08/03 안드로이드 교육 - 1 (패스트캠퍼스)

인터페이스 (Interface) - '이것이라면 반드시 이런 기능은 있어야 한다'라는 의미로 생각
- 인터페이스는 어떠한 기능의 이름은 명시해주지만, 구체적인 내용은 인터페이스를 구현하는 클래스에 일임
- 인터페이스 안의 함수 구현은 인터페이스를 상속하는 클래스에 일임하므로 함수 내용을 기입하는 중괄호가 없음

ex.

```
Interface Human {
public void eat();
}

class Father implements Human {
public void eat() {
"많이 먹는다"
}
}
class Mother implements Human {
public void eat() {
"적게 먹는다"
}
}
class Son implements Human {
public void eat() {
"적당히 먹는다"
}
}
```

인터페이스의 장점
1. 인터페이스를 구현하는 클래스에게 인터페이스가 가지고 있는 함수의 구현을 강제할 수 있음
2. 사용자와 구현자 간의 협업이 가능함
- 구현자가 클래스의 기능을 구현하는 동안 사용자는 가짜 클래스를 만들어서 개발을 진행하고 향후 구현된 클래스를 사용
- 구현자와 사용자가 모두 인터페이스를 기준으로 클래스를 구현/사용하므로 이러한 협업이 가능함

인터페이스를 구현하고 자식 클래스에서 implements로 호출한 후,
아래에 우클릭 Generate - Override Methods 선택하면 인터페이스의 기능들을 쉽게 불러올 수 있음 (단축키: Ctrl + O)

접근제어자 - 해당 부분의 접근을 어디까지 허용할 지 결정하는 키워드
- 자바의 4가지 종류의 접근제어자 : Private -> Defalut -> Protected -> Public (이 순으로 허용 범위가 넓어짐)
- 접근제어자는 클래스 / 인터페이스 / 변수 / 함수에 적용할 수 있음

Private : 해당 클래스 안에서만 접근이 가능함 (클래스 안의 변수를 Private으로 설정하면 해당 변수는 속해있는 클래스 내에서만 접근이 가능)

Default : 접근제어자를 설정하지 않으면 기본적으로 설정되는 접근제어자, 해당 패키지 내에서만 접근이 가능함
Protected : 해당 패키지 내부 및 해당 클래스를 상속받은 클래스까지 접근을 허용
Public : 어떤 클래스에서든지 접근이 가능함

ex. 다른 클래스에서 변경되면 안되는 변수이지만, 다른 클래스에서 출력은 하고싶은 경우
```
private int money = 1000;
public int account() {
return money;
}
```

*Static : 프로그램 어떤 곳에서도 접근이 가능함
프로그램이 시작되면 Static으로 설정된 부분을 메모리에 올리게 됨
모든 부분을 Static으로 설정하면 프로그램 효율이 떨어짐

제너릭 (Generic)
- 인스턴스를 생성할 때, 인스턴스에서 사용될 데이터의 타입을 정해주는 역할
- 타입의 안전성을 높힐 수 있고, 형변환을 할 필요가 없음
ex.
제너릭을 사용하지 않는 경우
```
ArrayList arrayList = new ArrayList();
arrayList.add("fast");
String data = (String)arrayList.get(0);
```
*(String) -> String으로 형변환한다는 의미

제너릭을 사용하는 경우
```
ArrayList arrayList = new ArrayList<String>();
arrayList.add("fast");
String data = arrayList.get(0);
```
*<String> -> ArrayList에 들어갈 수 있는 데이터의 타입을 String으로 지정