다른 형태의 클래스

일반적으로 잘 사용되지 않는 형태의 클래스 구성 방법

Inner Class(내부클래스, 중첩클래스) 형태와 Anonymuse Class(익명 클래스)가 있다.

#01. Inner Class

클래스 안에 또 다른 클래스를 포함시켜, 클래스의 구조를 내부적으로 확장하는 방법.

클래스가 갖는 자료(=변수)의 계층화가 필요한 경우 사용.

```
public class HelloWorld {
  // 일반 inner class
  public class Korean { ... }

  // static inner class
  public static class English { ... }
}
```

1) 일반 Inner Class

객체를 생성하기 위해서 상위 클래스의 객체를 통한 접근이 필요하다.

```
HelloWorld h = new HelloWorld();
HelloWorld.Korean k = h.new Korean();
```

2) static inner class

객체를 생성하기 위해서 상위 클래스의 객체가 필요하지 않고, 상위 클래스의 이름을 통해 접근한다.

```
HelloWorld.Korean k = new HelloWorld.Korean();
```

#02. Anonymouse 클래스

인터페이스나 추상 클래스에 대한 상속이 전제가 되는 경우.

인터페이스나 추상 클래스를 상속받는 클래스에 대한 객체가 단 한곳에서만 사용될 경우 클래스 파일을 생성하지 않고 직접적으로 객체를 생성하는 문법 표현

아래와 같이 인터페이스가 존재할 경우

```
public interface Foo {
  public void bar();
}
```

일반적이라면 인터페이스를 상속받는 구현체 클래스를 정의하는 것이 정석이다.

```
public class Bar implments Foo {
    @Override
    public void bar() {
        ...
    }
}

// 선언은 부모, 할당은 자식
Foo b = new Bar();
b.bar();
```

위의 형태에서 `Bar` 클래스를 사용하는 경우가 단 한곳만 존재한다면 별도의 클래스 파일을 생성하는 것은 비효율적일 수 있다.

익명클래스를 사용하기 위해서는 Interface에 대한 객체를 직접 할당한다.

```
Foo f = new Foo();
```

하지만 `Foo`는 인터페이스이므로 직접 객체를 할당할 수 없다. (위 코드는 에러)

인터페이스는 반드시 상속을 통해서만 사용 가능하므로 `new Foo()` 구문 오른쪽에 `Foo` 인터페이스를 상속받는 이름 없는 클래스를 정의하기 위해 `{}`를 명시한다.

```
Foo f = new Foo() { ... 이 안이 이름 없는 구현체 클래스 ... };
```

인터페이스를 상속받는 구현체는 인터페이스의 메서드를 모두 Override 해야 하므로 `{}`안에서 Override를 수행한다.

```
Foo f = new Foo() {
    @Override
    public void bar() {
        ...
    }
};
```