상황(context)에 맞는 설계와 구현 방법을 찾아라.

프로그래밍 설계와 구현은 아날로그적인 영역이 많은 부분이다.
프로그래밍을 기술이 아닌 예술의 일부라고 생각하는 이유도 이런 점 때문이다.
프로그래밍 설계와 구현에 정답은 없다.
정답을 찾기보다 **요구사항에 적합한 최선의 설계와 구현 코드를 찾기 위해 노력**한다.

반복문 대신 재귀 함수로 구현할 수도 있다.

반복문으로 구현할 부분을 재귀 함수로 구현하는 것도 좋은 접근 방법이 될 수 있다.

```
private long inputPrice() {
    String price = "";
    do {
        System.out.println("구입 금액을 입력해 주세요.");
        price = SCANNER.nextLine();
    } while (!Validator.isNaturalNumber(price));
    return Long.parseLong(price.trim());
}
```

위와 같은 구현을 다음과 같이 구현할 수도 있다.

```
private static long getPrice() {
    try {
        System.out.println("구입 금액을 입력해 주세요.");
        return Long.parseLong(SCANNER.nextLine());
    } catch (IllegalArgumentException e) {
        return getPrice();
    }
}
```

원시 타입과 문자열을 포장하라

구입 금액을 Money 객체로 포장

```
public class Money {
    private static final int MONEY_PER_LOTTO = 1_000;

    private final int money;

    public Money(int money) {
        if (money < MONEY_PER_LOTTO) {
            throw new IllegalArgumentException
                ("로또 구입금액은 1000원 이상이어야 합니다.");
        }
        this.money = money;
    }

    [...]
}
```

로또 숫자 하나를 LottoNumber 객체로 포장

```
class LottoNumber {
    private final int no;

    LottoNumber(int no) {
        if (no < 1 || no > 45) {
            throw new IllegalArgumentException();
        }

        this.no = no;
    }

    [...]
}
```

적절한 Collection(자료구조)을 활용하라.

- 일차로 List, Set, Map을 적절하게 활용한다.
- 적절한 자료구조가 없다면 나만의 자료구조를 구현한다.
    - 나만의 클래스를 추가하는 것 == 나만의 자료구조

6개의 서로 다른 값을 가지는지 확인

- List대신 Set을 사용해 구현 가능함

```
public class Lotto {
    private static final int LOTTO_SIZE = 6;

    private final Set<LottoNumber> lotto;

    private Lotto(Set<LottoNumber> lotto) {
        if (lotto.size() != LOTTO_SIZE) {
            throw new IllegalArgumentException
               ("로또는 6개의 서로 다른 숫자를 입력해야 합니다.");
        }

        this.lotto = lotto;
    }
}
```

각 등수별 당첨된 로또 수

- Map을 통해 구현 가능함

```
public class LottoResult {
    private Map<Rank, Integer> result = new HashMap<>();

    public LottoResult() {
        for (Rank rank : Rank.values()) {
            result.put(rank, 0);
        }
    }

    public void putRank(Rank rank) {
        result.put(rank, result.get(rank) + 1);
    }
}
```

객체에 메시지를 보내라

상태 데이터를 가지는 객체에서 데이터를 꺼내려(get)하지 말고 객체에 메시지를 보내라.

당첨번호와 사용자 구매로또 비교 로직

다음과 같이 메시지를 보내는 경우 Lotto의 6개 숫자 값을 꺼내지 않아도 된다.

```
public class WinningLotto {
    private final Lotto lotto;
    private final int bonusNo;

    public WinningLotto(Lotto lotto, int bonusNo) {
        this.lotto = lotto;
        this.bonusNo = bonusNo;
    }

    public Rank match(Lotto userLotto) {
        int countOfMatch =
            userLotto.countOfMatch(lotto);
        boolean matchBonus =
            userLotto.isContains(bonusNo);
        return Rank.valueOf(countOfMatch, matchBonus);
    }
}
```

```
public class Lotto {
    [...]

    int countOfMatch(Lotto lotto) {
        int countOfMatch = 0;
        for (Integer number : numbers) {
            countOfMatch +=
                getMatchingCount(lotto, number);
        }
        return countOfMatch;
    }

    private int getMatchingCount(
        Lotto lotto, Integer number) {
        if (lotto.isContains(number)) {
            return 1;
        }
        return 0;
    }


    boolean isContains(int bonusNo) {
        return numbers.contains(bonusNo);
    }
}
```

각 등수별 당첨금 계산하기

enum에서도 값을 꺼내지(get) 말고 메시지를 보내 로직을 구현한다.

```java
public enum Rank {
    FIRST(6, 2_000_000_000), // 1등
    SECOND(5, 30_000_000), // 2등
    THIRD(5, 1_500_000), // 3등
    FOURTH(4, 50_000), // 4등
    FIFTH(3, 5_000), // 5등
    MISS(0, 0);

    private int countOfMatch;
    private int winningMoney;

    private Rank(int countOfMatch, int winningMoney) {
        this.countOfMatch = countOfMatch;
        this.winningMoney = winningMoney;
    }

    public long prize(int countOfMatchLotto) {
        return countOfMatchLotto * winningMoney;
    }

    [...]
}
```