하기는 쉽지 않았다.

하지만 걱정하지 마시라, Java 8부터는 몇가지 규칙만 지킨다면 새롭고 단순한 방식으로 병렬 실행을 할 수 있다!

Java 9에서는 리액티브 프로그래밍이라는 병렬 실행 기법을 지원하는데, RxJava (리액티브 스트림 툴킷) 를 표준적인 방식으로 지원한다.

Java 8의 새로운 기능

```
- 스트림 API
- 메서드에 코드를 전달하는 기법 ( 일등함수 만들기 )
- 인터페이스의 디폴트 메서드
```

1장에서는 이것들에 대해 **정말 간단하게**만 설명하고, 추후에 자세히 설명한다.

스트림 API

DB의 **Query언어**에서 표현식을 처리하는 것처럼 **병렬 연산을 지원**하는 **Stream API**를 제공한다.

- DB Query에서 고수준 언어로 원하는 동작을 표현하면, 구현(자바는 Stream 라이브러리)에서 최적의 저수준 실행 방법을 선택하는 방식으로 동작함

→ 스트림을 이용하면 에러가 많고, 멀티코어 CPU보다 훨씬 비싼 **synchronized를 안써도 된다.**

메서드에 코드를 전달하는 기법

> 새롭고 간결한 방식으로 동작 파라미터화(behavior parameterization)를 구현 가능함

약간만 다른 두 메서드가 있을 때, 인수를 이용해서 다른 동작을 하도록 하나의 메서드로 합치는 것이 바람직할 수 있다. (이를 통해 복붙을 피함) 이는 **함수형 프로그래밍**에서 위력을 발휘한다.

1.2 왜 아직도 자바는 변화하는가?

700개가 넘는 수많은 언어들이 탄생했고, 각각의 언어는 필요에 따라 유행하고, 도태된다. 자바는 어떻게 유행을 이어갈 수 있는지 알아보도록 하자.

프로그래밍 언어 생태계에서 자바의 위치

자바는 `출발`이 좋았다. 스레드와 락을 통해 소소한 **동시성도 지원**했다. 코드를 **JVM 바이트코드**로 컴파일 하고, 모든 브라우저에서 가상 머신 코드를 지원해 **인터넷 애플릿 프로그램의 주요한 언어**가 됐다.

최근 **JVM의 최신 업데이트** 덕분에 경쟁 언어는 JVM에서 더 부드럽게 실행되며, **자바와 상호동작**할 수 있게 되었다. 추가로 다양한 임베디드 컴퓨팅 분야를 성공적으로 장악하고 있다.

> **자바는 어떻게 대중적인 프로그래밍 언어로 성장했나?**

- 캡슐화 & 객체지향의 정신적인 모델 덕
- WORA (write once run anywhere)
- 어플을 실행하는 데 추가적인 비용으로 반감이 있었으나 하드웨어의 발전으로 프로그래머의 시간이 더욱 중요한 요소로 부각됨

불어오는 빅데이터의 바람

> 빅데이터를 효과적으로 처리해야 한다 → 병렬 프로세싱을 활용해야 한다

자바 8은 이러한 환경변화에 맞춰서, 자바에 없던 완전히 새로운 개념이지만 현재 시장에서 요구하는 기능을 효과적으로 제공한다.

앞으로는 다른 장들과는 다르게 **"필요성"**을 곁들여 세 가지 개념을 설명하며유닉스 기반 환경과 비슷한 점은 무엇이며, 자바 8의 새로운 멀티코어 병렬성이 강화된 이유를 풀어간다.

스트림 처리

> 스트림이란 한 번에 한 개씩 만들어지는 연속적인 데이터 항목들의 모임이다.

연속적인 자동차 생산 공장처럼, 조립 라인은 자동차를 물리적인 순서로 한 개씩 운반하지만 각각의 작업장에서는 동시에 작업을 처리한다.

스트림 API는 우리가 하려는 작업을 고수준으로 추상화해서 일련의 스트림으로 만들어 처리할 수 있다. 스트림 파이프라인을 이용해 입력 부분을 여러 CPU 코어에 쉽게 할당할 수 있다.

동작 파라미터화로 메서드에 코드 전달하기

> 코드 일부를 API로 전달하는 기능

자바 8에서는 메서드를 다른 메서드의 인수로 넘겨주는 기능을 제공함.

→ **동적 파라미터화**

병렬성과 공유 가변 데이터

> '병렬성을 공짜로 얻을 수 있다'라는 말에서 시작된다.

스트림 메서드로 전달하는 코드는 다른 코드와 동시에 실행해도 안전하게 실행돼야 한다.

→ **공유된 가변 데이터에 접근하지 않아야** 한다. (순수 함수, 부작용 없는 함수, 상태 없는 함수)

공유되지 않은 가변 데이터, 메서드, 함수 코드를 다른 메서드로 전달하는 두 가지 기능은 **함수형 프로그래밍** 패러다임의 핵심적인 사항이다.

자바가 진화해야 하는 이유

자바는 제네릭, List와 같이 많은 기능들이 추가되어 왔다. 하지만 이로 편리함을 누리고 있다.

또한, 틀에 박힌 Iterator 대신 for-each 루프를 사용할 수 있어 함수형 프로그래밍으로 다가섰다는 것이 자바 8의 가장 큰 변화이다.

언어는 하드웨어나 프로그래머 기대의 변화에 부응하는 방향으로 변화해야 한다.

1.3 자바 함수

> 자바 8에서는 함수를 **새로운 값의 형식**으로 추가했다. **멀티코어**에서 **병렬 프로그래밍**을 활용할 수 있는 **스트림과 연계**될 수 있도록 함수를 만들었다.

자바에서의 함수 ❓

- 메서드, 특히 정적 메서드와 같은 의미로 사용
- 이에 더해 **수학적인 함수**처럼 사용, 부작용을 일으키지 않는 함수

함수를 값처럼 취급?

프로그래밍 언어의 핵심은 값을 바꾸는 것이다. 값을 바꿀 수 있는 값들을 **일급 값(혹은 시민 [citizens])**이라고 한다. 값을 바꿀 수 없는 이급 값들을 일급 값으로 바꾸려는 노력이 스몰토크, 자바스크립트같은 언어에서 진행되고 있다.

이는 런타임에 메서드를 전달할 수 있다면, 즉 메서드를 일급 시민으로 만들면 프로그래밍에 유용하게 활용 할 수 있어 중요하다고 볼 수 있다.

메서드와 람다를 일급 시민으로

> ### 메서드 참조 (method reference)

한 예로, 디렉터리의 모든 숨겨진 파일을 필터링한다고 가정하자.

```
File[] hiddenFiles = new File(".").listFiles(new FileFilter(){
    public boolean accept(FIle file){
        return file.isHidden();
  }
});
```

이와 같이 File에 isHidden()메서드가 있지만 굳이 FileFilter 인스턴스로 감싸줘야 한다.

이를 아래와 같이 간단하게 표현할 수 있다.

```
File[] hiddenFiles = new File(".").listFiles(File::isHidden);
```

이미 isHidden이라는 **함수**는 준비되어 있으므로 자바 8의 **메서드참조 ::(이 메서드를 값으로 사용하라는 의미)** 를 이용해 listFiles에 직접 전달할 수 있다.

기존에 객체 참조(new로 객체 참조를 생성함) 를 이용해서 객체를 주고받았던 것처럼,

자바 8에서는 File:isHidden을 이용해서 **메서드 참조**를 만들어 전달할 수 있게 되었다.

람다

자바 8은 람다(익명함수)를 통해서, 간단하게 사용되는 메서드들을 인자로 보낼 수 있다.

1.4 스트림

스트림 API로 컬렉션을 처리하면서 발생하는 모호함과 반복적인 코드 문제, 멀티코어 활용 어려움을 해결했다.

주로 제공하는 기능

- 필터링
- 추출 (ex. 리스트에서 각 사과의 무게 필드 추출)
- 그룹화

이러한 동작들을 쉽게 병렬화 할 수 있다.

→ 포킹 단계 (나누기) → 각각의 CPU가 자신의 구역을 처리 → 결과 합치기

더욱 자세한 내용은 4,5장에서 알아본다

1.5 디폴트 메서드와 자바 모듈

모듈

자바 9 이전에는 자바 패키지 집합을 포함하는 JAR 파일을 제공하는 것에 비해,

자바 9의 모듈 시스템은 모듈을 정의하는 문법을 제공해 패키지 모음을 포함하는 모듈을 정의 할 수 있다.

→ 모듈 덕분에 JAR같은 컴포넌트에 구조를 적용할 수 있으며, 문서화와 모듈 확인 작업이 용이해짐

디폴트 메서드

인터페이스를 쉽게 바꿀 수 있도록 지원됨

→ 미래에 프로그램이 쉽게 변화할 수 있는 환경을 제공하는 기능

아래의 1.4 절 예와 같은 코드에서, List<T>가 stream, parallelStream을 제공하지 않는다.

```
List<Apple> heavyApples1 = inventory.stream().filter((Apple a) -> a.getWeight() > 150)
                         .collect(toList());
List<Apple> heavyApples2 = inventory.parallelStream().filter((Apple a) -> a.getWeight() > 150)
                         .collect(toList());
```

이를 바꾸기 위해 원래는 Collection 인터페이스(List 가 구현하고 있는 인터페이스)에 stream, parralleStream 메서드를 추가하고 ArrayList 클래스에서 메서드를 구현해야 한다.

하지만 만약에 이렇게 한다면, **Collection API의 인터페이스를 구현한 모든 클래스에서 새로 추가된 메서드를 구현해야 한다.**

> 그럼, 어떻게 기존의 구현을 고치지 않고 이미 공개된 인터페이스를 변경할 수 있을까??

자바 8은 구현 클래스에서 **구현하지 않아도 되는 메서드**를 인터페이스에 추가할 수 있는 기능을 제공한다. 메서드 본문(bodies)은 클래스 구현이 아니라 인터페이스의 일부로 포함된다. 이를 **디폴트 메서드**라고 부른다.

→ 이를 통해 기존의 코드를 건드리지 않고 인터페이스 설계를 자유롭게 확장할 수 있다.

예로, 자바 8에서는 List에 직접 sort 메서드를 호출 할 수 있는데, 이는 자바 8의 List 인터페이스에 아래와 같은 디폴트 메서드 정의가 추가되었기 때문이다.

```
default void sort(Comparator<? super E> c{
    Collections.sort(this, c);
}
```

자바 8 이전에는 List를 구현하는 모든 클래스가 sort를 구현해야 했지만 자바 8부턴 디폴트 sort 구현안해도된다.

하나의 클래스에 여러 인터페이스를 구현할 수 있고, **다중 디폴트 메서드**가 존재한다면?? **다중 상속**이 허용되는것인가?

→ 어느정도는 '그렇다'라고 말할 수 있다. 9장에서 `다이아몬드 상속 문제` 를 피할 수 있는 방법 소개

1.6 함수형 프로그래밍에서 가져온 다른 유용한 아이디어

스트림 API는 이 두가지 아이디어를 모두 활용한다.

- 메서드&람다를 일급값으로 사용
- 가변 공유 상태가 없는 병렬 실행을 이용해서 효율적이고 안전하게 함수나 메서드 호출할수 있다.

Nullpointer 예외를 피하기 위한 방법

Optional<T> 클래스

자바 8에서 제공되는 Optional<T>는 값을 갖거나 갖지 않을 수 있는 컨테이너 객체이다.

값이 없는 상황을 어떻게 처리할지 명시적으로 구현하는 메서드를 포함한다.

→ Nullpointer 예외 피할 수 있다.

(구조적) 패턴 매칭 기법

여기서 패턴 매칭은 if-then-else가 아닌, 케이스로 정의하는 수학과 함수형 프로그래밍의 기능 의미

→ ex) 주어진 디렉터리에서 모든 IMG*, JPG 파일을 검색하는 정규표현식

패턴매칭은 switch를 확장한 것으로 데이터 형식 분류와 분석을 한 번에 수행할 수 있다.