**자바 인터페이스 사용 이유**

**1. Cooperation (협업)**

예를 들어 우리는 한국인을 위한 번역기를 만들어 볼거에요. 아 전세계 언어가 너무 많아요ㅠㅠ 그래서 업무를 나눠봤어요.  
"너는 영어 번역 클래스를 만들어, 난 중국어 번역 클래스를 만들게. 준비~ 시작!" 하고 각자 다 만들어서 결과물을 공유했어요. 아래와 같은 결과물이 나왔어요.

/\*\*

\* **@author** 팀원1

\*/

**public** **class** **EnglishTrans** {

**public** String **koreanToEnglish**(String str) {

//**TODO:** 한국어를 영어로 번역

}

**public** String **englishToKorean**(String str) {

//**TODO:** 영어를 한국어로 번역

}

}

다른 친구가 만든 클래스를 볼게요.

/\*\*

\* **@author** 팀원2

\*/

**public** **class** **ChineseRendering** {

**public** String **translateChinese**(String str) {

//**TODO:** 한국어를 중국어로 번역

}

**public** String **translateKorean**(String str) {

//**TODO:** 중국어를 한국어로 번역

}

}

개인의 개성이 적나라하게 반영되었네요.

메서드명은 물론이고, 클래스명도 달라요.  
번역 기능을 만드는 일만으로도 너무 벅차서, 한 명의 친구를 새로 불러다가 "야, 우리 이렇게 클래스들 만들어 놨는데, 알려줄테니깐 이것들 사용해서 넌 화면에 제공할 API를 만들어주라"하고 시켰어요.  
새로온 친구는 클래스명을 받고 개발을 시작했어요.

EnglishTrans trans = **new** EnglishTrans();

String str = trans.koreanToEnglish(str);

// **TODO:** Business Login

...

ChineseRendering rendering = **new** ChineseRendering();

String str = rendering.translateChinese(str);

// **TODO:** Business Login

...

어떤가요?

이번엔 인터페이스를 사용해볼게요.

**public** **interface** **Translation** {

// 한국어를 번역하다

**public** String **translate**(String str);

// 한국어로 번역하다

**public** String **translateInto**(String str);

}

위와 같이 공통적으로 implements할 인터페이스를 만들어요.  
그리고 각자 만들 클래스에서 Translation 인터페이스를 implements 할거에요.  
그럼 강제로 해당 메소드를 구현해야 해요.

/\*\*

\* **@author** 팀원1

\*/

**public** **class** **EnglishTranslation** **implements** **Translation** {

@Override

**public** String **translate**(String str) {

//**TODO:** 한국어를 번역

}

@Override

**public** String **translateInto**(String str) {

//**TODO:** 한국어로 번역

}

}

다른 친구도 인터페이스를 구현할거에요.

강제적으로 정해진 이름에 맞춰 구현할 수 밖에 없게 되요.

/\*\*

\* **@author** 팀원2

\*/

**public** **class** **ChineseTranslation** **implements** **Translation** {

@Override

**public** String **translate**(String str) {

//**TODO:** 한국어를 번역

}

@Override

**public** String **translateInto**(String str) {

//**TODO:** 한국어로 번역

}

}

호출하는 부분도 볼게요.

Translation trans = **new** EnglishTranslation();

String str = trans.translate(str);

// **TODO:** Business Login

...

Translation trans = **new** ChineseTranslation();

String str = trans.translate(str);

// **TODO:** Business Login

...

이제 어떤가요?

사용할 때 인터페이스를 이용해서 원하는 언어에 따라 클래스를 골라서 객체를 생성해서 쓰면 되요.  
통째로 로직을 북붙해서 재활용하기도 편해요.

물론 인터페이스를 만들지 않아도 서로 약속하고 맞출 수도 있겠지만 오타가 발생하거나 작업량이 많아지면 잊어버릴 수도 있구요..

굳이 협업이 아니더라도 위와 같은 구조를 가지고 가는 것이 좋겠네요.

**2. Easy to replace (교체 용이)**

이번엔 암호화 기능을 개발해볼거에요.

**public** **interface** **Crypto** {

// 암호화

**public** String **encrypt**(String str);

// 복호화

**public** String **decrypt**(String str);

}

위와 같이 인터페이스로 표준을 잡고 아래와 같이 구현체를 개발할거에요.

**public** **class** **SHA512Crypto** **implements** **Crypto** {

@Override

**public** String **encrypt**(String str) {

// **TODO:** SHA512 방식으로 암호화

}

@Override

**public** String **decrypt**(String str) {

// **TODO:** SHA512 방식으로 복호화

}

}

아 SHA512 방식은 더 이상 안전하지 않는 것 같아요. 다른 방식으로 바꾸고 싶네요.  
클래스를 개발해서 이름만 교체하면 되니 아주 간단하네요.

교체할 때에는 객체를 생성하는 클래스만 변경해주면 아래의 로직은 일체 수정할 일이 없어요.

Crypto crypto = **new** SHA512Crypto(); // 기존 SHA512 방식

String enc = crypto.encrypt("1234");

교체

Crypto crypto = **new** SuperPowerCrypto(); // 신규 개발한 암호화 방식

String enc = crypto.encrypt("1234");

**3. Multiple inheritance (다중 상속)**

이 부분은 실제 자바에서 사용하고 있는 부분을 볼게요.

**public** **class** **HashMap**<**K**,**V**> **extends** **AbstractMap**<**K**,**V**>

**implements** **Map**<**K**,**V**>, **Cloneable**, **Serializable** {

}

기본적으로 Map이니깐 Map 인터페이스를 구현했네요.  
Cloneable로 인스턴스가 복제 가능하도록 하였고 Serializable로 직렬화했어요  
기본적으로 Cloneable과 Serializable은 구현할 필요없이 자바가 내부적으로 해줘요.  
여튼, 필요한 인터페이스를 상속받아서 구현하면 여러 기능들을 구조적으로 포함할 수 있어요.

결론

이외에도 인터페이스를 사용하는 여러 디자인 패턴이 있어요.  
상세히 알고 싶으시다면, 이 부분은 자바 디자인 패턴을 따로 공부하시면 되요.  
  
인터페이스는 잘 사용하면 프레임워크나 배포되는 공용 컴포넌트, 유틸 등을 개발할 때 아주 효과적이에요.  
다만, 전제가 있어요. **잘 사용해야되요.**  
  
반대로, 일반 웹 개발 프로젝트에서는 오히려 복잡도만 증대시키는 상황이 많아요.  
다시 말씀드리지만, 인터페이스는 **잘 사용해야되요.**  
많은 사람들이 불편해하고 필요성을 못느낀다면 과감히 버리는게 맞아요.  
  
결론은? **잘 사용해야되요.**