아이템3: 연산 또는 액션을 전달할 때는 인터페이스 대신 함수 타입을 사용하라

SAM과 함수 타입

대부분의 프로그래밍 언어에서는 액션이나 연산을 전달할 때 SAM(Single-Abstract method)를 이용한다.

```
interface OnClick{
  fun clicked(view: View)
}

fun setOnClickListener(listener: OnClick) {..}

// 실제 전달: 인터페이스를 객체를 생성해서 잔달한다.
setOnClickListener(object: OnClick {
  override fun clicked(view: View)
})
```

이런 코드는 함수 타입으로 변경하면 많은 자유를 얻을 수 있다.

```
// 연산 또는 액션을 전달받는 함수
fun setOnClickListener(listener: (View) → Unit) {

}

setOnClickListener {} // 람다 표현식으로 전달
setOnClickListener (fun(view){}) // 익명 함수로 전달
setOnClickListener(::println) // 함수 레퍼런스로 전달
setOnClickListener(this::showUsers) // 제한된 함수 레퍼런스로 전달

class ClickListener: (View) -> Unit {
  override fun invoke(view: View{
  }
}

setOnClickListener(ClickListener()) // 선언된 함수 타입을 구현한 객체로 전달
```

SAM이 아규먼트에 이름을 붙일 수 있다는 장점이 있다고 말하는 사람도 있지만, 함수 타입도 typealias를 이용해서 이름을 붙일 수 있다.

함수 타입은 ide를 통해서 지원을 받거나, typeAlias를 이용해서 별칭을 갖는 등의 자유도를 보장받는다.

```
typealias OnClick2 = (abc:View) -> Unit

setOnClickListenerInterface(object :OnClickListener{
        override fun onClick(view: View) {
            TODO("Not yet implemented")
        }
    })
// 함수타입 얼라이어스
setOnClickListener(object : OnClick2 {
    override fun invoke(abc: View) {
        TODO("Not yet implemented")
    }
})
```

추가로, 함수 타입을 사용하면 아규먼트 분해도 사용할 수 있다.

- 리스너 두개로 분해할 수는 있긴 하다

```
class CalendarView {
  var listener: Listener? = null
  interface Listener {
    fun onDateClicked(date: Date)
    fun onPageChanged(date: Date)
  }
}

// 아래처럼!!
class CalendarView{
  var onDateClicked: ((date:Date) -> Unit)? =null
  var onPageChanged: ((date:Date) -> Unit)? =null
}
```

↔ 한 대상에 대해 여러 액션을 전달받을 때, 자바에서는 인터페이스를 기반으로 구현했다. API의 소비자 관점에서는 함수 타입을 따로 가지는 편이 사용하기 쉽다.

소결

1. 인터페이스를 사용해야 하는 특별한 이유가 없다면, 함수 타입을 활용하자. (자유도 높음)
2. 함수 타입은 다양한 지원을 받을 수 있으며 코틀린에서 이미 널리 사용되고 있음

언제 SAM을 사용해야 할까?

다른 언어에서 사용할 클래스를 사용하는 경우

- Java의 경우 함수 타입으로 만들어진 클래스는 타입 별칭, IDE 지원을 제대로 받지 못한다.
- 다른 언어에서 코틀린의 함수 타입을 사용하려면, Unit을 명시적으로 리턴하는 함수가 필요하다.

```
class CalendarView {
    var onDateClicked: ((string: String) -> Unit)? = null
    var onPageChanged: OnDateClicked? = null

}

interface OnDateClicked {
    fun onClick(string: String)
}
//Java
c.setOnDateClicked(str -> Unit.INSTANCE);
c.setOnDateClicked(str -> {}); //error
c.setOnPageChanged(str -> Unit.INSTANCE); // error
c.setOnPageChanged(str -> {});
```