아이템 60) 정확한 답이 필요하다면 float 와 double 은 피하라

- float와 double 타입은 넓은 범위의 수를 빠르게 정밀한 `근사치` 로 계산하도록 설계되었기 때문에, **정확한 결과가 필요할 때는 사용하면 안 된다.**

금융 계산에는 BigDecimal, int 혹은 long을 사용하라

- BigDecimal 사용 예시

```
public static void main(String[] args) {
    final BigDecimal TEN_CENTS = new BigDecimal(".10");

    int itemBought = 0;
    BigDecimal funds = new BigDecimal("1.00");
    for(BigDecimal price = TEN_CENT; funds.compareTo(price) >= 0; price = price.add(TEN_CENTS)) {
        funds = funds.subtract(price);
        itemsBought++;
    }

    System.out.println(itemBought + "개 구입");
    System.out.println("잔돈(달러): " + funds);
}
// 프로그램의 실행결과는 사탕 3개를 구입한 후 잔돈은 0.39999999999999달러가 남는다고 나옴
```

- int 사용 예시 - Cent 로 단위 변환

```
public static void main(String[] args) {
    int itemBought = 0;
    int funds = 100;
    for(int price = 10; funds >= price; price += 10) {
        funds -= price;
        itemBought++;
    }

    System.out.println(itemBought + "개 구입");
    System.out.println("잔돈(달러): " + funds);
}
```

BigDecimal

- 불변**의 성질을 띠며, 임의 정밀도와 부호를 지니는 10진수**라고 표현
  - 임의 정밀도 : 기본적으로 큰 숫자를 배열에 나눠 담는 방식 / 큰 숫자 = [int] + [int] + [int] + [int] …
  - BigDecimal은 임의 정밀도를 나타내는 `unscaled value`와 32bit의 `scale`로 이루어져있다

> `unscaled value`는 정수부를 표현하고, `scale`은 소수점 아래 자릿수를 표현한다.
>
> ex) BigDecimal 3.14의 경우 `unscaled value`는 314이고 `scale`은 2가 된다.

- Java 언어에서 숫자를 정밀하게 저장하고 표현할 수 있는 유일한 방법
- BigDecimal 객체 간의 연산마다 새로운 객체 생성 → float나 double과 같은 기본 타입에 비해 사용하기가 훨씬 느림
  - int나 long이라는 대체제가 있지만, 실수를 표현할 수 없고 값의 범위가 비교적 제한된다는 점 때문에 **BigDecimal은 금융 관련 계산에서 필수적으로 사용됨**
- BigDecimal 초기화

```
// double 타입을 그대로 초기화하면 기대값과 다른 값을 가진다.
// 0.01000000000000000020816681711721685132943093776702880859375
new BigDecimal(0.01);
```

```
// 문자열로 초기화하면 정상 인식
// 0.01
new BigDecimal("0.01");

// 위와 동일한 결과, double#toString을 이용하여 문자열로 초기화
// 0.01
BigDecimal.valueOf(0.01);
```

- **BigDecimal**은 기본 타입이 아닌 오브젝트이기 때문에 특히, 동등 비교 연산을 유의해야 한다
  - equals() : unscaled value, scale 을 모두 비교
  - compareTo() : 소수점 맨 끝의 0을 무시하고 값만을 비교하고 싶을 때 사용

```
final BigDecimal b1 = new BigDecimal("7.10");
final BigDecimal b2 = new BigDecimal("7.1");

System.out.println(b1 == b2); // fale _ 주소값 비교
System.out.println(b1.equals(b2)); // false _ 7.10과 7.1은 논리적으로 같은 수일지라도, 소수점아래 자릿수가 다르므로 equals의 결과는 false
가 된다.
System.out.println(b1.compareTo(b2)); // 0
```

- **BigDecimal**의 애플리케이션 내부 연산과 저장소 말고도 신경써야할 것이 바로, 외부 서비스 간의 **API** 요청-응답 처리
  - JSON은 기본적으로 부동 소수점 숫자를 표현하기 위해 IEEE 754 부동 소수점 표현을 사용 → 이는 정확한 10진수를 보장 X
  - JSON을 다시 역직렬화하면 BigDecimal 값이 아닌 부동 소수점 숫자로 변환
    - 변환 과정에서 부동 소수점에서 발생하는 **반올림 및 정밀도 손실 문제가 발생**할 수 있음
    - 따라서 BigDecimal 값을 JSON에서 문자열로 처리하고, 역직렬화할 때 문자열에서 BigDecimal로 변환하는 것이 좋음

```
public class BigDecimalSerializationExample {

    // BigDecimal을 문자열로 직렬화하는 Serializer 클래스
    public static class BigDecimalSerializer extends JsonSerializer<BigDecimal> {
        @Override
        public void serialize(BigDecimal value, JsonGenerator gen, SerializerProvider serializers)
                throws IOException {
            gen.writeString(value.toString());
        }
    }

    // 문자열을 BigDecimal로 역직렬화하는 Deserializer 클래스
    public static class BigDecimalDeserializer extends JsonDeserializer<BigDecimal> {
        @Override
        public BigDecimal deserialize(JsonParser p, DeserializationContext ctxt)
                throws IOException, JsonProcessingException {
            String valueAsString = p.getValueAsString();
            return new BigDecimal(valueAsString);
        }
    }

    public static void main(String[] args) throws IOException {
        ObjectMapper objectMapper = new ObjectMapper();

        // BigDecimal 직렬화 및 역직렬화를 위한 모듈 생성
        SimpleModule module = new SimpleModule();
        module.addSerializer(BigDecimal.class, new BigDecimalSerializer());
        module.addDeserializer(BigDecimal.class, new BigDecimalDeserializer());
        objectMapper.registerModule(module);

        // 예제로 사용할 BigDecimal 객체 생성
        BigDecimal bigDecimalValue = new BigDecimal("123.456");

        // BigDecimal을 JSON 문자열로 직렬화
        String jsonString = objectMapper.writeValueAsString(bigDecimalValue);
        System.out.println("JSON String: " + jsonString);

        // JSON 문자열을 BigDecimal로 역직렬화
        BigDecimal deserializedValue = objectMapper.readValue(jsonString, BigDecimal.class);
        System.out.println("Deserialized BigDecimal: " + deserializedValue);
    }
}
```

결론

- 정확한 답이 필요한 계산에는 float나 double은 피하라.
- 소수점 추적은 시스템에 맡기고, 코딩 시의 불편함이나 성능 저하를 신경 쓰지 않겠다면 BigDecimal을 사용하라.
- 반면, 성능이 중요하고 소수점을 직접 추적할 수 있고 숫자가 너무 크지 않다면, int나 long을 사용하라.
- 9자리 십진수로 표현할 수 있다면 `int` 사용, 18자리 십진수로 표현할 수 있다면 `long` 을 사용하라.
  - 열여덟 자리를 넘어가면 `BigDecimal` 을 사용해야 함