2장 - 동적 파라미터화 코드 전달하기

이 장의 내용

- 변화하는 요구사항에 대응
- 동작 파라미터화
- 익명 클래스
- 람다 표현식 미리보기
- 실전 예제 : Comparator, Runnable, GUI

동작 파라미터화

- 유지보수 유리
- 어떻게 실행할 것인지 결정하지 않은 코드 블록
- 컬렉션 탐색 로직과 각 항목에 적용할 동작을 분리할 수 있다.

> 거의 비슷한 코드가 반복 존재한다면 그 코드를 추상화한다.

❓ 프레디케이트

- 참, 거짓을 반환하는 함수를 가르킴

전략 디자인 패턴

- 각 알고리즘(전략)을 캡슐화하는 알고리즘 패밀리를 정의해둔 다음 런타임에 알고리즘을 선택하는 기법

- 예제에선 ApplePredicate가 알고리즘 패밀리, AppleHeavyWeightPredicate, AppleGreenColorPredicate가 전략이다.

여기서 이제 이런 코드가 나올 수 있게 되었다.

```
public static List<Apple> filterApples(List<Apple> inventory, ApplePredicate p){
  List<Apple> result = new ArrayList<>();
  for(Apple apple: inventory){
  // 이 아래 부분이 매우 중요하다!!!
    if(p.test(apple)){
        result.add(apple);
    }
  }
  return result;
}
```

원래는 p.test(apple) 부분을 계속 복.붙으로 구현을 했어야했지만, 이제는 interface를 구현해 전달하면서 쓸 수 있다.

```
List<Apple> redAndHeavyApples = filterApples(inventory, new AppleRedAndHeavyPredicate());
```

이제 여기서 저 쓸데없이 매번 instance를 만드는 부분이 거슬리게 된다.

복잡한 과정 간소화

인터페이스 구현 후 클래스를 정의하고, 인스턴스화 하는게 귀찮아!

→ 익명 클래스 기법 사용

→ 사실 람다 표현식이 더 좋음

익명 클래스

- 지역 클래스와 비슷한 개념임

- 클래스 선언과 인스턴스화를 동시에 할 수 있음!

람다

- 익명 클래스도 장황함이 없지는 않다.
- 이를 해결하기위해 람다를 도입했다.

리스트 형식으로 추상화

- 이제는 Apple 뿐만 아니라 다른 클래스도 받도록 개조했다.

```
public interface Predicate<T> {
    boolean test(T t);
}

public static <T> List<T> filter(List<T> list, Predicate<T> p){
  List<T> result = new ArrayList<>();
  for(T e: list){
    if(p.test(e)){
      result.add(e);
    }
  }
  return result;
}
```

~~쩌..쩐다!~~