아이템 1. 생성자 대신 정적 팩토리 메서드를 고려하라

생성일	@2021년 8월 19일 오후 6:32
태그	

아이템 1. 생성자 대신 정적 팩토리 메서드를 고려하라

- 클래스는 생성자 말고도 정적 팩토리 메서드를 제공할 수 있다.

```
//단순한 기본타입 boolean을 받아 Bollean 객체 참조로 변환 해 줌
public static Boolean valueOf(boolean b) {
  return b ? Boolean.TRUE : Boolean.FALSE;
}
```

- 정적 팩토리 메서드가 생성자 보다 좋은 5가지
  - 이름을 가질 수 있다.

```
public class Member {
    private String name;
    private int age;

    //생성자를 private 으로 막아 불필요한 객체 생성을 피할 수 있게 함
    private Member(String name) {
        this.name = name;
    }
    private Member(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public static Member memberWithName(String name) {
        return new Member(name);
    }
}
```

  - 호출될 때 마다 인스턴스를 새로 생성하지 않아도 된다.

- 이러한 클래스를 인스턴스 통제 클래스라고 한다.
- 인스턴스를 통제하는 이유
  - 싱글톤으로 만들 수 있음
  - 불변 값 클래스에서 같은 인스턴스가 단 하나뿐임을 보장 할 수 있음

```
public class Dog {
    //싱글톤+불변객체화
    private static final Dog INSTANCE = new Dog();
    private Dog() {};

    public static Dog getInstance() {return INSTANCE;}
}
```

- 반환 타입의 하위 타입 객체를 반환할 수 있는 능력이 있다.

```
public class Fruit {
    public static Apple createApple() {
        return new Apple();
    }
}
class Apple extends Fruit {}
```

- 입력 매개변수에 따라 매번 다른 클래스의 객체를 반환할 수 있다.
  - 클라이언트는 아래 리턴되는 두 클래스에 대해 몰라도 된다.

```
public static <E extends Enum<E>> EnumSet<E> noneOf(Class<E> elementType) {
        Enum<?>[] universe = getUniverse(elementType);
        if (universe == null)
            throw new ClassCastException(elementType + " not an enum");

        if (universe.length <= 64)
            return new RegularEnumSet<>(elementType, universe);
        else
            return new JumboEnumSet<>(elementType, universe);
    }

//원소가 64개 이하면 long 변수 하나로 관리하는 RegularEnumSet 리턴,
//원소가 65개 이상이면 long 배열로 관리하는 JumboEumSet 리턴
```

- 정적 팩토리 메서드를 작성하는 시점에는 반환할 객체의 클래스가 존재하지 않아도 된다.

```
말이 좀 어렵게 느껴졌는데 말 그대로의 뜻을 담고 있다.
예를들어 정적 팩토리 메서드가

interface Soon {}

public static List<Soon> getInstance() {
  return new ArrayList<>();
}

일 경우 아직 인터페이스가 구현되지 않았음에도 getInstance() 호출 할 수 있다.
단 해당 인스턴스 컬렉션에 값을 넣기전에는 제네릭 인터페이스의 경우 구현되어야
에러가 발생하지 않는다.
```

- 정적 팩토리 메서드의 단점
  - 상속을 하려면 public 이나 protected 생성자가 필요하니 정적 팩토리 메서드만 제공하면 하위 클래스를 만들 수 없다.

```
public class Animal {
    private Animal() {}

    public static Cat createCat() {
        return new Cat();
    }
}

public class Cat extends Animal{
    public Cat(){
        super(); // 에러?!
    }
}

//정적 팩토리 메서드만 하기에 자식 클래스를 생성 시 부모 클래스를 먼저 생성 해야 하는데
//부모 클래스 생성자가 private 이라 컴파일 에러..!
```

    - 대신 컴포지션을 사용하도록 유도하는 점에서 장점으로도 받아들여짐

```
public class Cat {
  private Animal animal;
```

```
}
-> 상속이 아닌 인스턴스 변수로 가져오는 것을 컴포지션 이라 함
```

- 정적 팩토리 메서드는 프로그래머가 찾기 어렵다.
  - 생성자처럼 API 설명에 명확히 드러나지 않으므로 사용자는 정적 팩토리 메서드 방식 클래스를 인스턴스화 할 방법을 알아내야 함.

  - 일반적인 명명 방식
    - from : 매개변수 하나 받아, 해당 타입의 인스턴스를 반환하는 형변환 메서드
      - ex) Date d = Date.from(instant);
    - of : 여러 매개변수를 받아 적합한 타입의 인스턴스를 반환하는 집계 메서드
      - ex) Set<Rank> faceCards = EumSet.of(JACK,QUEEN,KING);
    - valueOf : from 과 of 의 더 자세한 버전
      - ex)BigInteger prime = BigInteger.valueOf(Integer.MAX_VALUE);
    - instance or getInstance : 매개변수를 받을 시에는 매개변수로 명시한 인스턴스를 반환 하지만 같은 인스턴스임을 보장하지는 않음. 매개변수 받지 않을 시 같은 인스턴스
    - create 혹은 newInstance : 매번 새로운 인스턴스를 생성해 반환함을 보장
      - ex) Object newArray = Array.newInstance(classObject, arrayLen);
    - getType : 생성할 클래스가 아닌 다른 클래스에 팩토리 메서드를 정의할 때 쓴다.
      - ex) FileStore fs = Files.getFileStore(path)
    - newType : newInstance 와 같으나 생성할 클래스가 아닌 다른 클래스에 팩토리 메서드를 정의할 때 씀
      - ex) BufferedReader br = Files.newBufferedReader(path);

- type : getType 과 newType 의 간결한 버전
  - ex) List<Complaint> litany = Collections.list(legacyitany);