Scala 용어

책을 볼 때에 용어를 아는 것이 도움이 되고, 이후 찾아보기 쉽게 하기 위해서 책에 있는 내용을 일부 변경하여 블로그에 옮겨 적습니다.

출처 : Programming in Scala 부록B 용어해설

1급 계층 함수(first-class function)(1급 함수)

- 스칼라는 1급 계층 함수를 지원
- 함수를 (x:Int) ⇒ x + 1 같은 함수 리터럴 문법으로 표현할 수 있음
- 함수를 객체로 표현할 수 있음
- 이런 객체를 함수 값(function value)이라 함

for 내장 (for comprehension)

- for 표현식

JVM 자바 가상 머신 (Java Virtula Machine)

- 스칼라 프로그램은 JVM 위에서 실행됨

런타임(runtime)

- 실행 시점과는 다른 의미
- 실행 시점에 프로그램을 실행하기 위한 환경

새도우 Shadow

- 지역 변수를 새로 선언하면 그 변수 선언이 있는 스코프를 둘러싼 스코프에 있는 같은 이름을 접근하지 못함

값 value

- 스칼라에서 어떤 계산이나 표현식의 결과 값
- 스칼라에서는 모든 값이 객체
- 값이라는 용어는 기본적으로 메모리에 있는 객체의 이미지를 의미

값에 의한 호출 파라미터 by-value parameter

- 파라미터의 타입 앞에 => 이 없는 경우에는 값에 의한 호출 파라미터 함
- 값에 의한 호출을 할 경우, 함수에 파라미터를 전달하기 전에 인자를 계산
- 값에 의한 호출 파라미터는 이름에 의한 호출 파라미터와 대조적

이름에 의한 호출 파라미터 by-name parameter

- 파라미터 타입 앞에 => 가 있는 경우 그 파라미터를 이름에 의한 호출 파라미터라 함
- (x: ⇒ Int)
- 이름에 의한 호출을 할 경우, 그 파라미터에 전달한 인자는 해당 메소드 호출 전에 계산하지 않고 메소드 내부에서 해당 이름을 사용하는 시점에 계산
- 어떤 파라미터가 이름에 의한 호출이 아니라면, 모두 값에 의 한 호출 임

값 타입 value type

- 값 타입은 AnyVal의 서브클래스
- Int, Double, Unit 등
- 이 용어의 의미는 스칼라 소스 코드 수준에서 성립
- 실행 시점에 자바 원시 타입에 대응하는 값 타입의 인스턴스는 원시 타입의 값이나 java.lang.Integer 같은 래퍼 타입(wrapper type)의 인스턴스로 구현
- 시간이 지남에 따라 런타임은 값타입의 인스턴스를 원시 타입이나 래퍼 타입 사이에서 변환해줌
- (박싱boxing 하거나 언박싱unboxing)

결과 result

- 스칼라 프로그램의 모든 표현식은 결과를 내놓음
- 스칼라의 모든 식이 내놓는 결과를 객체

결과 타입 result type

- 메소드의 결과 타입은 해당 메소드를 호출한 결과 만들어지는 값의 타입
- 자바에서는 이를 반환 타입 result type 이라 함

경로 의존적 타입 path-dependent type

- swiss.cow.Food 같은 타입을 말함
- 이때 swiss.cow는 어떤 객체를 가리키는 경로임
- 이 타입의 의미는 접근 시 사용하는 경로에 따라 달라짐
- swiss.cow.Food 와 fish.Food 가 있을 때, 이 둘은 각기 다른 타입임

공변성 covariant

- 타입 파라미터 앞에 덧셈 기호(+)를 추가하면 클래스나 트레이트의 타입 파라미터에 공변 표기를 붙일 수 있음
- 클래스나 트레이트를 표기가 붙은 파라미터 타입과 공변적인 타입으로 다룸
- 서브타입 관계가 타입 파라미터의 서브타입 관계와 같은 방향으로 성립한다는 뜻
- List 는 타입 파라미터에 대해 공변적
- List[String] 은 List[Any] 의 서브타입이다.

구조적 타입 structural type

- 세분화한 타입 refinement type 중에 세분화한 것이 기반 타입이 아니라 멤버들인 경우를 말함
- { def close(): Unit } 는 구조적 타입
- 이 타입의 기반 타입이 AnyRef 이며, AnyRef 예는 close 라는 멤버가 없기 때문

꼬리 재귀 tail recursion

- 함수가 자기 본문의 맨 마지막에서만 자기 자신을 재귀적으로 호출할 때 이를 꼬리 재귀라 함

내놓다 yield

- 표현식은 결고를 내 놓을 수 있음
- yield 키워드는 for 표현식의 결과를 지정함

다중 정의 multiple definition

- `val v1, v2, v3 =` 표현식 같은 표현 방식을 사용해 동일한 표현식을 가지고 여러 변수를 정의하고 초기화 하는 것
- 표현식을 한 번만 실행해서 각 변수에 같은 값을 초기화하는 게 아님
- 한 표현식을 여러 번 실행해 각 실행 결과를 각각의 변수에 할당함

단일 접근 원칙 uniform access principle

- 단일 접근 원칙은 변수나 파라미터가 없는 함수를 같은 문법 표현을 사용해 사용할 수 있어야 한다고 주장함
- 스칼라는 파라미터가 없는 함수의 경우 괄호를 생략할 수 있게 함으로써 이 원칙을 지원
- 클라이언트 코드에는 영향을 끼치지 않으면서 파라미터가 없는 함수 정의를 val 로 바꾸거나, 거꾸로 val을 파라미터가 없는 함수로 바꿀 수 있음

대수적 데이터 타입 algebraic data type

- 여러 가지 대안들로 이뤄진 타입으로, 각 대안은 각각의 생성자를 갖음
- 보통 대수적 데이터 타입은 패턴 매치를 사용해 분해할 수 있음
- 이런 개념은 명세 언어 specification language 나 함수형 언어 functional language에서 많이 볼 수 있음
- 스칼라에서는 케이스 클래스 case class 를 사용해 대수적 데이터 타입을 표현할 수 있음

대안 alternative

- 매치 표현식의 경우 중 하나를 의미
- 형식은 `case 패턴 => 표현식`
- 케이스 case 라 부르기도 함

도달할 수 없는 unreachable

- 스칼라 수준에서 어떤 객체에 도달할 수 없게 될 수 있음
- 이런 경우 해당 객체가 차지하고 있는 메모리는 런타임이 회수할 수 있음
- 도달할 수 없는지 여부를 꼭 참조 여부만으로 결정하는 것은 아님
- 모든 참조 타입 (AnyRef의 인스턴스)은 JVM 힙 heap에 있는 객체로 구현
- 어떤 차조 타입의 인스턴스가 도달할 수 없는 상태가 되면, 실제로도 그 객체에 대한 참조가 없는 상태여야 하며, 쓰레기 수집기가 수집 할 수 있음
- 값 타입 AnyVal 은 원시 타입 값이나 힙에 있는 자바 래퍼 Wrapper 타입이 인스턴스로 구현할 수 있음 (java.lang.Integer)
- 값 타입 인스턴스는 그 인스턴스를 의미하는 변수의 전체 생명주기 동안에 때때로 박스에 들어갈 수도 있고(원시 값에서 래퍼 객체로 변환), 박스에서 나올 수도 있음(래퍼 객체에서 원시 타입 값으로 변환)
- 어느 순간 래퍼 객체로 JVM 힙에 있는 값 타입 인스턴스가 도달할 수 없게 되려면, 실제로도 그 인스턴스를 참조하는 것이 없어야 함
- 이런 경우 쓰레긱 수집기가 처리할 수 있음
- 원시 값으로 구현한 값 타입 인스턴스에 도달할 수 없는 경우라면, 참조가 없는 상태가 되지 않음
- 해당 객체는 실행 지점에 JVM 힙에 존재하지 않기 때문
- 런타임이 이런 도달할 수 없는 값이 차지하는 메모리를 회수할 수 있는 경우도 있음
- 실행 시점에 자바 원시 타입인 int로 구현한 Int 타입의 인스턴스는 실행 중에 메소드의 스택 프레임상에 존재할 것임
- 그 메모리는 메소드가 끝나고 스택 프레임을 팝pop 할 때 '회수' 하는 셈
- String 같은 참조 타입의 메모리는 도달할 수 없는 상태가 된 후, JVM 의 쓰레기 수집기가 회수할 것임

도우미 메소드 helper method

- 클래스의 멤버인 도우미 함수를 의미
- 도우미 메소드는 보통 비공개 private 인 경우가 많음

도우미 함수 helper function

- 근처의 다른 함수에게 서비스를 제공하기 위해 만든 함수
- 도우미 함수는 지역 함수로 구현하는 경우가 일반적

독립 객체 standalone object

- 동반 클래스가 없는 싱글톤 객체
- 독립 싱글톤 객체라고도 함

동반 객체 companion object

- 어떤 클래스와 같은 소스 파일 안에 있고, 그 클래스와 이름이 같은 싱글톤 객체
- 동반 객체와 클래스는 서로의 비공개 멤버에 접근할 수 있음
- 동반 객체에 있는 모든 암시적 변환 implicit conversion 은 해당 클래스를 사용하는 모든 곳에서 암시적 변환 검색 스코프 안에 들어감

동반 클래스 companion class

- 한 소스 파일 안에 있는 싱글톤 객체 singleton object 와 클래스의 이름이 같다면, 그 클래스를 싱글톤 객체의 동반 클래스라 함

동일성 equality

- 동일성은 == 가 표현하는 관계
- 참조 동일성을 참고

참조 동일성 reference equality

- 두 참조가 같은 자바 객체를 가리킨다는 뜻
- 참조 동일성은 참조 타입에만 해당
- AnyRef의 eq 메소드로 검사할 수 있음
- (자바 프로그램에서는 자바 참조 타입에 ==를 사용해 참조 동일성을 결정할 수 있음)

런타임 runtime

- 스칼라 프로그램을 실행해주는 자바 가상 머신 JVM을 말함
- 런타임은 자바 가상 머신 명세가 정의한 가상머신
- 스칼라 표준 API와 자바 API를 위한 자바 런타임 라이브러리로 이뤄짐
- 실행 시점이라는 어구는 프로그램이 실행 중인 순간을 의미
- 컴파일 시점 compile time 과 대조적인 개념

리터럴 literal

- 1, “one”, (x: Int) ⇒ x+1 등
- 리터럴은 어떤 객체를 묘사하는 간단한 방법
- 리터럴은 만들어낼 대상 객체의 구조를 정확하게 텍스트로 표현함

메소드 method

- 어떤 클래스, 트레이트, 싱글톤 객체의 멤버인 함수

메시지 message

- 액터는 서로에게 메시지를 보내서 통신함
- 메시지를 받아도 수신자가 실행 중인 것을 방해하지 않음
- 수신자는 현재 진행 중인 활동을 마치고 자신의 불변조건이 계속 성립할 때까지 기다렸다가 다음 메시지를 처리할 수 있음

메타 프로그래밍 meta-programming

- 메타 프로그래밍 소프트웨어는 소프트웨어를 입력으로 받는 소프트웨어

- 컴파일러도 메타 프로그램이고, 스칼라독도 메타 프로그램
- 애노테이션을 가지고 어떤 작업을 하려면 메타 프로그래밍 소프트웨어가 필요함

멤버 member

- 클래스, 트레이트, 싱글톤 객체의 본문에 있는 이름이 붙은 구성요소를 말함
- 멤버는 그 멤버의 소유자의 이름 다음에 마침표(.)를 붙이고 메버의 간단한 이름을 붙여서 접근 할 수 있음
- 어떤 클래스의 최상위 수준에서 정의한 필드나 멤버는 그 클래스의 멤버
- 어떤 클래스 안에 정의한 트레이트는 그 클래스의 멤버
- 한 클래스의 내부에 type 키워드로 정의한 타입도 바로 그 클래스의 멤버
- 어떤 패키지 안에 정의한 클래스는 그 패키지의 맴버
- 지역 변수나 지역 함수는 그 변수나 함수를 둘러싼 블록의 멤버가 아님.

명령형 스타일 imperative style

- 명령형 프로그래밍 스타일은 연산의 순서를 주의 깊게 설계해서 각 연산의 효과가 올바른 순서로 일어나게 만드는 것
- 루프를 통한 반복, 데이터의 값을 그 자리에서 변경하기, 부수 효과 side effect 가 있는 메소드로 특징 지을 수 있음
- 명령형 스타일은 C, C++, C#, 자바 등의 주요 패러다임으로 함수형 스타일 functional style 과 대조적

무공변성 nonvariant(invariant라고도 함)

- 어떤 클래스나 트레이트의 타입 파라미터는 아무표시가 없으면 무공변임
- 그 타입 파라미터에 해당하는 타입 인자가 다른 경우 각각의 클래스나 트레이트는 서로 서브타입 관계가 없음
- Array 가 타입 파라미터에 대해 무공변이기 때문에 Array[String] 은 Array[Any] 의 서브타입도 아니고 슈퍼타입도 아님

문 statement (명령, 명령문)

- 어떤 표현식 expression, 정의 definition, 임포트import
- 스카랄 소스 코드에서 템플릿이나 블록에 들어갈 수 있는 것들을 의미

믹스인 mixin (혼합, 섞어 넣기)

- 믹스인 합성에서 사용하는 트레이트를 일컬음
- `trait Hat`이라고 말하면 Hat은 그냥 트레이트에 불과함
- `new Cat extends AnyRef with Hat`이라는 문장에서는 Hat을 믹스인이라고 부름
- 동사로 사용하는 경우 믹스인한다(혼합한다)라고 말함
- 트레이트를 클래스에 믹스인하거나, 다른 트레이트에 혼합할 수 있음

믹스인 합성 mixin composition

- 트레이트를 클래스나 다른 트레이트에 섞어 넣는 과정을 말함

- 믹스인 합성과 전통적인 다중 상속이 다른 부분은 트레이트를 정의하는 시점에는 어떤 타입이 슈퍼타입이 될지 알 수 없음
- 트레이트를 다른 클래스나 트레이트에 합성할 때마다 컴파일러가 슈퍼타입을 새로 결정함

바운드 변수 bound variable

- 어떤 표현식에서 바운드 변수란 그 식에서 정의하고 그 식 안에서 사용하는 변수를 말함
- (x: Int) ⇒ (x, y)라는 함수 리터럴 표현식의 본문에서는 x와 y 라는 변수를 사용하지만, x만 바운드 변수
- Int 타입의 유일한 인자로 정의했기 때문
- 묶인 변수나 엮인 변수라고 부르기도 함

반공변성 contravariant

- 클래스나 트레이트의 타입 파라미터 앞에 뺄셈 기호(-)를 추가하면 타입 파라미터에 반공변 표기를 붙일 수 있음
- 클래스나 트레이트를 표기가 붙은 파라미터 타입과 반공변적인 타입으로 다룸
- 반공변 관계란 서브타입 관계가 타입 파라미터의 서브타입 관계와 반대 방향으로 성립한다는 뜻
- Function1의 첫 번째 타입 파라미터에 대해 반공변이면, Function1[Any, Any] 는 Function1[String, Any] 의 서브타입이다.

반 구조화 데이터 semi-structured data

- XML 데이터는 반 구조화 되어 있음
- 일반 이진 파일이나 텍스트 파일보다는 더 구조화되어 있지만, 프로그래밍 언어의 데이터 구조처럼 완전한 구조를 갖춘 데이터는 아님

반환 return

- 스칼라 프로그램에서는 함수가 값을 돌려줌
- 이 값을 그 함수의 결과 return 라 함
- 그 함수가 그 값을 반환한다고 말하기도 함
- 스칼라에서는 모든 함수가 객체를 반환함

변경 불가능 immutable(불변)

- 어떤 객체를 만들고 나서 클라이언트가 그 값의 변화를 관찰 할 수 없는 경우
- 변경 불가능한 객체와 변경 가능한 객체가 있음

변성 variance

- 어떤 클래스나 트레이트의 타입 파라미터에 변성 표기를 붙일 수 있음
- 이런 표기에는 공변성covariant(+로 표시)이나 반공변성contravariant(-로 표시)가 있음
- 이런 변성 표기는 제네릭 클래스나 트레이트에 대해 서브타입 관계를 어떻게 적용할지 알려줌
- List 클래스는 타입 파라미터에 대해 공변적
- List[String]은 List[Any]의 서브타입

- + 나 - 애노테이션이 없는 경우에 타입 파라미터는 무공변성

변수 variable

- 어떤 객체를 의미하는 이름이 붙은 실체entity
- 변수는 val이거나 var일 수 있음
- val 과 var 모두 정의시 초기화해야만 함
- var만 나중에 다른 객체를 가리키도록 재할당 할 수 있음

부분 적용 함수 partially applied function

- 표현식에서 함수를 사용하는데 인자 중 하나 이상을 제공하지 않는 경우
- 어떤 함수 f가 Int ⇒ Int ⇒ Int 타입이라면 f 와 f(1) 은 부분 적용 함수임

불변조건

- 어떤 데이터 구조를 제대로 구성했을 때 well formed, 항상 만족시키는 어떤 특성을 의미할 수도 있음
- 정렬한 이진 트리의 불변조건은 각 노드가 자신의 오른쪽 하위 노드(오른쪽 하위 노드가 있는 경우)보다 순서가 앞서야 한다는 것, 이를 불변조건이라 함

블록 block

- 하나 이상의 표현식이나 정의를 중괄호{}로 둘러싼 것
- 블록을 계산할 때는 내부의 모든 표현식과 선언을 순서대로 처리하고, 가장 나중에 있는 표현식의 값을 블록 전체의 결과 값으로 반환
- 블록은 함수나 for 표현식, while 루프의 몸통으로 자주 사용
- 여러 명령을 한 그룹으로 다루고 싶을 때 어느 곳에서나 사용할 수 있음
- 블록은 외부에서 결과 값과 부수 효과만을 관찰할 수 있는 캡슐화한 프로그램 구성요소
- 클래스나 객체 정의 부분의 중괄호는 그 안에 있는 메소드나 필드를 외부에서 볼 수 있기 때문에 블록이 아님
- 이 경우에는 중괄호가 템플릿template을 형성함

서브클래스subclass

- 어떤 클래스는 자신의 슈퍼클래스와 슈퍼트레이트의 서브클래스임

서브타입 subtype

- 스칼라 컴파일러는 어떤 타입이 필요한 곳이라면 어디라도 그 타입의 서브타입을 허용함
- 타입 파라미터를 취하지 않는 클래스나 트레이트의 경우 서브타입 관계는 서버클래스 관계를 그대로 반영함
- Cat 클래스가 추상 클래스 Animal의 서브클래스이고, 두 클래스 모두 타입 파라미터를 받지 않는다면, Cat 타입은 Animal 타입의 서브타입
- Apple 트레이트가 Fruit 트레이트의 서브트레이트이고, 둘 다 타입 파라미터를 취하지 않으면, Apple 타입은 Fruit 타입의 서브타입
- 타입 파라미터를 취하는 클래스나 트레이트라면, qustjd vvariance이 작용하기 시작

- List 추상 클래스는 자신의 유일한 타입 파라미터에 대해 공변convariant적이라고 선언에 적혀 있임(List 는 List[+A])
- List[Cat] 은 List[Animal]의 서브타입이며, List[Apple]은 List[Fruit]의 서브타입
- Set은 타입 파라미터에 대해 공변적이지 않음
- 따라서, Set[Cat]은 Set[Animal]의 서브타입이 아님
- 서브타입은 슈퍼타입과의 계약Contract을 올바르게 구현해야 함
- 리스코프 치환 원칙 적용
- 컴파일러는 단지 타입 검사 수준에서 이런 특성을 검증 할 수 있을 뿐

서브트레이트 subtrait

- 어떤 트레이트는 자신의 모든 슈퍼트레이트supertrait의 서브트레이트

선언 declaration

- 추상 필드field, 메소드, 타입을 선언할 수 있음
- 선언은 어떤 엔티티entity에 이름을 부여하지만 구현은 하지 않음
- 선언과 정의definition의 핵심 차이는 정의는 엔티티에 이름을 붙이면서 구현도 제공하는 반면 선언은 그렇지 않다는 점임

세분화한 타입 refinement type

- 기반 타입 base type 뒤에 중괄호를 넣고 그 안에 멤버를 나열해서 정의한 타입을 말함
- 중괄호 안에 있는 멤버가 꼭 들어 있는 타입만으로 기반 타입을 더 세분화함
- '풀grass을 먹는 동물animal'의 타입은 Animal { type SuitableFood = Grass }

셀렉터 selector

- match 식에서 매치시킬 대상 값을 셀렉터라 함
- 's match { case _ ⇒ }' 에서 셀렉터는 s

셀프 타입 self type

- 어떤 트레이트의 셀프 타입은 트레이트 안에서 호추루 대상 객체인 this에 대해 가정하는 타입
- 트레이트를 혼합하는 모든 구체적 클래스는 트레이트의 셀프 타입과 부합해야 함
- 셀프 타입을 가장 일반적으로 사용하는 경우는 큰 클래스를 여러 트레이트로 나누는 경우

수식자 modifier

- 어떤 클래스나 트레이트, 필드, 메소드 정의를 어떤 식으로 제한하는 키워드
- private 수식자는 정의하는 중인 클래스, 트레이트, 필드, 메소드 등이 비공개라라는 사실을 지정함

슈퍼클래스 superclass

- 어떤 클래스의 슈퍼클래스는 자신의 직접 슈퍼클래스direct superclass
- 직접 슈퍼클래스의 직접 슈퍼클래스 등으로 거슬러 올라가면서 Any에 이르는 모든 클래스

슈퍼타입 supertype

- 어떤 타입은 자신의 모든 서브타입의 슈퍼타입

슈퍼트레이트 supertrait

- 어떤 클래스나 트레이트의 슈퍼트레이트는 어떤 클래스나 트레이트에 믹스해 넣은 모든 트레이트와
- 그것의 슈퍼클래스의 슈퍼트레이트,
- 그리고 이러한 트레이트의 슈퍼트레이트들

술어 predicate(조건 함수)

- 술어는 결과 타입이 Boolean 인 함수를 의미

스크립트 script

- 최상위 선언과 명령문을 포함하는 파일
- 컴파일하지 않고 스칼라가 직접 실행
- 스크립트는 반드시 정의가 아닌 표현식으로 끝나야 함

시그니처 signature

- 시그니처는 타입 시그니처 type signiture 를 짧게 줄인 말

실행 시점 타입 runtime type

- 어떤 객체가 실행 시점에 갖는 타입
- 정적 타입 static type 은 어떤 표현식이 컴파일 시점에 갖는 타입
- 대부분의 실행 시점 타입은 타입 파라미터가 없는 간단한 클래스
- “Hi”의 실행 시점 타입은 String
- (x: Int) ⇒ x+1 의 실행 시점 타입은 Function1
- isInstanceOf 를 사용하면 실행 시점 타입을 검사할 수 있음

싱글톤 객체 singleton object

- object 키워드로 정의한 객체
- 각 싱글톤 객체의 인스턴스는 유일
- 어떤 싱글톤 객체와 같은 이름의 클래스가 한 소스 파일 안에 있으면, 싱글톤 객체를 그 클래스의 동반 객체라 함
- 거꾸로 그 클래스는 객체의 동반 클래스
- 동반 클래스가 없는 싱글톤 객체를 독립 객체 standalone object 라 함

애노테이션 annotation

- 소스 코드에 들어가 있으며 프로그램 문법 구조의 일부분에 덧붙인 텍스트
- 애노테이션은 컴퓨터가 처리 가능하기 때문에, 스칼라를 확장할 때 사용

연산 operation

- 스칼라에서 모든 연산은 메소트 호출
- 메소드를 b+2 와 같이 연산자 표기법 operator notation 을 사용해 호출 가능
- 이런 표기법으로 작성한 표현식에서 + 는 연산자operator

외부 생성자 auxiliary contructor

- 클래스 정의 본문의 중괄호 안에서 정의한 추가 생성자
- 마치 결과 타입이 없는 this라는 이름의 메소드 정의와 같음

익명 클래스 anonymous class

- 익명 클래스는 클래스나 트레이트 이름 뒤에 중괄호({})가 오는 new 표현식으로부터 스칼라가 만들어내는 합성 서브클래스 synthetic subclass
- 중괄호에는 익명 서브클래스의 본문을 넣는데, 비워둘 수도 있음
- new 뒤에 있는 이름이 추상 멤버가 있는 클래스나 트레이트라면 반드시 그 추상 멤버를 익명 클래스의 중과롷 안에 있는 본문에서 구체화해야만 함

익명 함수 anonymous function

- 함수 리터널 function literal 을 이렇게 부름

인스턴스 instance

- 인스턴스 또는 클래스 인스턴스는 객체 object 이며, 실행 시점 run time에만 존재하는 개념

인스턴스화 instantiate

- 어떤 클래스를 인스턴스화하는 것은 그 클래스로부터 객체를 새로 만드는 것
- 인스턴스화는 실행 시점에만 일어남

인자 argument

- 함수를 호출할 때 함수의 모든 파라미터 parameter 마다 전달하는 실제 값이 인자
- 파라미터는 인자를 가리키는 변수
- 인자는 호출 시 전달하는 객체
- 애플리케이션은 애플리케이션 싱글톤 객체의 main 메소드에 Array[String] 타입의(명령행) 인자로 받을 수 있음

자유 변수 free variable

- 어떤 표현식에 있는 자유 변수란 표현식 안에서 쓰이는데 표현식안에 정의가 없는 변수
- 함수 리터럴 표현식 (x: Int) ⇒ (x,y) 를 보면 x와 y라는 변수가 있지만 y는 이 표현식 안에 정의가 없기 때문에 자유 변수 임

재귀적 recursive

- 어떤 함수가 자기 자신을 호출하면 재귀적이라 함
- 어떤 함수가 가장 나중에 수행하는 표현식이 자기 자신을 재귀적으로 호출하는 것이면, 이를 꼬리 재귀적 tail recursive 라고 함

재할당 가능 reassignable

- 어떤 변수는 재할당 가능하거나 그렇지 않음
- var 변수는 재할당 가능
- val 변수는 재할당 불가능

정의 definition

- 스칼라 프로그램에서 무언가를 정의하려면 이름과 구현을 제고해야 함
- 클래스, 트레이트, 싱글톤 객체, 필드, 메소드, 지역 함수, 지역 변수 등을 정의할 수 있음
- 정의를 하려면 항상 구현이 따라와야 하기 때문에, 추상 멤버는 선언할 수는 있지만 구현할 수는 없음

제너레이터 generator

- 제너레이터는 for 식에서 어떤 val을 정의하면서 일련의 값을 할당하는 부분
- for(i ← 1 to 10) 에서 제너레이터는 i← 1 to 10
- ← 의 오른쪽에 있는 표현식을 제너레이터 표현식이라 함

제너레이터 표현식 generator expression

- 제너레이터 표현식은 for 표현식에서 일련의 값을 만들어냄

제네릭 클래스 generic class

- 타입 파라미터를 받는 클래스
- 타입 파라미터를 받기 때문에 scala.List는 제네릭(타입 일반적) 클래스

제네릭 트레이트 generic trait

- 타입 파라미터를 받는 트레이트
- scala.collection.Set 트레이트는 타입 파라미터를 받기 때문에 제네릭 트레이트

존재 타입 existential type

- 존재 타입은 알지 못하는 타입 변수에 대한 참조가 들어 있는 타입
- Array[T] forSome { type T } 는 존재타입

주 생성자 primary constructor

- 어떤 클래스의 중심 생성자로 주 생성자만 슈퍼클래스의 생성자를 호출할 수 있음
- 전달받은 인자 값을 사용해 필드를 초기화하고 클랫의 중괄호 사이에 있는 최상위 코드를 모두 실행함
- 슈퍼클래스 생성자에 넘기는 값 파라미터를 제외한 나머지 필드만을 초기화하며, 클래스의 본문에서 사용하지 않는 필드는 제외함

지역 변수 local variable

- 지역 변수는 블록 안에 들어 있는 val 이나 var 정의를 말함
- 지역 변수와 비슷하지만, 함수의 파라미터는 지역 변수라고 하지 않으며, 그냥 파라미터라고 말하거나 '지역'을 붙이지 않고 '변수'라고만 부름

지역 함수 local function

- 지역 함수는 블록 안에 들어 있는 def 정의를 말함
- 클래스나 트레이트, 싱글톤 객체의 멤버인 def 는 메소드라 함

직렬화 serialization

- 어떤 객체를 파일에 저장하거나 네트워크로 전송할 수 있도록 바이트 스트림으로 직렬화할 수 있음
- 나중에 바이트 스트림을 역질렬화해서 원래의 객체와 같은 객체를 얻을 수 있음

참조 reference

- 참조는 자바에서 포인터pointer 를 추상화한 것
- 참조는 JVM 힙 heap에 있는 객체를 유일하게 식별
- 참조 타입 변수는 객체에 대한 참조를 저장
- 스칼라에서는 참조 타입 (AnyRef 의 인스턴스)을 JVM 힙에 있는 객체로 구현하기 때문
- 값 타입 value type 변수도 때때로 참조를 저장할 수 있고, 그렇지 않을 때도 있음
- 일반적으로 스칼라 변수는 객체를 나타냄
- 만약 어떤 변수의 타입이 scala.Int 이고, 그 객체를 자바의 원시 int 값으로 표현하고 있다면, 해당 변수는 여전히 Int 객체를 나타내지만, 실제 이 변수는 어떤 참조와도 관련이 없음

참조 동일성 reference equality

- 참조 동일성은 두 참조가 같은 자바 객체를 가리킨다는 뜻
- 참조 동일성은 참조 탕비에만 해당하며 AnyRef 의 eq 메소드로 검사할 수 있음
- (자바 프로그램에서는 자바 참조 타입에 == 를 사용해 참조 동일성을 결정할 수 있음)

참조 투명성 referential transparency

- 어떤 함수가 시간에 따른 환경 변화 temporal context 와 무관하고, 부수 효과가 없을 때 이 함수를 참조 투명하다고 말함
- 참조 투명한 함수에 어떤 입력을 전달해 호출하는 것은 프로그램의 의미에 영향을 주지 않고, 해당 입력에 대한 그 함수의 결과 값으로 대치할 수 있음

refer 참조한다, 나타낸다

- 실행 중인 스칼라 프로그램의 변수는 항상 어떤 객체를 나타냄
- 해당 변수에 null을 할당했다고 해도 개념적으로 그 변수는 Null 객체를 나타냄
- 런타임에서는 객체를 자바 객체나 자바 원시 타입의 값으로 구현
- 스칼라에서는 프로그래머가 코드의 실행을 상상할 때 더 높은 추상화 수준에서 생각하도록 함

초기화 initialization

- 스칼라 소스 코드에서는 어떤 변수를 정의하려면 그 변수를 객체로 초기화해야 함

커링 currying

- 여러 파라미터 목록을 받는 함수를 정의하는 방법
- def f(x: Int)(y: Int) 는 파라미터 목록을 2개 받는 커링 함수
- 커리한 함수는 f(3)(4) 와 같이 여러 인자 목록을 넘겨서 호출
- f(3)처럼 처리한 함수를 부분 적용 partial application 할 수도 있음

클래스 class

- class 키워드를 사용해 정의
- 추상 abstract 클래스이거나 구체concrete 클래스일 수도 있음
- 인스턴스를 생성할 때 타입 파라미터와 값 파라미터를 받게 만들 수 있음
- new ArrayString 에서 인스턴스화하고 있는 클래스는 Array이며, 결과 값의 타입은 Array[String]
- 파라미터를 받는 클래스를 타입 생성자type constructor라 함
- Array[String]타입의 클래스는 Array임

클로저 closure

- 자유 변수 free variable 를 포획한 capture 함수 객체를 말함
- 함수 객체를 생성하는 시점에 스코프scope 에서 볼 수 있었던 변수에 대해 '닫혀 있다closed'라고 설명하기도 함

타깃 타이핑 target typing

- 예상 타입을 고려해 넣는 형태의 타입 추론type inference
- nums.filter((x)⇒ x >0) 이라는 표현식을 보고, 스칼라 컴파일러는 x의 타입이 nums의 원소 타입과 같다는 사실을 추론해 냄
- filter 메소드는 인자로 받은 함수를 nums의 모든 원소에 호출할 것이기 때문

타입 type

- 스칼라는 컴파일 시점에 프로그램의 모든 변수와 표현식에 타입을 부여함
- 타입은 어떤 변수가 의미할 수 있는 값의 범위를 제한
- 실행 시점에 표현식이 만들어낼 수 있는 값의 범위를 제한
- 변수나 표현식의 타입을 해당 객체의 런타임에서의 타입과 구분하고 싶은 경우 정적 타입static type 이라 표현함
- '타입'이라고만 하면 보통은 정적 타입을 의미
- 타입과 클래스는 다름
- 클래스는 타입 파라미터를 받아서 여러 타입을 만들어낼 수 있음
- List는 클래스이지만 타입은 아님
- List[T] 는 자유 타입 파라미터가 있는 타입
- List[Int]와 List[String]은 타입
- 어떤 타입에 대한 '클래스'나 '트레이트'가 있을 수 있음

타입 생성자 type constructor

- 타입 파라미터를 취하는 클래스나 트레이트를 말함

타입 시그니처 type signature

- 메소드의 타입 시그니처는 파라미터의 이름과 타입, 개수, 순서와 함께 메소드의 결과 타입으로 이뤄짐
- 클래스의 타입 시그니처는 그 클래스가 선언한 상속과 믹스인 관계와 각 멤버와 생성자의 타입 시그니처를 합한 것

타입 제약 type constraint

- 몇몇 애노테이션은 타입 제약임
- 의미는 타입에 추가로 제한이나 제약을 부여해서 해당 타입에 속한 값의 범위를 줄인다는 뜻
- @positive라는 타입 제약은 Int에 부여해서, 32비트 정수 중에 양수인 경우만을 의미
- 표준 스칼라 컴파일러는 타입 제약을 검사하지 않음
- 별도의 도구를 사용하거나 컴파일러 플러그인을 사용해 처리해야 함

타입 파라미터 type parameter

- 제네릭 클래스나 메소드의 파라미터 중에서 반드시 타입으로 채워야만 하는 것을 의미
- List의 정의는 'case List[T] { ...' 라고 시작
- Predef 객체의 멤ㅁ버인 identity 메소드 정의는 'def identityT = x'
- 이 두 경우 모두 T가 타입 파라미터

템플릿 template

- 템플릿은 클래스, 트레이트, 또는 싱글톤 객체 정의의 본문
- 템플릿은 타입 시그니처를 정의
- 클래스나 트레이트 또는 객체의 동작과 초기 상태를 정의

트레이트 trait

- trait 키워드로 정의
- 어떤 값 파라미터도 받지 않는 추상 클래스와 비슷
- 트레이트는 믹스인 합성 mixin composition 이라는 과정을 통해 다른 트레이트나 클래스에 '섞어 넣을' 수 있음
- 트레이트를 하나 이상의 타입으로 파라미터화 할 수 있음

패턴 pattern

- match 식의 대안(케이스)에서
- 패턴은 case 키워드 뒤에 오며,
- 패턴 가드나 ⇒ 기호 앞에 옴

패턴 가드 pattern guard

- match 식의 대안에서, 패턴 가드는 패턴 뒤에 옴
- 'case x if x % 2 == 0 ⇒ x + 1' 에서 패턴 가드는 'if x % ==0'
- 해당 패턴과 매치할 수 있고, 패턴 가드가 참인 경우에만 패턴 가드가 있는 케이스를 선택

표현식 expression

- 결과를 내놓는 스칼라 코드 조각

함수형 스타일 functional style

- 함수형 프로그래밍 스타일은 함수와 결과 계산evaluation 을 강조
- 연산이 일어나는 순서를 강조하지 않음
- 함수 값을 루프 메소드(재귀 메소드)에 전달하는 것
- 변경 불가능한 데이터(immutable data)
- 부수 효과가 없는 메소드로 특징 지을 수 있음
- 하스켈Haskell이나 얼랭Erlang 등의 언어에서 주요 패러다임이며, 명령형 스타일imperative style 과 대조적

관련 문서

- Scala

From:
https://jace.link/ - **Various Ways**

Permanent link:
https://jace.link/open/scala-%EC%9A%A9%EC%96%B4

Last update: **2020/06/02 09:25**