Use EnumSet instead of bit fields

상황

열거된 값들이 주로 집합에서 사용될 경우, 기존에는 int enum pattern을 사용했습니다.(Item 34)

각각 bit의 1,10,100등에 해당하는 값으로 표현하는 것이죠.

```
// Bit field enumeration constants - OBSOLETE!
public class Text {
  public static final int STYLE_BOLD            = 1 << 0;
    public static final int STYLE_ITALIC        = 1 << 1;
    public static final int STYLE_UNDERLINE     = 1 << 2;
    public static final int STYLE_STRIKETHROUGH = 1 << 3;
  // Parameter is bitwise OR of zero or more STYLE_ constants
  public void applyStyles(int styles) { ... }
}
```

이와 같이 비트로 표현한다면 합집합과 교집합을 bit 연산으로 쉽게 표현할 수 있게 됩니다.

```
text.applyStyles(STYLE_BOLD | STYLE_ITALIC);
```

문제점

1. Item 34에서 설명했던 int enum pattern이 가진 문제를 그대로 가지고 있습니다.
2. 해석 난해

   비트 필드 값이 노출되면 해석하기 어려워 집니다. (1,2,4 등...)
3. 순회 어려움
4. 최대 몇 비트가 필요한지 API작성시 미리 결정해야 합니다.

해결방법 - EnumSet

java.util에서 EnumSet을 지원해준 뒤로는 쓸 이유가 전혀 없습니다.

장점

1. 비트로 표현할 필요없이 EnumSet이 적절히 잘 표현해줍니다.
2. Set 인터페이스를 완벽히 구현합니다.
3. 타입 안전합니다.
4. 다른 Set구현체와 함께 사용할수도 있습니다.

구현

내부가 비트 벡터로 표현되어 있고, 원소가 총 64개 이하라면 대부분 long 변수 하나로 표현할 수 있습니다.

removeAlldhk retailAll같은 대량 작업은 효율적으로 처리할 수 있는 산술 연산을 써서 구현했습니다.

수정된 코드

```
// EnumSet – a modern replacement for bit fields
public class Text {
  public enum Style { BOLD, ITALIC, UNDERLINE, STRIKETHROUGH }
  // Any Set could be passed in, but EnumSet is clearly best
  public void applyStyles(Set<Style> styles) { ... }
}
```

client 코드

EnumSet은 static factory를 제공해주기 때문에 간단히 값을 넘길 수 있습니다.

```
text.applyStyles(EnumSet.of(Style.BOLD, Style.ITALIC));
```

> 한편으로, 굳이 applyStyles를 Set으로 표현한 이유는 client가 EnumSet Type이 아닌 것을 넘길 수도 있기 때문에 interface로 받아주도록 합니다.