1. 다형성

1. 다형성이라는 것은 많은 형태를 가질 수 있는 것을 의미한다.
2. 자식클래스가 부모클래스를 상속받게 되면 자식클래스는 자식클래스의 형태도 가지면서 부모클래스의 형태도 가지게 된다.
3. 부모클래스 형태의 변수에 자식클래스의 객체를 넣어서 사용하게 되면 하나의 변수로 다양한 기능을 만들 수 있다.
4. 기능의 다양화는 오버라이딩을 통해 이뤄진다. 부모클래스에 존재하는 메소드를 오버라이딩을 통해 자식클래스에서 다양한 기능으로 만들어서 부모클래스 형태의 변수로 같은 메소드를 호출했을 때 서로 다른 결과나 기능을 나타나게 할 수 있다.
5. 다형성을 구현하면 부모의 메소드는 호출되는 일이 없기 때문에 부모의 메소드는 추상화를 통해서 형태(껍데기)만 만들어놓고 상속받은 자식 클래스에서 오버라이딩을 통해서 구현한다.

2. 다형성의 장점

1. 유연성: 여러 객체를 하나의 변수로 처리할 수 있어서 유연하게 객체를 변경할 수 있다.
2. 확장성: 부모클래스를 상속받은 다양한 클래스를 새로 만들어도 같은 변수에 객체만 변경하면 되기때문에 클래스를 추가하여 확장하기에도 이점을 갖는다.
3. 코드의 가독성: 부모클래스 타입의 변수 하나만 선언해서 사용하며 일관된 방식으로 메소드 호출을 하기 때문에 가독성을 높일 수 있다.
4. 코드의 재사용성: 변수 하나로 여러개의 객체를 사용할 수 있기때문에 객체만 변경하면 코드를 재사용할 수 있다.
5. 유지보수성: 클래스가 추가되거나 기능이 변경될 때 수정해되는 부분이 다형성을 사용하지 않을 때보다 많이 줄어들게 된다.

3. 다형성의 단점

1. 성능하락: 다형성을 이용하면 객체의 타입을 찾아야하므로 일반적인 메소드의 호출보다 느리게 동작할 수 있다. 메모리 성능 많이 증가한 현시점에서는 크게 문제될 것이 없다. 성능하락이 일어나도 무시할 수 있는 수준이다.
2. 복잡성: 클래스들간의 관계가 많아질수록 관계를 이해하는 게 복잡해지고 코드를 분석하기가 어려워집니다. 적당한 관계만 생성하고 처음부터 클래스를 만들기 전에 클래스 설계와 구조를 생각하는 데 시간을 투자하는게 좋다.

3. 오버라이딩의 기능 오류: 다형성을 구현할 때 오버라이딩을 잘 못 사용하면 오버라이딩 기능이 오작동할 때가 있다.
4. 가독성: 관계의 레벨이 깊어지면 깊어질수록 한 번에 객체의 타입을 파악하기 힘들어진다.

4. super 키워드

1. 자식클래스에서 부모클래스의 멤버변수나 메소드, 부모의 생성자에 접근하거나 호출하려면 super라는 키워드를 사용한다.
2. 부모클래스의 멤버변수 접근방식

- 접근제어자가 protected이상(public, protected)일 때 : super.변수명으로 접근가능
- 접근제어자가 default일 때
    - 같은 패키지에 존재 : super.변수명으로 접근가능
    - 다른 패키지에 존재 : getter/setter를 통해 접근 super.getter/setter 사용.
- 접근제어자가 private일 때 : super.getter/setter.

3. 부모클래스의 메소드 호출 방식

- public, protected : super.메소드명();
- default : 같은 패키지에 존재하지 않으면 호출불가. 같은 패키지에 존재하면 super.메소드명();
- private : 부모클래스의 메소드 호출불가.

4. 부모클래스의 생성자 호출 방식

- super(), super(매개변수)로 기본생성자나 매개변수가 있는 생성자를 호출할 수 있다.
- 자식객체가 만들어 질 때 부모클래스의 인스턴스가 먼저 생성되기 때문에 자식클래스의 생성자에서 항상 처음으로 호출돼야 한다. 그 후에 남은 초기화를 처리한다.

```
public 자식클래스() {
    super(); -->항상 먼저 호출
    나머지 작업
}
```

5. 자바의 최상위 클래스 Object

1. 자바에서 최상위 클래스는 Object로 모든 클래스가 만들어질 때 Object를 상속받아 만들어진다.

2. 모든 클래스는 Object를 상속받기 때문에 Object 구현되어 있는 메소드(toString(), notify(), notifyAll() ...)를 사용할 수 있다.
3. 다형성의 측면에서 Object를 타입으로 지정하면 모든 클래스들을 넣어서 사용할 수 있기 때문에 엄청난 확장성을 가지게 된다.

`Map<String, Object>` => `value`의 타입을 `Object`로 정했기 때문(