아이템 34. int 상수 대신 열거 타입을 사용하라.

- 정수 열거 패턴

```
public static final int APPLE_FUJI = 0;
public static final int APPLE_PIPPIN = 1;
public static final int ORAGNE_NAVEL = 0;
```

  - 단점
    - 타입 안전 보장할 방법이 없다. (상수라도 클래스 내에서 값을 변경하기가 쉬움)
    - 정수 열거 패턴을 위한 별도의 namespace를 지원하지 않음. 그래서 보통 접두어로 구분 짓는다.

- 열거 타입

```
public enum Apple {FUJI, PIPPIN}
public enum Orange {NAVEL}
```

  - enum 타입은 생성자를 제공하지 않으므로 사실상 final
  - 타입안정성 제공
    - Apple 열거 타입을 매개변수로 받는 메서드가 있다면 건네 받은 참조는 Apple의 세 가지 값 중 하나임이 확실
  - 데이터와 메서드를 갖는 열거 타입

```
//최대한 짧게 구성
public enum Planet {
  EARTH(5.975e+24, 6.378e6);

  private final double mass; //질량
  private final double radius; //반지름
  private final double surfaceGravity; //표면중력

  private static final double G = 6.67300E-11; //중력 상수..
```

```
  Planet(double mass, double radius) {
     this.mass = mass;
     this.radius = radius;
     surfaceGravity = G * mass / (radius * radius);
  }

  public double mass() {return mass;}
  public double radius() {return radius;}
  public double surfaceGravity() {return surfaceGravity;}

  public double surfaceWeight(double mass){
     return mass * surfaceGravity;
  }
}
```

- 상수별 메서드 구현을 활용한 열거 타입 (상수별로 메서드 다르게 동작하게 할 때)

```
//열거 타입에 추상 메서드를 선언하고 각 상수별로 메서드 재정의 하여 사용
public enum Operation {
 PLUS { public double apply(double x , double y) {return x+y;}},
 MINUS { public double apply(double x, double y) {return x-y;}};

  public abstract double apply(double x, double y);
}
```

- 상수별 클래스 몸체와 데이터를 사용한 열거 타입

```
public enum Operation {
  PLUS("+") {public double apply(double x , double y) {return x+y;}},
  MINUS("-") {public double apply(double x , double y) {return x-y;}};

  private final String symbol;

  Operation(String symbol) {this.symbol = symbol;}

  @Override public String toString() {return symbol;}

  public abstract double apply(double x, double y);
}
```

- 전략 열거 타입 패턴 (열거 타입 상수 일부가 같은 동작을 공유 시 사용하면 편리)

```
enum PayrollDay{
   MONDAY(WEEKDAY), SUNDAY(WEEKEND);

    private final PayType payType;

    PayrollDay(PayType payType) {this.payType = payType;}

    int pay(int minutesWorked, int payRate) {
        return payType.pay(minutesWorked, payRate);
    }

    //enum 안에 enum ?! 전략 열거 타입
    enum PayType {
        WEEKDAY {
            int overtimePay(int minsWorked, int payRate) {
                return minsWorked<= MINS_PER_SHIFT ? 0 :
                        (minsWorked- MINS_PER_SHIFT) * payRate / 2;
            }
        },
        WEEKEND {
            int overtimePay(int minsWorked, int payRate) {
                return minsWorked* payRate / 2;
            }
        };

        abstract int overtimePay(int mins, int payRate);
        private static final int MINS_PER_SHIFT = 8 * 60;

        int pay(int minWorked, int payRate) {
            int basePay = minWorked * payRate;
            return basePay + overtimePay(minWorked, payRate);
        }
    }
}
```

- 그래서 열거 타입은 언제 쓰나요?
  - 필요한 원소를 컴파일타임에 다 알 수 있는 상수 집합이라면 항상! 열거 타입을 사용하자.
  - 열거타입에 정의된 상수 개수가 영원히 고정 불변일 필요는 없다.