#11 오버라이딩과 추상화

✔ 오버라이딩

상속 시에는 기본적으로 수퍼클래스에 있는 함수와 같은 이름과 형태를 가진 함수는 서브클래스에서는 만들 수가 없다. 하지만 수퍼클래스에서 허용만 한다면 오버라이딩 방법으로 서브클래스에서 같은 이름과 형태로 된 함수의 내용을 다시 구현할 수 있다.

```
fun main(){
    var t = Tiger()                    — ④
    t.eat()                            — ⑤
}
open class Animal {                    — ①
    fun eat(){                         — ②
        println("음식을 먹습니다.")
    }
}
class Tiger: Animal()                  — ③
```

① 상속이 가능하게 open 된 **Animal** 클래스 만들기
② 클래스 내에 음식을 먹을 때 실행할 **eat** 함수 만들기 - "음식을 먹습니다." 라고 출력하는 함수
③ **Tiger** 서브클래스 만들어 **Animal** 클래스 상속받기
④ **tiger** 클래스의 인스턴스 만들기
⑤ **eat** 실행하기 - 음식을 먹습니다 출력

그런데 **Tiger** 클래스에서는 "음식을 먹습니다." 대신 "고기를 먹습니다." 라고 출력하고 싶은데 이미 **eat** 함수는 **Animal**에서 "음식을 먹습니다"라고 구현되어있기 때문에 서브클래스에서는 함수를 재구현할 수 없다.

```
fun main(){
    var t = Tiger()
    t.eat()                            — ③
}
open class Animal {
    open fun eat(){                    — ①
        println("음식을 먹습니다.")
    }
}
class Tiger: Animal(){
    override fun eat(){                — ②
        println("고기를 먹습니다.")
    }
}
```

① 이때, **Animal** 클래스에서 **eat** 함수 앞에 **open**이 붙는다면 **Tiger**에서 재구현이 허용 된다
② 수퍼 클래스에서 **open**이 붙은 함수는 서브클래스에서 **override**를 붙여 재구현할 수 있다.
③ 이제 **Tiger**에서 **eat**을 다시 수행해보면 "음식을 먹습니다." 대신 "고기를 먹습니다."를 출력하는 것을 확인할 수 있다.

위의 사례에서는 이미 수퍼클래스에서 구현이 끝난 함수를 오버라이딩을 통해 재구현 하는 경우를 소개했다면 이번엔 오버라이딩과 다르게 수퍼클래스에서 함수의 구체적인 구현은 없고 단지 Animal의 모든 서브클래스는 eat이라는 함수가 반드시 있어야 한다는 점만 명시하여 각 서브클래스가 비어있는 함수의 내용을 필요에 따라 구현하도록 하려면 **추상화**라는 개념을 사용한다.

✔ 추상화

선언부만 있고 기능이 구현되지 않은 추상함수 그리고 추상함수를 포함하는 추상클래스라는 요소로 구성된다. 그럼 추상함수를 포함한 추상클래스를 만들고 서브클래스에서 상속받아 구현까지 하는 과정을 알아보자

```
fun main(){
    var r = Rabbit()                    — ⑥
    r.eat()
    r.sniff()
}
abstract class Animal {                 — ①
   abstract fun eat()                   — ②
   fun sniff() {                        — ③
       println("킁킁")
    }
}
class Rabbit: Animal(){                 — ④
    override fun eat(){                 — ⑤
       println("당근을 먹습니다.")
    }
}
```

< 수행결과 >
당근을 먹습니다.
킁킁

① **Animal** 클래스를 다시 만들되 그 앞에 **abstract**를 붙여준다.
② 추상함수인 **eat** 함수를 만드는데 그 앞에 **abstract**를 붙이고 함수의 내용은 적지 않는다.
*추상함수는 비어있는 껍데기라고 생각하기
③ 냄새를 맡는 **sniff**라는 일반함수 추가하기
abstract를 붙인 추상클래스는 일부 함수가 구현되지 않은 '미완성 클래스'이기 때문에 단독으로는 인스턴스를 만들 수 없다. 따라서 반드시 서브클래스에서 상속을 받아 **abstract** 표시가 된 함수들을 구현해줘야 한다.
④ **Rabbit** 클래스 만들어 **Animal**을 상속받기
⑤ **eat**이라는 추상함수의 실제 동작이 되는 구현부를 만들어보자 **override** 키워드 잊지 않기
⑥ **Rabbit**에 인스턴스를 만들어 **eat**과 **sniff**를 수행하면 당근을 먹고 킁킁거리는 토끼가 출력된다.

✔ **인터페이스** 추상화를 하는 또 다른 방법

다른 언어에서 인터페이스는 추상함수로만 이루어져 있는 '순수 추상화 기능'이라고 알고 있을 것이다. 코틀린에서는 인터페이스 역시 추상함수와 일반함수 모두 가질 수 있다.

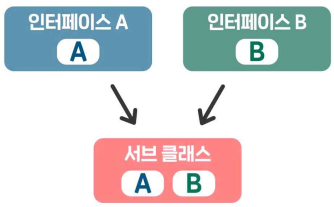
다만 추상함수는 생성자를 가질 수 있는 반면 인터페이스는 생성자를 가질 순 없다.

인터페이스에서

구현부가 있는 함수 -> open 함수로 간주

구현부가 없는 함수 -> abstract 함수로 간주

이 때문에 별도의 키워드가 없어도 포함된 모든 함수를 서브클래스에서 구현 및 재정의가 가능하다.



또한 한 번에 여러 인터페이스를 상속받을 수 있으므로 좀 더 유연한 설계가 가능하다.

```
fun main(){                                  — ⑤
    var d = Dog()
    d.run()
    d.eat()
}
interface Runner {                           — ①
    fun run()
}
interface Eater {                            — ②
    fun eat(){
        println("음식을 먹습니다.")
    }
}
class Dog : Runner, Eater {                  — ③
    override fun run() {                     — ④
        println("우다다다 뜁니다.")
    }
    override fun eat() {
        println("허겁지겁 먹습니다.)
    }
}
```

< 수행결과 >

우다다다 뜁니다.

허겁지겁 먹습니다.

① 먼저 Runner라는 인터페이스를 만들고 run이라는 함수를 구현부 없이 만들어보자
② Eater라는 인터페이스를 만들고 “음식을 먹습니다.”를 출력하는 eat 함수를 만들자
③ Dog이라는 클래스로 두 인터페이스를 상속받으려면 클래스 선언 뒤에 콜론을 붙이고 쉼표로 두 인터페이스를 구분하여 표기해주면 된다.
④ 그리고 구현부가 없던 run 함수에는 override를 붙여 “우다다다 뜁니다”를 출력하도록 구현해주고 이미 구현이 있는 eat이라는 함수는 override를 붙여 “허겁지겁 먹습니다.”를 출력하도록 재구현 해준다.
⑤ 그러면 Dog는 두 인터페이스의 형식을 모두 물려받아 사용하는 서브클래스가 된다.

✔ 주의사항
여러 개의 인터페이스나 클래스에서 같은 이름과 형태를 가진 함수를 구현하고 있다면 서브클래스에서는 혼선이 일어나지 않도록 반드시 오버라이딩하여 재구현해주어야 한다.

오버라이딩은 이미 구현이 끝난 함수의 기능을 서브클래스에서 변경해야 할 때
추상화는 형식만 선언하고 실제 구현은 서브클래스에 일임할 때 사용하는 기능이며
인터페이스는 서로 다른 기능들을 여러 개에 물려주어야 할 때 유용한 기능입니다