객체 지향 언어 자바와 하이브리드형 언어 코틀린 비교

김동훈, 엄태현, 이혁준
광운대학교

donghun0510@naver.com, crackscendo@gmail.com, hlee@kw.ac.kr

Comparison Between Java and Kotlin

Kim Dong Hun, Eom Tae Hyun, Lee Hyun Jun
Kwangwoon Univ.

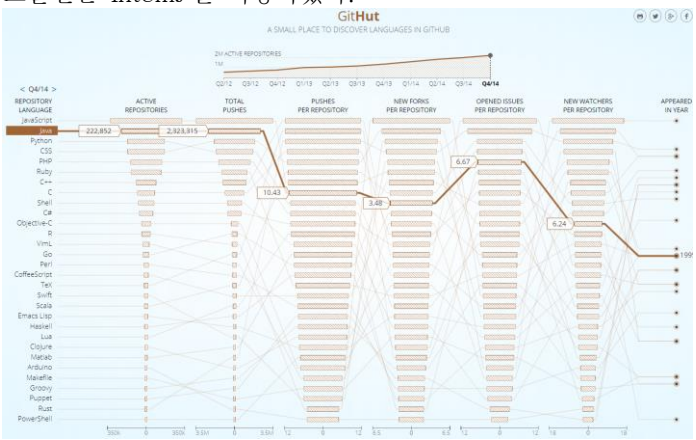
요 약

현재 프로그래밍 언어 점유율 1 위를 기록하는 자바에 대응하여 구글 공식 프로그래밍 언어로 채택된 코틀린은 점진적 추이를 보이고 있다. 자바와 코틀린에 대한 다양한 시각이 존재하지만, 그에 따른 명확한 차이점 및 실질적 비교 우위를 알기 어렵다. 본 논문은 객체 지향적 언어인 자바와 하이브리드형 언어인 코틀린, 해당 각 언어적 특징, 차이 및 언어별 프로그래밍을 통한 언어적 차이와 성능 비교를 해보고자 한다.

Ⅰ. 서 론

[그림 1]을 참고하면 프로그래밍 언어 점유율 1 위를 앞다투는 자바와 최근 구글의 공식 프로그래밍 언어로 채택된 코틀린에 대한 다양한 시각이 존재한다. 하지만 그러한 시각을 보여주는 것이 프로그래머로서, 개발자로서 말하려는 것이 아닌 물질적 이득을 위한 정보 제공인 경우가 많다. 이러한 정보에 노출된 비전공자 및 주니어 개발자는 명확한 차이점 및 각 언어의 장단점을 알지 못하여 비효율적인 프로그래밍 설계 및 구현이 이루어질 수 있다.

본 논문에서는 자바와 코틀린의 전반적인 내용을 다루고, 해당 언어를 사용한 안드로이드 어플리케이션 개발을 통해 각 언어별 장단점 및 비교우위를 공유한다. 안드로이드 어플리케이션 개발을 위해서는 안드로이드 스튜디오, 성능 비교를 위해서는 자바는 이클립스, 코틀린은 IntelliJ 를 사용하였다.



[그림 1] GitHub 레퍼지토리 점유 언어 순위[1]

Ⅱ. 자바 및 코틀린 전반적인 기술 및 비교

1) 자바

특징	설명
1.캡슐화/은닉화 (Encapsulation/ Information Hiding)	단순화하는 것
2. 상속 (Inheritance)	계층 구조의 표현으로 이미 정의된 상위 클래스의 속성과 연산 등을 하위 클래스가 물려받는 것
3. 다형성 (Polymorphism)	하나의 계층에 속한 객체들이 같은 명령에 대해 다른 행위를 수행하는 능력.
4. 추상화 (Abstraction)	객체에서 공통된 속성과 행위를 추출하는 것
5. 사전 정의	사전 정의된 타입은 모두 객체이다.
6. 사용자 정의	사용자가 정의하는 것 모두 객체이다.
7. 메소드 (Method)	모든 연산은 객체 안의 메소드를 통해야 한다.

[표 1]Pure/Fully OOP(Object Oriented Programming)의 조건

자바는 처음부터 객체 지향적 언어로 설계된 언어로 클래스 계층 구조, 상속, 다형성 및 캡슐화 등을 지원하지만, [표 1]과 비교한다면, 자바는 완전한 객체 지향 언어라고 볼 수 없다.[2] 자바는 기본 데이터 유형을 객체로 표현하지 않고, 클래스를 static 으로 선언한다면, 객체로써 쓰이지 않을 수 있으며, 자바에서는 Wrapper class 내부에 존재하는 메소드만을 이용해 사용할 수 없고, 기본형에 의존하여 연산해야 하므로 완전한 객체 지향 언어는 아니지만, 객체 지향적 언어의 특징을 가지고 있다.[3]

자바는 객체 지향 언어의 캡슐화(Encapsulation)의 원칙에 철저하여 변수, 메서드는 반드시 클래스 내에 구현해야 하고, 클래스 내부에 새로운 클래스, 내부 클래스를 만들 수 있으며, 서로 관련된 클래스를 패키지로 묶어 관리할 수 있다. 하나의 자바 소스 파일에는 여러 클래스를 작성할 수 있지만, 하나의 클래스 파일에는 반드시 하나의 컴파일 된 자바

클래스만 포함되고, 여러 클래스가 존재할 경우 클래스마다 별도 클래스파일이 생성된다.
Java8 부터 도입된 람다식, 함수적 코딩 방식을 통해 코드를 간결하게 작성할 수 있고, 대용량 데이터에 대한 처리가 가능하다.[4]

2) 코틀린
코틀린은 함수형 프로그래밍으로 기존 함수형 프로그래밍 언어보다 더 많은 프로그래밍 기법을 제공하며 객체 지향 프로그래밍도 지원한다.
개발 코드가 자바 클래스로 빌드되어 JVM 에서 동작하기 때문에 자바와 완전하게 상호 호환이 되며, JVM 기반 모든 언어로 개발할 수 있다.[6] 람다 표현식 역시 지원하며, 이미 널리 쓰이는 코드 패턴을 간결화 할 수 있도록 설계되었다. 타입 추론을 지원하여 일반적인 경우 타입 생략이 가능하다.
코틀린의 비동기 처리 모델인 코루틴은 작업들이 자발적으로 양보하여 실행을 바꾸어 주는 협력형 멀티테스킹을 통해 동시성 프로그래밍을 지원한다. [8] 해당 루틴을 일시 중단(Suspended)하는 방식으로 문맥교환을 없애고, 최적화된 비동기 함수를 통해 비선점형으로 작동하여 기존의 복잡한 Non-blocking 코드를 간결하고, 더 나은 성능을 지원한다.
코틀린은 최신 언어로 잦은 업데이트로 장기적인 코딩을 진행한다면 주기적인 해당 공식 문서 확인이 필요하고, 플러그인이나 임포트 과정이 잦다는 점에서 비교적 불안정하다.

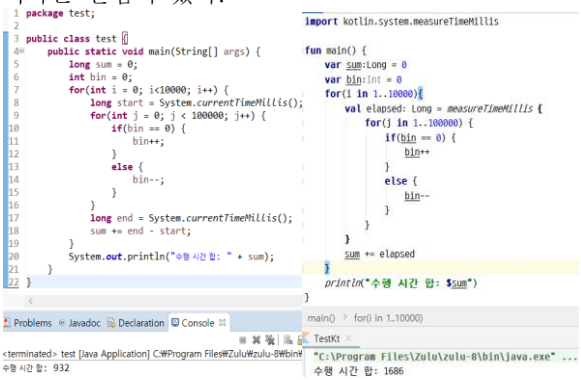
3) 비교
Null 에 대한 안전성에 있어, 자바의 경우 NPE 관련 코드 작성이 필요하지만, 코틀린은 NPE 처리 기법 제공으로 좀 더 수월한 개발이 가능하다. 또한 자바는 예외 지원을 확인하여 프로그래머가 예외를 선언 및 포착으로 최종적으로 오류 처리 기능을 가진 코드가 되지만, 코틀린은 예외 확인을 하지 않는다. 예외 처리를 할 필요가 없다는 점에서 편리할 수 있지만 자바만큼 안전하다고는 할 수 없다.
자바의 배열은 암시적으로 공변성 배열이며, 명시적 반변성 배열이다. 코틀린과 같이 불변성 배열을 사용하기 위해서, 매개 변수화 된 제너릭 및 와일드 카드 바인딩을 통해 가능하게 하려 했지만, 결과적으로 와일드 카드 유형은 추상적이고 참조용으로만 사용할 수 있다.[9]
코틀린은 자바 보다 직관적이고 간결한 표현과 Anko 라이브러리 사용 및 형식 유추, 스마트 캐스트를 통해 데이터 형식 등으로 비교적 코드를 간략하게 할 수 있다.[10]
비동기 데이터 흐름 데이터 프로그래밍을 위한 프로그래밍 패러다임으로 동작이 시간의 흐름에 따라 바뀌도록 하는 FRP(Functional Reactive Programming)는 자바와 코틀린 모두 가능한데, 코틀린은 코루틴을 통해 비동기 처리 방식을 통해서, 자바는 라이브러리 추가 또는 RxJava 를 통해 FRP 를 가능하게 한다.[11]
자바는 백그라운드에서 멀티 스레드(Multi-Thread)를 작성하고 실행하는 기능을 제공하지만 관리가 복잡하다. 하지만 코틀린의 코루틴 지원을 통해 장기 실행 집중 작업을 실행하는 동안 스레드를 차단하지 않고 특정 지점에서 실행을 일시 중단한다.
확장 기능으로 자바는 확장 기능을 사용할 수 없으므로, 기존 클래스의 기능을 확장하기 위해서는 새로운 클래스를 작성하고 상위 클래스를 상속해야 한다. 코틀린은 확장 기능을 가져 클래스 이름 앞에 새 함수 이름을 붙여 확장 함수를 만들 수 있다.

Features	Java	Kotlin
Fully OOP (Object-Oriented Programming)	Not pure OOP	Fully OOP
Null 안정성	No	Yes
Checked Exception	Yes	No
불변성배열(Invariant Array)	No	Yes
스마트 캐스트	No	Yes
Singletons Object	Yes	Easily create Singleton Object
Functional Reactive Programming	Yes	Yes

[표 2] 자바와 코틀린의 언어적 차이 비교

Ⅲ. 자바 및 코틀린 코드 차이 및 성능 비교

프로그램 구현 자체를 위해 작성해야 하는 코드가 코틀린을 사용할 때 비교적 짧은 코드로 자바의 코드와 동일한 수행을 할 수 있다. 코틀린은 형식 유추 기능을 통해 데이터 형식 생략이 가능하고, NPE 의 경우 코틀린에서 자바에서는 NPE 과정을 구현해야 한다는 것과 다르게 간결하게 표현할 수 있다. 하지만 컴파일 시간적으로 코틀린은 자바의 컴파일 속도에 비해 느리다는 단점이 있다.



[그림 2] 자바와 코틀린을 사용한 동일 수행 코드 및 결과(좌측: 자바/eclipse 및 우측: 코틀린/Intellij)

[그림 2]는 자바와 코틀린의 성능 비교를 위해, 같은 동작, 각 변수 별 동일한 자료형 및 동일한 방식으로 시간을 측정하는 함수를 사용하는 프로그램이다. 해당 측정을 1 만번 시행했을 시, 측정된 시간과 합계를 구한다. 각 수행 시간 합계는 자바에서 932, 코틀린에서 1686 이 나왔다. 코틀린이 약 자바에 비해 1.8 배 느리다는 것을 확인할 수 있었다.

Ⅳ. 결론

본 논문에서는 자바와 코틀린의 전반적인 언어적 차이점 및 안드로이드 어플리케이션 개발을 통한 비교에 초점을 두고 있다. 이를 위해 안드로이드 스튜디오라는 동일한 환경에서 각 언어를 사용하여 모바일 어플리케이션을 개발하고 비교한다. 이러한 과정은 주니어 개발자 및 컴퓨터 관련 비전공자가 모바일 어플리케이션 개발 언어 선택에 있어, 좀 더 명확하고 구체적인 정보를 얻을 수 있도록 한다.

ACKNOWLEDGMENT

본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW 중심대학지원사업(과제번호 : 2017-0-00096) 및 광운대학교 2016 년도 교내학술연구비 지원으로 수행하였습니다.

참 고 문 헌

[1] https://github.info/

[2] https://www.java.com/ko/about/

[3] https://docs.oracle.com/javase/7/docs/api/java/sql/Wrapper.html

[4] https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

[5]https://stackoverflow.com/

[6] https://kotlinlang.org/

[7] https://kotlinlang.org/docs/reference/null-safety.html

[8]https://kotlinlang.org/docs/reference/coroutines-overview.html

[9] https://docs.oracle.com/javase/tutorial/extra/generics/wildcards.html

[10] https://developer.android.com/kotlin/interop

[11]https://academy.realm.io/posts/droidcon-gomez-functional-reactive-programming/