이팩티브 자바 완벽 공략 1부

“객체 생성과 파괴”, “모든 객체의 공통 메소드”

인프런 / 백기선 (AKA, WHITESHIP)

이팩티브 자바 (Effective Java)

조류와 블로크 지음 / 개앞맵시(이복연) 번역

- 모든 자바 개발자의 필독서
- 하지만, 대부분의 자바 개발자가 소화하기 어려운 내용이 많다.
- 그래서 이 강의를 준비했다.

객체 생성과 파괴

객체를 생성하는 다양한 방법과 파괴 전에 수행해야 할 정리 작업을 관리하는 방법

아이템 1. 생성자 대신 정적 팩터리 메서드를 고려하라.

핵심 정리

- 장점
  - 이름을 가질 수 있다. (동일한 시그니처의 생성자를 두개 가질 수 없다.)
  - 호출될 때마다 인스턴트를 새로 생성하지 않아도 된다. (Boolean.valueOf)
  - 반환 타입의 하위 타입 객체를 반환할 수 있는 능력이 있다. (인터페이스 기반 프레임워크, 인터페이스에 정적 메소드)
  - 입력 매개변수가 따라 매번 다른 클래스의 객체를 반환할 수 있다. (EnumSet)
  - 정적 팩터리 메서드를 작성하는 시점에는 반환할 객체의 클래스가 존재하지 않아도 된다. (서비스 제공자 프레임워크)
- 단점
  - 상속을 하려면 public이나 protected 생성하기 필요하니 정적 팩터리 메서드만 제공하면 하위 클래스를 만들 수 없다.
  - 정적 팩터리 메서드는 프로그래머가 찾기 어렵다.

아이템 1. 생성자 대신 정적 팩터리 메서드를 고려하라.

완벽 공략

- p9, 열거 타입은 인스턴트가 하나만 만들어짐을 보장한다.
- p9, 같은 객체가 자주 요청되는 상황이라면 플라이웨이트 패턴을 사용할 수 있다.
- p10, 자바 8부터는 인터페이스가 정적 메서드를 가질 수 없다는 제한이 풀렸기 때문에 인스턴스화 불가 동반 클래스를 둘 이유가 별로 없다.
- p11, 서비스 제공자 프레임워크를 만드는 근간이 된다.
- p12, 서비스 제공자 인터페이스가 없다면 각 구현체를 인스턴스로 만들 때 리플렉션을 사용해야 한다.
- p12, 브리지 패턴
- p12, 의존 객체 주입 프레임워크

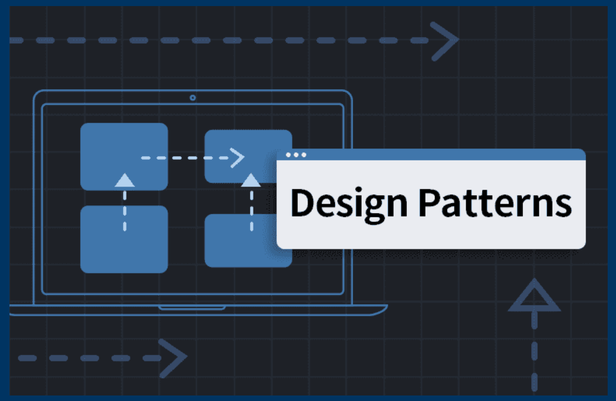
완벽 공략 1. 열거 타입

Enumeration

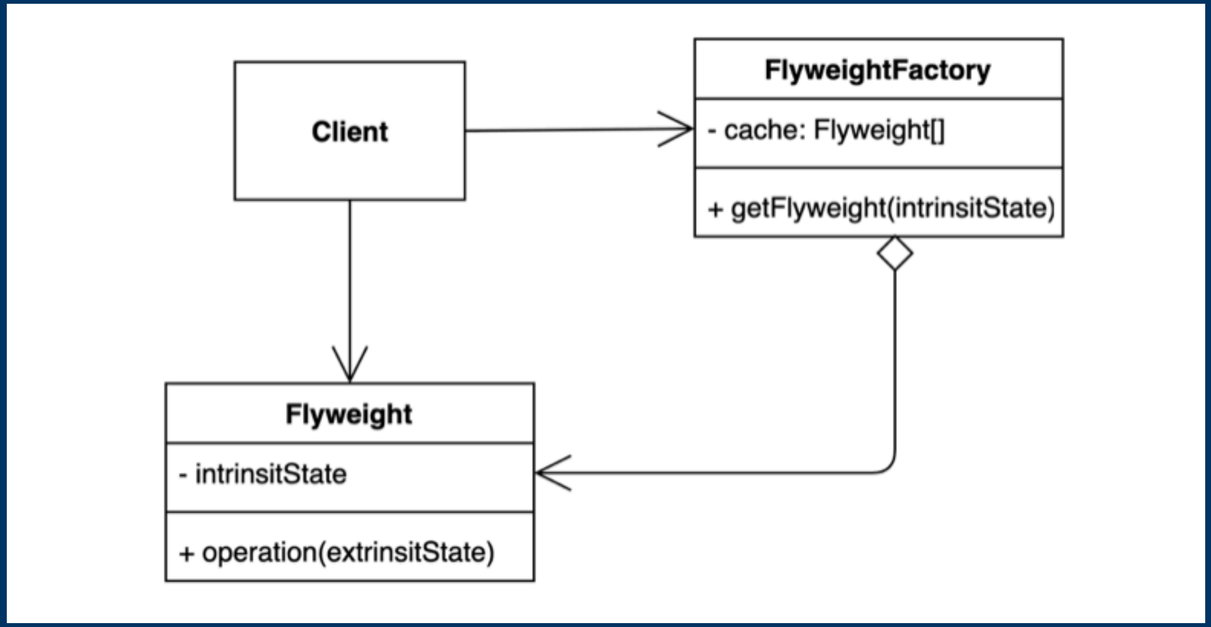
- 상수 목록을 담을 수 있는 데이터 타입.
- 특정한 변수가 가질 수 있는 값을 제한할 수 있다. 타입-세이프티 (Type-Safety)를 보장할 수 있다.
- 싱글톤 패턴을 구현할 때 사용하기도 한다.
- 질문1) 특정 enum 타입이 가질 수 있는 모든 값을 순회하며 출력하라.
- 질문2) enum은 자바의 클래스처럼 생성자, 메소드, 필드를 가질 수 있는가?
- 질문3) enum의 값은 == 연산자로 동일성을 비교할 수 있는가?
- 과제) enum을 key로 사용하는 Map을 정의하세요. 또는 enum을 담고 있는 Set을 만들어 보세요.

완벽 공략 2. 플라이웨이트 패턴

Flyweight (가벼운 체급)



- 객체를 가볍게 만들어 메모리 사용을 줄이는 패턴.
- 자주 변하는 속성(또는 외적인 속성, extrinsit)과 변하지 않는 속성(또는 내적인 속성, intrinsit)을 분리하고 재사용하여 메모리 사용을 줄일 수 있다.



완벽 공략 3. 인터페이스에 정적 메소드

자바 8과 9에서 주요 인터페이스의 변화

- 기본 메소드 (default method)와 정적 메소드를 가질 수 있다.
- 기본 메소드
  - 인터페이스에서 메소드 선언 뿐 아니라, 기본적인 구현체까지 제공할 수 있다.
  - 기존의 인터페이스를 구현하는 클래스에 새로운 기능을 추가할 수 있다.
- 정적 메소드
  - 자바 9부터 private static 메소드도 가질 수 있다.
  - 단, private 필드는 아직도 선언할 수 없다.
- 질문1) 내림차순으로 정렬하는 Comparator를 만들고 List<Integer>를 정렬하라.
- 질문2) 질문1에서 만든 Comparator를 사용해서 오름차순으로 정렬하라.

완벽 공략 4. 서비스 제공자 프레임워크

확장 가능한 애플리케이션을 만드는 방법

- 주요 구성 요소
  - 서비스 제공자 인터페이스 (SPI)와 서비스 제공자 (서비스 구현체)
  - 서비스 제공자 등록 API (서비스 인터페이스의 구현체를 등록하는 방법)
  - 서비스 접근 API (서비스의 클라이언트가 서비스 인터페이스의 인스턴스를 가져올 때 사용하는 API)
- 다양한 변형
  - 브릿지 패턴
  - 의존 객체 주입 프레임워크
  - java.util.ServiceLoader
    - https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html
    - https://docs.oracle.com/javase/tutorial/ext/basics/spi.html

완벽 공략 5. 리플렉션
reflection

- 클래스로더를 통해 읽어온 클래스 정보(거울에 반사"된 정보)를 사용하는 기술
- 리플렉션을 사용해 클래스를 읽어오거나, 인스턴스를 만들거나, 메소드를 실행하거나, 필드의 값을 가져오거나 변경하는 것이 가능하다.
- 언제 사용할까?
  - 특정 애노테이션이 붙어있는 필드 또는 메소드 읽어오기 (JUnit, Spring)
  - 특정 이름 패턴에 해당하는 메소드 목록 가져와 호출하기 (getter, setter)
  - …
- https://docs.oracle.com/javase/tutorial/reflect/

아이템 2. 생성자에 매개변수가 많다면 빌더를 고려하라.

핵심 정리

- 정적 팩터리와 생성자에 선택적 매개변수가 많을 때 고려할 수 있는 방안
  - 대안1: 점층적 생성자 패턴 또는 생성자 체이닝
    - 매개변수가 늘어나면 클라이언트 코드를 작성하거나 읽기 어렵다.
  - 대안2: 자바빈즈 패턴
    - 완전한 객체를 만들려면 메서드를 여러번 호출해야 한다. (일관성이 무너진 상태가 될 수도 있다.)
    - 클래스를 불변으로 만들 수 없다.

아이템 2. 생성자에 매개변수가 많다면 빌더를 고려하라.

핵심 정리

- 권장하는 방법: 빌더 패턴
  - 플루언트 API 또는 메서드 체이닝을 한다.
  - 계층적으로 설계된 클래스와 함께 사용하기 좋다.
  - **점층적 생성자보다 클라이언트 코드를 읽고 쓰기가 훨씬 간결하고, 자바빈즈보다 훨씬 안전하다.**

아이템 2. 생성자에 매개변수가 많다면 빌더를 고려하라.

완벽 공략

- p15, 자바빈즈, 게터, 세터
- p17, 객체 얼리기 (freezing)
- p17, 빌더 패턴
- p19, IllegalArgumentException
- P21, 가변인수 (varargs) 매개변수를 여러 개 사용할 수 있다.

완벽 공략 6. 자바빈(JavaBean)이란?

(주로 GUI에서) 재사용 가능한 소프트웨어 컴포넌트

2.1 What is a Bean?

Let's start with an initial definition and then refine it:

> **"A Java Bean is a reusable software component that can be manipulated visually in a builder tool."**

This covers a wide range of different possibilities.

The builder tools may include web page builders, visual application builders, GUI layout builders, or even server application builders. Sometimes the "builder tool" may simply be a document editor that is including some beans as part of a compound document.

Some Java Beans may be simple GUI elements such as buttons and sliders. Other Java Beans may be sophisticated visual software components such as database viewers, or data feeds. Some Java Beans may have no GUI appearance of their own, but may still be composed together visually using an application builder.

https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/

완벽 공략 6. 자바빈(JavaBean)이란?

java.beans 패키지 안에 있는 모든 것

- 그 중에서도 자바빈이 지켜야 할 규약
  - 아규먼트 없는 기본 생성자
  - getter와 setter 메소드 이름 규약
  - Serializable 인터페이스 구현
- 하지만 실제로 오늘날 자바빈 스팩 중에서도 getter와 setter가 주로 쓰는 이유는?
  - JPA나 스프링과 같은 여러 프레임워크에서 리플렉션을 통해 특정 객체의 값을 조회하거나 설정하기 때문입니다.

완벽 공략 7. 객체 얼리기 (freezing)

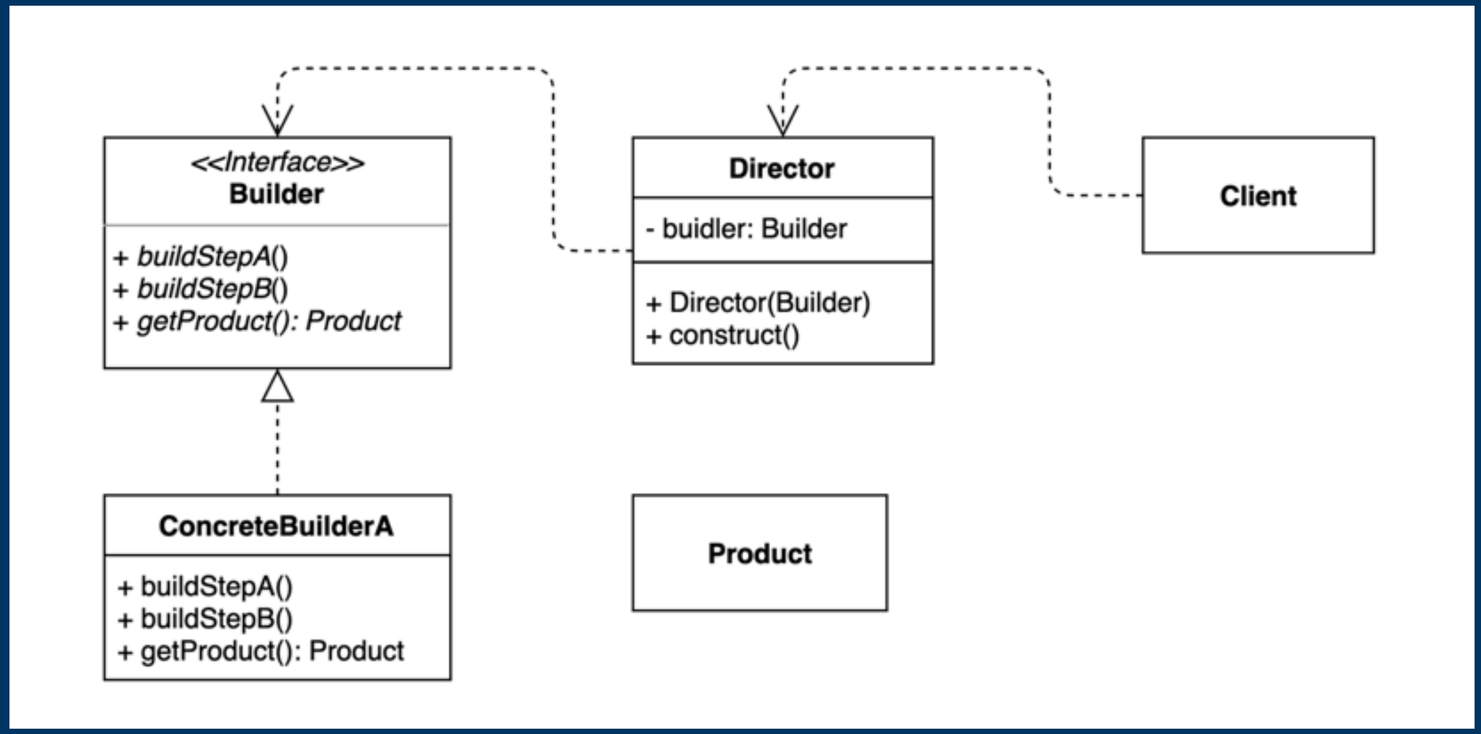
임의의 객체를 불변 객체로 만들어주는 기능.

- Object.freeze()에 전달한 객체는 그뒤로 변경될 수 없다.
  - 새 프로퍼티를 추가하지 못함
  - 기존 프로퍼티를 제거하지 못함
  - 기존 프로퍼티 값을 변경하지 못함
  - 프로토타입을 변경하지 못함
- strict 모드에서만 동작함
- 비슷한 류의 펑션으로 Object.seal()과 Object.preventExtensions()가 있다.

완벽 공략 8. 빌더 패턴

동일한 프로세스를 거쳐 다양한 구성의 인스턴스를 만드는 방법.

- 복잡한 객체를 만드는 프로세스를 독립적으로 분리할 수 있다.



완벽 공략 9. IllegalArgumentException

잘못된 인자를 넘겨 받았을 때 사용할 수 있는 기본 런타임 예외

```
public void setAsText(String text) throws java.lang.IllegalArgumentException {
    if (value instanceof String) {
        setValue(text);
        return;
    }
    throw new java.lang.IllegalArgumentException(text);
}
```

- 질문1) checked exception과 unchecked exception의 차이?
- 질문2) 간혹 메소드 선언부에 unchecked exception을 선언하는 이유는?
- 질문3) checked exception은 왜 사용할까?
- 과제1) 자바의 모든 RuntimeException 클래스 이름 한번씩 읽어보기.
- 과제2) 이 링크에 있는 글을 꼭 읽으세요.

완벽 공략 10. 가변인수

여러 인자를 받을 수 있는 가변적인 argument (Var+args)

- 가변인수는 메소드에 오직 하나만 선언할 수 있다.
- 가변인수는 메소드의 가장 마지막 매개변수가 되어야 한다.

```
public void printNumbers(int... numbers) {
    System.out.println(numbers.getClass().getCanonicalName());
    System.out.println(numbers.getClass().getComponentType());
    Arrays.stream(numbers).forEach(System.out::println);
}
```

아이템 3. 생성자나 열거 타입으로 싱글턴임을 보증하라.

첫번째 방법: private 생성자 + public static final 필드

```
public class Elvis {
    public static final Elvis INSTANCE = new Elvis();

    private Elvis() { }

}
```

- 장점, 간결하고 싱글턴임을 API에 들어낼 수 있다.
- **단점 1, 싱글톤을 사용하는 클라이언트 테스트하기 어려워진다.**
- 단점 2, 리플렉션으로 private 생성자를 호출할 수 있다.
- 단점 3, 역직렬화 할 때 새로운 인스턴스가 생길 수 있다.

아이템 3. 생성자나 열거 타입으로 싱글턴임을 보증하라.

두번째 방법: private 생성자 + 정적 팩터리 메서드

- 두번째 방법: private 생성자 + 정적 팩터리 메서드
  - 장점 1. API를 바꾸지 않고도 싱글턴이 아니게 변경할 수 있다.
  - 장점 2. 정적 팩터리를 제네릭 싱글턴 팩터리로 만들 수 있다.
  - 장점 3. 정적 팩터리의 메서드 참조를 공급자(Supplier)로 사용할 수 있다.

```
public class Elvis {
    private static final Elvis INSTANCE = new Elvis();
    private Elvis() { }
    public static Elvis getInstance() { return INSTANCE; }
}
```

아이템 3. 생성자나 열거 타입으로 싱글턴임을 보증하라.

세번째 방법: 열거 타입

- 세번째 방법: 열거 타입
  - 가장 간결한 방법이며 직렬화와 리플렉션에도 안전하다.
  - 대부분의 상황에서는 원소가 하나뿐인 열거 타입이 싱글턴을 만드는 가장 좋은 방법이다.

```
public enum Elvis {
    INSTANCE;
}
```

아이템 3. 생성자나 열거 타입으로 싱글턴임을 보증하라.

완벽 공략

- p23, 리플렉션 API로 private 생성자 호출하기
- p24, 메서드 참조를 공급자로 사용할 수 있다.
- p24, Supplier<T>, 함수형 인터페이스
- p24, 직렬화, 역직렬화, Serializable, transient

완벽 공략 11. 메소드 참조

메소드 하나만 호출하는 람다 표현식을 줄여쓰는 방법

- 스태틱 메소드 레퍼런스
- 인스턴스 메소드 레퍼런스
- 임의 객체의 인스턴스 메소드 레퍼런스
- 생성자 레퍼런스
- https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

완벽 공략 12. 함수형 인터페이스

자바가 제공하는 기본 함수형 인터페이스

- 함수형 인터페이스는 람다 표현식과 메소드 참조에 대한 “타겟 타입”을 제공한다.
- 타겟 타입은 변수 할당, 메소드 호출, 타입 변환에 활용할 수 있다.
- 자바에서 제공하는 기본 함수형 인터페이스 익혀 둘 것. (java.util.function 패키지)
- 함수형 인터페이스를 만드는 방법.
- 심화 학습 1) Understanding Java method invocation with invokedynamic
- 심화 학습 2) LambdaMetaFactory

완벽 공략 13. 객체 직렬화

객체를 바이트스트림으로 상호 변환하는 기술

- 바이트스트림으로 변환한 객체를 파일로 저장하거나 네트워트를 통해 다른 시스템으로 전송할 수 있다.
- Serializable 인터페이스 구현
- transient를 사용해서 직렬화 하지 않을 필드 선언하기
- serialVersionUID는 언제 왜 사용하는가?
- 심화 학습) 객체 직렬화 스팩

아이템 4. 인스턴스화를 막으려거든 private 생성자를 사용하라

핵심 정리

- 정적 메서드만 담은 유틸리티 클래스는 인스턴스로 만들어 쓰려고 설계한 클래스가 아니다.
- 추상 클래스로 만드는 것으로는 인스턴스화를 막을 수 없다.
- private 생성자를 추가하면 클래스의 인스턴스화를 막을 수 있다.
- 생성자에 주석으로 인스턴스화 불가한 이유를 설명하는 것이 좋다.
- 상속을 방지할 때도 같은 방법을 사용할 수 있다.

아이템 5. 자원을 직접 명시하지 말고 의존 객체 주입을 사용하라

핵심 정리

- 사용하는 자원에 따라 동작이 달라지는 클래스는 정적 유틸리티 클래스나 싱글턴 방식이 적합하지 않다.
- 의존 객체 주입이란 인스턴스를 생성할 때 필요한 자원을 넘겨주는 방식이다.
- 이 방식의 변형으로 생성자에 자원 팩터리를 넘겨줄 수 있다.
- 의존 객체 주입을 사용하면 클래스의 유연성, 재사용성, 테스트 용이성을 개선할 수 있다.

아이템 5. 자원을 직접 명시하지 말고 의존 객체 주입을 사용하라

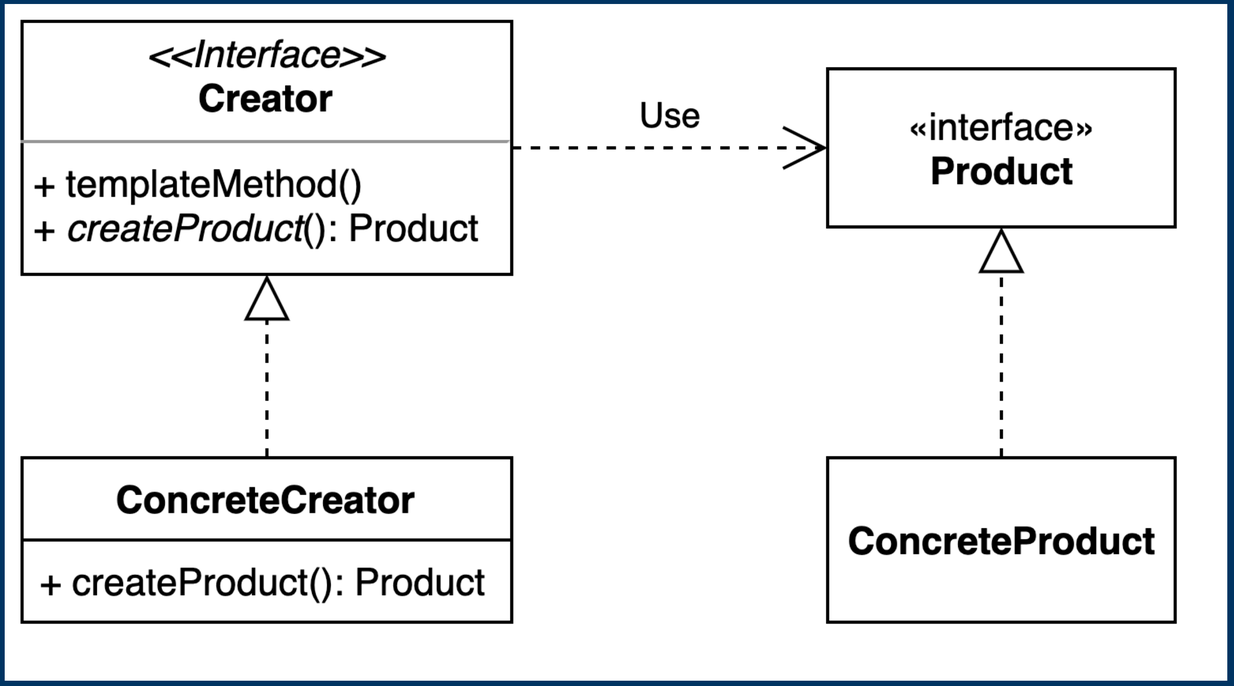
완벽 공략

- p29, 이 패턴의 쓸만한 변형으로 생성자에 자원 팩터리를 넘겨주는 방식이 있다.
- p29, 자바 8에서 소개한 Supplier<T> 인터페이스가 팩터리를 표현한 완벽한 예다.
- p29, 한정적 와일드카드 타입을 사용해 팩터리의 타입 매개변수를 제한해야 한다.
- p29, 팩터리 메소드 패턴
- p30, 의존 객체가 많은 경우에 Dagger, Guice, 스프링 같은 의존 객체 주입 프레임워크 도입을 고려할 수 있다.

완벽 공략 14. 팩터리 메소드 패턴

구체적으로 어떤 인스턴스를 만들지는 서브 클래스가 정한다.

- 새로운 Product를 제공하는 팩토리를 추가하더라도, 팩토리를 사용하는 클라이언트 코드는 변경할 필요가 없다.



완벽 공략 15. 스프링 IoC

BeanFactory 또는 ApplicationContext

- Inversion of Control - 뒤짚힌 제어권
  - 자기 코드에 대한 제어권을 자기 자신이 가지고 있지 않고 위부에서 제어하는 경우.
  - 제어권? 인스턴스를 만들거나, 어떤 메소드를 실행하거나, 필요로하는 의존성을 주입 받는 등…
- 스프링 IoC 컨테이너 사용 장점
  - 수많은 개발자에게 검증되었으며 자바 표준 스팩(@Inject)도 지원한다.
  - 손쉽게 싱글톤 Scope을 사용할 수 있다.
  - 객체 생성 (Bean) 관련 라이프사이클 인터페이스를 제공한다.

아이템 6. 불필요한 객체 생성을 피하라

핵심 정리

- 문자열
  - 사실상 동일한 객체라서 매번 새로 만들 필요가 없다.
  - new String(“자바”) 을 사용하지 않고 문자열 리터럴 (“자바”)을 사용해 기존에 동일한 문자열을 재사용하는 것이 좋다.
- 정규식, Pattern
  - 생성 비용이 비싼 객체라서 반복해서 생성하기 보다, 캐싱하여 재사용하는 것이 좋다.
- 오토박싱 (auto boxing)
  - 기본 타입(int)을 그게 상응하는 박싱된 기본 타입(Integer)으로 상호 변환해주는 기술.
  - 기본 타입과 박싱된 기본 타입을 섞어서 사용하면 변환하는 과정에서 불필요한 객체가 생성될 수 있다.
- **“객체 생성은 비싸니 피하라.”는 뜻으로 오해하면 안 된다.**

아이템 6. 불필요한 객체 생성을 피하라
완벽 공략

- p31, 사용 자제 API (Deprecation)
- p32, 정규 표현식
- p32, 한 번 쓰고 버려져서 가비지 컬렉션 대상이 된다.
- p33, 초기화 지연 기법 (아이템 83에서 다룸)
- p34, 방어적 복사 (아이템 50에서 다룸)

완벽 공략 16. Deprecation

클라이언트가 사용하지 않길 바라는 코드가 있다면…

- 사용 자제를 권장하고 대안을 제시하는 방법이 있다.
- @Deprecated
  - 컴파일시 경고 메시지를 통해 사용 자제를 권장하는 API라는 것을 클라이언트에 알려줄 수 있다.
- @deprecated
  - 문서화(Javadoc)에 사용해, 왜 해당 API 사용을 지양하며, 그 대신 권장하는 API가 어떤 것인지 표기할 수 있다.

완벽 공략 17. 정규 표현식

내부적으로 Pattern이 쓰이는 곳

- String.matches(String regex)
- String.split(String regex)
  - 대안, Pattern.compile(regex).split(str)
- String.replace*(String regex, String replacement)
  - 대안, Pattern.compile(regex).matcher(str).replaceAll(repl)
- 과제) 자바 정규표현식 Pattern 문법 학습하기
- 참고 1) https://docs.oracle.com/javase/tutorial/essential/regex/
- 참고 2) https://regex101.com/ 또는 https://regexr.com/

완벽 공략 18. 가비지 컬렉션

기본 개념

- Mark, Sweep, Compact
- Young Generation (Eden, S0, S1), Old Generation
- Minor GC, Full GC
- Throughput, Latency (Stop-The-World), Footprint
- Serial, Parallel, CMS, G1, ZGC, Shenandoah
- 참고) How to choose the best Java garbage collector

아이템 7. 다 쓴 객체 참조를 해제하라.

핵심 정리

- 어떤 객체에 대한 레퍼런스가 남아있다면 해당 객체는 가비지 컬렉션의 대상이 되지 않는다.
- 자기 메모리를 직접 관리하는 클래스라면 메모리 누수에 주의해야 한다.
  - 예) 스택, 캐시, 리스너 또는 콜백
- 참조 객체를 null 처리하는 일은 예외적인 경우이며 가장 좋은 방법은 유효 범위 밖으로 밀어내는 것이다.

아이템 7. 다 쓴 객체 참조를 해제하라.
완벽 공략

- p37, NullPointerException
- p38, WeakHashMap
  - p38, 약한 참조 (weak reference)
- p39, 백그라운드 쓰레드
  - p39, ScheduledThreadPoolExecutor

완벽 공략 19. NullPointerException

Java 8 Optional을 활용해서 NPE를 최대한 피하자.

- NullPointerException을 만나는 이유
  - 메소드에서 null을 리턴하기 때문에 && null 체크를 하지 않았기 때문에
- 메소드에서 적절한 값을 리턴할 수 없는 경우에 선택할 수 있는 대안
  - 예외를 던진다.
  - null을 리턴한다.
  - Optional을 리턴한다.

완벽 공략 19. NullPointerException

Java 8 Optional을 활용해서 NPE를 최대한 피하자.

- Optional 사용시 주의 할 점
  - 리턴값으로만 쓰기를 권장한다. (메소드 매개변수 타입, 맵의 키 타입, 인스턴스 필드 타입으로 쓰지 말자.) 왜?
  - Optional을 리턴하는 메소드에서 null을 리턴하지 말자.
  - 프리미티브 타입용 Optional을 따로 있다. OptionalInt, OptionalLong,...
  - Collection, Map, Stream Array, Optional은 Opiontal로 감싸지 말 것.
- 아이템 55, Optional 반환은 신중히 하라.

완벽 공략 20. WeakHashMap

더이상 사용하지 않는 객체를 GC할 때 자동으로 삭제해주는 Map

- Key가 더이상 강하게 레퍼런스되는 곳이 없다면 해당 엔트리를 제거한다.
- 레퍼런스 종류
  - Strong, Soft, Weak, Phantom
- 맵의 엔트리를 맵의 Value가 아니라 Key에 의존해야 하는 경우에 사용할 수 있다.
- 캐시를 구현하는데 사용할 수 있지만, 캐시를 직접 구현하는 것은 권장하지 않는다.

완벽 공략 21. ScheduledThreadPoolExecutor

Thread와 Runnable을 학습했다면 그 다음은 Executor.

- Thread, Runnable, ExecutorService
- 쓰레드풀의 개수를 정할 때 주의할 것
  - CPU, I/O
- 쓰레드툴의 종류
  - Single, Fixed, Cached, Scheduled
- Runnable, Callable, Future
- CompletableFuture, ForkJoinPool

아이템 8. finalizer와 cleaner 사용을 피하라

핵심 정리

- finalizer와 cleaner는 즉시 수행된다는 보장이 없다.
- finalizer와 cleaner는 실행되지 않을 수도 있다.
- finalizer 동작 중에 예외가 발생하면 정리 작업이 처리되지 않을 수도 있다.
- finalizer와 cleaner는 심각한 성능 문제가 있다.
- finalizer는 보안 문제가 있다.
- **반납할 자원이 있는 클래스는 AutoCloseable을 구현하고 클라이언트에서 close()를 호출하거나 try-with-resource를 사용해야 한다.**

아이템 8. finalizer와 cleaner 사용을 피하라

Cleaner 사용법

- 자원 반납용 안전망으로 사용할 수 있다.
  - PhantomReference를 사용한다.
  - 호출되리라는 보장이 없지만 없는 것 보다는 나을 수 있다.
- 네이티브 피어 자원 회수
  - 단, 성능 저하를 감당할 수 있고 네이티브 피어가 심각한 자원을 가지고 있지 않을 때에만 해당한다.
  - 네이티브 피어가 사용하는 자원을 즉시 회수해야 한다면 close() 메소드를 호출해야 한다.

아이템 8. finalizer와 cleaner 사용을 피하라

완벽 공략

- p42, Finalizer 공격
- p43, AutoClosable
- p45, 정적이 아닌 중첩 클래스는 자동으로 바깥 객체의 참조를 갖는다.
- p45, 람다 역시 바깥 객체의 참조를 갖기 쉽다.

완벽 공략 22. Finalizer 공격

만들다 만 객체를 finalize 메소드에서 사용하는 방법

- Finalizer 공격

```
public class BrokenAccount extends Account {

    public BrokenAccount(String accountId) {
        super(accountId);
    }

    @Override
    protected void finalize() throws Throwable {
        this.transfer(BigDecimal.valueOf(100), to: "keesun");
    }
}
```

- 방어하는 방법
  - final 클래스로 만들거나
  - finalizer() 메소드를 오버라이딩 한 다음 final을 붙여서 하위 클래스에서 오버라이딩 할 수 없도록 막는다.

완벽 공략 23. AutoClosable

try-with-resource를 지원하는 인터페이스

- void close() throws Exception
  - 인터페이스에 정의된 메서드에서 Exception 타입으로 예외를 던지지만
  - 실제 구현체에서는 구체적인 예외를 던지는 것을 추천하며
  - 가능하다면 예외를 던지지 않는 것도 권장한다.
- Closeable 클래스와 차이점
  - IOException을 던지며
  - 반드시 idempotent 해야 한다.

아이템 9. try-finally 보다 try-with-resouces를사용하라.

핵심 정리

- try-finally는 더이상 최선의 방법이 아니다. (자바7부터)
- try-with-resources를 사용하면 코드가 더 짧고 분명하다.
- 만들어지는 예외 정보도 훨씬 유용하다.

```
static String firstLineOfFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(
            new FileReader(path))) {
        return br.readLine();
    }
}
```

아이템 9. try-finally 보다 try-with-resouces를사용하라.

완벽 공략

- p48, 자바 퍼즐러 예외 처리 코드의 실수
- P49, try-with-resources 바이트코드

모든 객체의 공통 메서드

Object가 제공하는 메서드를 언제 어떻게 재정의해야 하는가

아이템 10. equals는 일반 규약을 지켜 재정의하라

핵심 정리: equals를 재정의 하지 않는 것이 최선

- 다음의 경우에 해당한다면 equals를 재정의 할 필요가 없다.
- 각 인스턴스가 본질적으로 고유하다.
- 인스턴스의 '논리적 동치성'을 검사할 필요가 없다.
- 상위 클래스에서 재정의한 equals가 하위 클래스에도 적절하다.
- 클래스가 private이거나 package-private이고 equals 메서드를 호출할 일이 없다.

아이템 10. equals는 일반 규약을 지켜 재정의하라

핵심 정리: equals 규약

- 반사성: A.equals(A) == true
- 대칭성: A.equals(B) == B.equals(A)
  - CaseInsensitiveString
- 추이성: A.equals(B) && B.equals(C), A.equals(C)
  - Point, ColorPoint(inherit), CounterPointer, ColorPoint(comp)
- 일관성: A.equals(B) == A.equals(B)
- null-아님: A.equals(null) == false

아이템 10. equals는 일반 규약을 지켜 재정의하라

핵심 정리: equals 구현 방법

- == 연산자를 사용해 자기 자신의 참조인지 확인한다.
- instanceof 연산자로 올바른 타입인지 확인한다.
- 입력된 값을 올바른 타입으로 형변환 한다.
- 입력 객체와 자기 자신의 대응되는 핵심 필드가 일치하는지 확인한다.

- 구글의 AutoValue 또는 Lombok을 사용한다.
- IDE의 코드 생성 기능을 사용한다.
- 과제) 자바의 Record를 공부하세요.

아이템 10. equals는 일반 규약을 지켜 재정의하라

핵심 정리: 주의 사항

- equals를 재정의 할 때 hashCode도 반드시 재정의하자. (아이템 11)
- 너무 복잡하게 해결하지 말자.
- Object가 아닌 타입의 매개변수를 받는 equals 메서드는 선언하지 말자.

아이템 10. equals는 일반 규약을 지켜 재정의하라

완벽 공략

- p53, 값 클래스
- p58, StackOverflowError
- p59, 리스코프 치환 원칙
- p59, 상속 대신 컴포지션을 사용하라. (아이템 18)

완벽 공략 24. Value 기반의 클래스

클래스처럼 생겼지만 int 처럼 동작하는 클래스

- 식별자가 없고 불변이다.
- 식별자가 아니라 인스턴스가 가지고 있는 상태를 기반으로 equals, hashCode, toString을 구현한다.
- == 오퍼레이션이 아니라 equals를 사용해서 동등성을 비교한다.
- 동일한(equals) 객체는 상호교환 가능한다.

- https://docs.oracle.com/javase/8/docs/api/java/lang/doc-files/ValueBased.html
- https://cr.openjdk.java.net/~jrose/values/values-0.html

완벽 공략 25. StackOverflowError

로컬 변수와 객체가 저장되는 공간의 이름은?

- 스택(stack)과 힙(heap)
- 메소드 호출시, 스택에 스택 프레임이 쌓인다.
  - 스택 프레임에 들어있는 정보: 메소드에 전달하는 매개변수, 메소드 실행 끝내고 돌아갈 곳, 힙에 들어있는 객채에 대한 레퍼런스 …
  - 그런데 더이상 스택 프레임을 쌓을 수 없다면? StackOverflowError!
- 스택의 사이즈를 조정하고 싶다면? -Xss1M
  - https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/optionX.html

완벽 공략 26. 리스코프 치환 원칙

객체 지향 5대 원칙 SOLID 중에 하나.

- 1994년, 바바라 리스코프의 논문, “A Behavioral Notion of Subtyping”에서 기원한 객체 지향 원칙.
- ‘하위 클래스의 객체’가 ‘상위 클래스 객체’를 대체하더라도 소프트웨어의 기능을 깨트리지 않아야 한다. (semantic over syntacic, 구문 보다는 의미!)

아이템 11. equals를 재정의하려거든 hashCode도 재정의하라

핵심 정리: hashCode 규약

- equals 비교에 사용하는 정보가 변경되지 않았다면 hashCode는 매번 같은 값을 리턴해야 한다. (변경되거나, 애플리케이션을 다시 실행했다면 달라질 수 있다.)
- 두 객체에 대한 equals가 같다면, hashCode의 값도 같아야 한다.
- 두 객체에 대한 equals가 다르더라도, hashCode의 값은 같을 수 있지만 해시 테이블 성능을 고려해 다른 값을 리턴하는 것이 좋다.

아이템 11. equals를 재정의하려거든 hashCode도 재정의하라

핵심 정리: hashCode 구현 방법

```
@Override public int hashCode() {
    int result = Short.hashCode(areaCode); // 1
    result = 31 * result + Short.hashCode(prefix); // 2
    result = 31 * result + Short.hashCode(lineNum); // 3
    return result;
}
```

1. 핵심 필드 하나의 값의 해쉬값을 계산해서 result 값을 초기화 한다.
2. 기본 타입은 Type.hashCode
   참조 타입은 해당 필드의 hashCode
   배열은 모든 원소를 재귀적으로 위의 로직을 적용하거나, Arrays.hashCode
   result = 31 * result + 해당 필드의 hashCode 계산값
3. result를 리턴한다.

아이템 11. equals를 재정의하려거든 hashCode도 재정의하라

핵심 정리: hashCode 구현 대안

- 구글 구아바의 com.google.common.hash.Hashing
- Objects 클래스의 hash 메서드
- 캐싱을 사용해 불변 클래스의 해시 코드 계산 비용을 줄일 수 있다.

아이템 11. equals를 재정의하려거든 hashCode도 재정의하라

핵심 정리: 주의 할 것

- 지연 초기화 기법을 사용할 때 스레드 안전성을 신경써야 한다. (아이템 83)
- 성능 때문에 핵심 필드를 해시코드 계산할 때 빼면 안 된다.
- 해시코드 계산 규칙을 API에 노출하지 않는 것이 좋다.

아이템 11. equals를 재정의하려거든 hashCode도 재정의하라

완벽 공략

- p68, 연결 리스트
- p70, 해시 충돌이 더욱 적은 방법을 꼭 써야 한다면…
- p71, 클래스를 스레드 안전하게 만들도록 신경 써야 한다.

완벽 공략 27. 해시맵 내부의 연결 리스트

내부 구현은 언제든지 바뀔 수도 있다.

- 자바 8에서 해시 충돌시 성능 개선을 위해 내부적으로 동일한 버켓에 일정 개수 이상의 엔트리가 추가되면, 연결 리스트 대신 이진 트리를 사용하도록 바뀌었다.
  - https://dzone.com/articles/hashmap-performance
- 연결 리스트에서 어떤 값을 찾는데 걸리는 시간은?
- 이진 트리에서 어떤 값을 찾는데 걸리는 시간은?

완벽 공략 28. 스레드 안전

멀티 스레드 환경에서 안전한 코드, Thread-safety

- 가장 안전한 방법은 여러 스레드 간에 공유하는 데이터가 없는 것!
- 공유하는 데이터가 있다면:
  - Synchronization
  - ThreadLocal
  - 불변 객체 사용
  - Synchronized 데이터 사용
  - Concurrent 데이터 사용
  - …

아이템 12. toString을 항상 재정의하라

핵심 정리

- toString은 간결하면서 사람이 읽히 쉬운 형태의 유익한 정보를 반환해야 한다.
- Object의 toString은 클래스이름@16진수로 표시한 해시 코드
- 객체가 가진 모든 정보를 보여주는 것이 좋다.
- 값 클래스라면 포맷을 문서에 명시하는 것이 좋으며 해당 포맷으로 객체를 생성할 수 있는 정적 팩터리나 생성자를 제공하는 것이 좋다.
- toString이 반환한 값에 포함된 정보를 얻어올 수 있는 API를 제공하는 것이 좋다.
- 경우에 따라 AutoValue, 롬복 또는 IDE를 사용하지 않는게 적절할 수 있다.

아이템 13. clone 재정의는 주의해서 진행하라.

핵심 정리: 애매모호한 clone 규약

- clone 규약
  - x.clone() != x 반드시 true
  - x.clone().getClass() == x.getClass() 반드시 true
  - x.clone().equals(x) true가 아닐 수도 있다.
- 불변 객체라면 다음으로 충분하다.
  - Cloenable 인터페이스를 구현하고
  - clone 메서드를 재정의한다. 이때 super.clone()을 사용해야 한다.

아이템 13. clone 재정의는 주의해서 진행하라.

핵심 정리: 가변 객체의 clone 구현하는 방법

- 접근 제한자는 public, 반환 타입은 자신의 클래스로 변경한다.
- super.clone을 호출한 뒤 필요한 필드를 적절히 수정한다.
  - 배열을 복제할 때는 배열의 clone 메서드를 사용하라.
  - 경우에 따라 final을 사용할 수 없을지도 모른다.
  - 필요한 경우 deep copy를 해야한다.
  - super.clone으로 객체를 만든 뒤, 고수준 메서드를 호출하는 방법도 있다.
  - 오버라이딩 할 수 있는 메서드는 참조하지 않도록 조심해야 한다.
  - 상속용 클래스는 Cloneable을 구현하지 않는 것이 좋다.
  - Cloneable을 구현한 스레드 안전 클래스를 작성할 때는 동기화를 해야 한다.

아이템 13. clone 재정의는 주의해서 진행하라.

핵심 정리: clone 대신 권장하는 방법

- “복사 생성자” 또는 변환 생성자, “복사 팩터리” 또는 변환 팩터리.
- 생성자를 쓰지 않으며, 모호한 규약, 불필요한 검사 예외, final 용법 방해 등에서 벗어날 수 있다.
- 또 다른 큰 장점 중 하나로 인터페이스 타입의 인스턴스를 인수로 받을 수 있다.
  - 클라이언트가 복제본의 타입을 결정할 수 있다.

- 읽어볼 것) Josh Bloch on Design, “Copy Constructor versus Cloning”

아이템 13. clone 재정의는 주의해서 진행하라.
완벽 공략

- p80, 비검사 예외(UnChecked Exception)였어야 했다는 신호다.
- p81. HashTable과 LinkedList
- p83, 깊은 복사 (deep copy)
- p83, 리스트가 길면 스택 오버플로를 일으킬 위험이 있기 때문이다.
- p85, clone 메서드 역시 적절히 동기화해줘야 한다.
- p86, TreeSet

완벽 공략 29. UncheckedException

왜 우리는 비검사 예외를 선호하는가?

- 컴파일 에러를 신경쓰지 않아도 되며,
- try-catch로 감싸거나
- 메서드 선언부에 선언하지 않아도 된다.
- 그렇다면 우리는 비검사 예외만 쓰면 되는걸까? 검사 예외는 왜 있는것일까?

완벽 공략 29. UncheckedException

그렇다면 우리는 비검사 예외만 쓰면 되는걸까?

- 왜 잡지 않은 예외를 메서드에 선언해야 하는가?
  - 메서드에 선언한 예외는 프로그래밍 인터페이스의 일부다. 즉, 해당 메서드를 사용하는 코드가 반드시 알아야 하는 정보다. 그래야 해당 예외가 발생했을 상황에 대처하는 코드를 작성할 수 있을테니까.
- 비검사 예외는 그럼 왜 메서드에 선언하지 않아도 되는가?
  - 비검사 예외는 어떤 식으로든 처리하거나 복구할 수 없는 경우에 사용하는 예외다. 가령, 숫자를 0으로 나누거나, null 레퍼런스에 메서드를 호출하는 등.
  - 이런 예외는 프로그램 전반에 걸쳐 어디서든 발생할 수 있기 때문에 이 모든 비검사 예외를 메서드에 선언하도록 강제한다면 프로그램의 명확도가 떨어진다.

완벽 공략 29. UncheckedException

언제 어떤 예외를 써야 하는가?

- 단순히 처리하기 쉽고 편하다는 이유만으로 RuntimeException을 선택하지는 말자.
- 가이드라인: 클라이언트가 해당 예외 상황을 복구할 수 있다면 검사 예외를 사용하고, 해당 예외가 발생했을 때 아무것도 할 수 없다면, 비검사 예외로 만든다.
- https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

완벽 공략 30. TreeSet

AbstractSet을 확장한 정렬된 컬렉션

- 엘리먼트를 추가한 순서는 중요하지 않다.
- 엘리먼트가 지닌 자연적인 순서(natural order)에 따라 정렬한다.
- 오름차순으로 정렬한다.
- 스레드 안전하지 않다.
- 과제) 이진 검색 트리, 레드 블랙 트리에 대해 학습하세요.

아이템 14. Comparable을 구현할지 고민하라

핵심 정리: compareTo 규약

- Object.equals에 더해서 순서까지 비교할 수 있으며 Generic을 지원한다.
- 자기 자신이 (this)이 compareTo에 전달된 객체보다 작으면 음수, 같으면 0, 크다면 양수를 리턴한다.
- 반사성, 대칭성, 추이성을 만족해야 한다.
- 반드시 따라야 하는 것은 아니지만 x.compareTo(y) == 0이라면 x.equals(y)가 true여야 한다.

아이템 14. Comparable을 구현할지 고민하라

핵심 정리: compareTo 구현 방법 1

- 자연적인 순서를 제공할 클래스에 implements Compratable<T> 을 선언한다.
- compareTo 메서드를 재정의한다.
- comprareTo 메서드 안에서 기본 타입은 박싱된 기본 타입의 compare을 사용해 비교한다.
- 핵심 필드가 여러 개라면 비교 순서가 중요하다. 순서를 결정하는데 있어서 가장 중요한 필드를 비교하고 그 값이 0이라면 다음 필드를 비교한다.
- 기존 클래스를 확장하고 필드를 추가하는 경우 compareTo 규약을 지킬 수 없다.
  - Composition을 활용할 것.

아이템 14. Comparable을 구현할지 고민하라

핵심 정리: compareTo 구현 방법 2

- 자바 8부터 함수형 인터페이스, 람다, 메서드 레퍼런스와 Comprator가 제공하는 기본 메서드와 static 메서드를 사용해서 Comprator를 구현할 수 있다.
- Comparator가 제공하는 메서드 사용하는 방법
  - Comparator의 static 메서드를 사용해서 Comparator 인스턴스 만들기
  - 인스턴스를 만들었다면 default 메서드를 사용해서 메서드 호출 이어가기 (체이닝)
  - static 메서드와 default 메소드의 매개변수로는 람다 표현식 또는 메서드 레퍼런스를 사용할 수 있다.

아이템 14. Comparable을 구현할지 고민하라
완벽 공략

- p90, 제네릭 인터페이스이므로 compareTo 메서드의 인수 타입은 컴파일타임에 정해진다.
- p92, 자바의 타입 추론 능력이 이 상황에서 타입을 알아낼 만큼…
- p93. 이 방식은 정수 오버플로를 일으키거나
- p93, IEEE 754 부동소수점 계산 방식에 따른 오류를 낼 수 있다.

감사합니다.

2부(클래스와 인터페이스, 제네릭)를 기대해 주세요.

백기선 올림