소개

> 이 장에서는 스칼라의 핵심 데이터와 변수 타입을 다룬다.
> 먼저, literal, value, variable, type의 정의부터 알아보자.

- 리터럴은 숫자 5, 문자 A, 텍스트 "Hello, World"처럼 소스 코드에 바로 등장하는 데이터다.
- 값은 불변의 타입을 갖는 저장 단위다. 값은 정의될 때 데이터가 할당될 수 있지만, 절대 재할당될 수는 없다.
- 변수는 가변의 타입을 갖는 저장 단위다. 변수는 정의 시 데이터를 할당할 수 있으며, 언제라도 데이터를 재할당할 수도 있다.
- 타입은 여러분이 작업하는 데이터의 종류로, 데이터의 정의 또는 분류를 의미한다. 스칼라의 모든 데이터는 특정 타입에 대응하며, 모든 스칼라 타입은 그 데이터를 처리하는 메소드를 갖는 클래스로 정의된다.

스칼라에서 값과 변수에 저장된 데이터를 더 이상 사용하지 않으면 JVM의 가비지 컬렉션이 자동으로 할당을 취소한다. 우리가 직접 할당 취소할 필요도, 그럴 능력도 없다.

변수 선언하기

- 스칼라의 값 구문 `val <이름>: <타입> = <리터럴>` 로 정의하며, 이 형태에 따라 타입이 `Int`인 이름 x를 만들고 여기에 숫자 5를 할당해보자.
- `scala> val x: Int = 5`

```
scala> x
x: Int = 5

scala> x * 2
res1: Int = 10

scala> x / 5
res2: Int = 1
```

- 이 '결과값'은 명시적으로 정의한 여느 값들과 마찬가지로 사용할 수 있다.
- 여기에 값 res0와 res1을 곱하면 그 결과로 값 50이 반환되고, 새로운 값 res3에 저장된다.

```
scala> res0 * res1
res3: Int = 50
```

- 이제 변수들을 가지고 작업해보자.
- 값과 달리 변수는 변경할 수 있으며, 새로운 값을 재할당할 수도 있다.
- 변수를 이용하여 작업하는 것을 예로 들어보면 다음과 같다.

```
scala> var a: Double = 2.72
a: Double = 2.72

scala> a = 355.0 / 113.0
a: Double = 3.1415929203539825

scala> a = 5
a: Double = 5.0
```

- double-precision floating-point number인 Double 타입을 가지는 변수 a를 정의하였다.
- 그리고 a는 변수이므로 여기에 다른 값을 재할당할 수 있다.

> 여기서는 간단하게 스칼라에서 값, 변수, 리터럴을 사용하는 법에 대해 소개하였다.
> 이 장의 나머지 부분에서는 각 주제별로 자세하게 다룰 것이다.

값

값(value) 는 불변의, 타입을 갖는 스토리지 단위이며, 관례적으로 데이터를 저장하는 기본적인 방법이다.
`val` 키워드를 사용하여 새로운 값을 정의할 수 있다.

- 구문: 값 정의
  - `val <식별자>[: <타입>] = <데이터>`
- 값은 이름과 할당된 데이터 모두 필요하지만, 명시적 타입이 있어야 하는 것은 아니다.
- 타입이 지정되지 않았다면(즉, ': <타입>' 구문이 포함되지 않았다면), 스칼라 컴파일러는 할당된 데이터를 기반으로 타입을 추론한다.
- 스칼라 REPL에서 값을 정의하는 몇 가지 예를 들면 다음과 같다.

```
scala> val x: Int = 20
x: Int = 20

scala val greeting: String = "Hello, World"
```

```
greeting: String = Hello, World

scala> val atSymbol: Char = '@'
atSymbol: Char = @
```

- 구문 다이어그램을 통해 값 정의에서 타입을 지정하는 것이 선택사항임을 알아챘을 것이다.
- 값 할당을 기반으로 그 타입을 추론하는 것이 가능한 상황이라면, 값 정의에서 타입을 빼도 된다.
- 스칼라 컴파일러는 할당된 값을 보고 그 값의 타입을 알아차릴 것이다.
- 이를 타입추론(Type Inference) 이라고 한다.
- 값을 타입 없이 정의하더라도 타입이 없는 것이 아니며, 마치 타입을 지정하여 정의한 것처럼 적절한 타입을 할당한다.
- 타입을 지정하지 않은 예를 보자.

```
scala> val x = 20
x: Int = 20

scala> val greeting = "Hello, World"
greeting: String = Hello, World

scala> val atSymbol = '@'
atSymbol: Char = @
```

- 이 예제에서 값은 결국 타입이 명시적으로 기술되었을 때와 동일한 타입을 갖는다.
- 스칼라 컴파일러는 REPL을 통해 리터럴 20은 타입 Int에, 리터럴 "Hello, World"는 String에, 리터럴 @은 타입 Char에 대응함을 추론할 수 있다.
- 스칼라의 타입 추론을 사용하면 명시적으로 값의 타입을 작성할 필요가 없기 때문에 코드를 작성할 때 유용하다.
- 다만, 코드의 가독성을 떨어뜨리지 않는 범위에서 사용해야 한다.

- 누군가 작성한 코드를 읽으면서 그 값의 타입이 무엇인지 알아볼 수 없다면, 값 정의에 명시적 타입을 포함하는 것이 더 낫다.

- 타입 추론이 데이터를 저장하는 데 사용할 올바른 타입을 추론하겠지만, 설정한 명시적인 타입을 대체하지는 않는다.

- 초깃값과 호환되지 않는 타입으로 값을 정의하면 컴파일 에러가 발생한다.

```
scala> val x: Int = "Hello"
<console>:7: error: type mismatch;
 found   : String("Hello")
 required: Int
        val x: Int = "Hello"
                     ^
```

변수

컴퓨터 과학에서 용어 변수(variable) 는 일반적으로 값을 저장하고 그 값을 가져올 수 있도록 할당되거나 예약된 메모리 공간에 대응하는 유일한 식별자를 의미한다. 메모리 공간이 예약되어 있는 동안에는 새로운 값을 할당할 수 있다. 따라서 메모리 공간의 내용은 동적이며 가변적(variable) 이다.

스칼라에서는 관례상 변수보다 값을 선호하는데, 이는 값을 사용하면 소스 코드가 안정적이며 예측할 수 있기 때문이다. 값을 정의하면 다른 어떤 코드에서 접근하더라도 같은 값을 유지한다. 코드 시작 부분에 값을 할당하면 코드 마지막 부분까지 값이 변경되지 않기 때문에 코드를 읽고 디버깅하는 일이 더 쉽다.
마지막으로, 애플리케이션 수명 기간에 사용할 수 있거나 동시 또는 멀티 스레드 코드에서 접근 가능한 데이터로 작업하는 경우, 변하지 않는 값은 예기치 않은 시점에 변경될 수도 있는 가변 데이터보다 더 안정적이며 에러가 발생할 경우도 적다.