# [코틀린 입문] 2주차 주간 질문

1 개의 메일

**Google 설문지** <forms-receipts-noreply@google.com>
받는사람: yenarue@gmail.com

2019년 4월 14일 오후 1:48

**[코틀린 입문] 2주차 주간 질문을(를) 작성해 주셔서 감사합니다.**

사용자로부터 다음과 같은 메시지를 받았습니다.

응답 수정

## [코틀린 입문] 2주차 주간 질문

**이메일 주소** *

yenarue@gmail.com

**제출자명** *

김예나

**참여 월/요일/시간대 (ex. 10월/화요/10시)** *

온라인 스터디가 없는 스터디의 경우 월만 적어주세요! (ex. 11월)

4월/일요/21시

**1. (Day1 강의 관련) : Safe access(?), Elvis Operator(?:), Not-null assertion(two exclamation marks operator)(!!) 가 무엇인지 각각 이해한 내용을 적어주세요.** *

- Safe access (`?`) : 해당 변수가 null이 아닐 때는 `.` 연산을 진행하고 null 일 때는 `null`을 리턴함으로서 access 하는 도중에 NPE가 발생하지 않도록 안전한 접근을 보장한다.
- Elvis Operator (`?:`) : Safe access 로 null을 체크할 때, null일 경우의 리턴 값을 `null`이 아닌 다른 값으로 설정하는 연산자. 디폴트 값을 설정하는 것.
- Not-null assertion (`!!`) : `null` 이 아니라는 것을 보장하는 연산자. 이 경우, 지정한 값이 `null` 이면 NPE가 발생한다. 그렇기 때문에 이 연산자는 논리적으로 확실하게 null일 가능성이 없다고 판단될 때에만 사용해야 한다. 남발하면 Billion dollar mistake 가 재현되는 꼴이 되기 때문이다. 결국 NPE가 발생가능하다는 점에서 어찌보면 Java와 비슷한 꼴이 아니냐는 생각이 들 수 있지만 java에 비해 *NPE 발생가능 지점을 명시적으로 보여준다는 점*이 이 연산자의 장점이라고 볼 수 있다.

**2. (Day 1 ~ 2 강의 관련) : Nullability 단원에서 배운 내용 중 가장 흥미로웠던 부분 1가지를 골라 조금 더 공부해보고 이해하신 내용을 적어주세요.** *

(ex: Nullability 문제는 왜 Billion Dollar Mistake라 불리는가?, 왜 Annotation 방식은 Optional 방식에 비해 성능 오버헤드가 없는가? or 그외 자유 주제)

- Null이 Billion Dollar Mistake 로 불리는 이유 : NPE 관련 문제들은 고치기도 어렵고 발견해내기도 어렵기 때문이다. 참조값이 존재하지 않는 경우를 위해 Null 은 꼭 존재하여야 하지만 런타임 에러로만 발견할 수 있기 때문에 처리하기도, 발견하기도 어려운 것이다. 이를 해결하기위한 현대적인 접근으로는 NPE 를 런타임이 아닌 컴파일 타

임에 알아낼 수 있도록 하자는 것이다. Kotlin은 이를 Nullable type을 도입함으로써 해결하려했다.
- Annotation 방식이 Optional 방식에 비해 성능 오버헤드가 없는 이유 : Kotlin에는 아예 Optional 클래스 자체가 존재하지 않는데, Null 처리를 위해 Wrapper 객체를 사용하던 Optional 클래스를 사용하지 않으므로 이에 따라 런타임 시에 추가적인 성능 오버헤드는 발생하지 않으면서도 Null 문제를 해결할 수 있게 된다.
- SubTyping : 서브클래스가 슈퍼클래스를 대체할 수 있는 경우 이를 서브타이핑이라고 한다. 서브클래스가 슈퍼클래스를 대체할 수 없는 경우에는 서브클래싱이라고 한다. 서브타이핑은 설계의 유연성이 목표인 반면 서브클래싱은 코드의 중복 제거와 재사용이 목적이다. Kotlin에서 Non-Nullable 타입과 Nullable 타입 서브타이핑으로서 서로 동일한 타입으로 동작하여 변수 할당에 유연성을 가져갈 수 있게 되었다.
- `lateinit` 키워드 : Kotlin에서는 Non-Nullability 타입인 경우 선언시 초기화 해주도록 강제하고 있다. 하지만 실무에서는 의존성 주입이나 설계상의 이유(값을 서버로 부터 받아와서 초기화해야한다든지…)로 나중에 초기화를 진행해야 하는 경우들이 생긴다. 이럴 경우를 대비하여 Kotlin에는 `lateinit` 키워드가 존재한다. `lateinit` 키워드는 변경가능한 변수(Mutable Variable)에 대해 나중에 초기화를 해주겠다고 약속하는 키워드이다.

**3. (Day 2 강의 관련) : Lambda란 무엇인가요? Lambda는 왜 유용한가요?** *

Lambda 는 표현식 (expression), 인스턴스 (instance), 인수 (argument, actual parameter)로서 사용가능한 *익명함수* 이다. *간결성이 극대화되고 자유성이 높아지는* 이점이 부각되어 최근에는 대부분의 modern language 들에서 람다 표현식을 지원하고 있다. (Java도 Java8 이후부터 도입)

**4. (Day 2 ~ 4 강의 관련) : Functional Programming 단원에서 배운 내용 중 가장 흥미로웠던 부분 1가지를 골라 조금 더 공부해보고 이해하신 내용을 적어주세요.** *

(ex: 함수형 프로그래밍 패러다임이란 무엇인가?, or 그외 자유 주제)

아직 드래프트 버전이라 블로그에 올리지는 않았습니다 :-) 깃헙 저장소에 올려둔 드래프트안으로 대체합니다 :
https://github.com/yenarue/TIL/blob/master/Kotlin/05_1_Functional_Paradigm.md

**5. (Day 5 강의 관련) : 2주차 Edu Tools 실습 예제 중 Taxi Park에 대한 답변 코드 (TaxiParkTask.kt 파일 부분)를 공유해주세요.** *

모두 해결하지 못하셨더라도 작성한 부분까지만 공유부탁드립니다.
제출된 파일:
TaxiParkTask - 김예나.kt

**토론시간에 주로 다뤄줬으면 하는 문제(중복체크가능)** *

- [ ] 문제 1
- [ ] 문제 2
- [x] 문제 3
- [x] 문제 4
- [ ] 문제 5
- [ ] 기타:

**이번 주차 파트 배분 스터디 방식에 대한 선호도 답변 부탁드립니다. 참고하여 다음 주차 진행 방식에 반영하겠습니다. (기타 의견 있으신 분은 기타 의견 부탁드립니다!)** *

- ( ) 다음주차도 동일한 방식 (파트 분배하여 맡은 파트에 대한 글 작성 및 공유) 진행을 원함
- (•) 다음주차는 1주차처럼 (파트 분배없이 전체에 대해 질문을 답변하는 방식) 진행을 원함
- ( ) 기타:

**파트 분배 방식으로 진행할 경우 아래 파트 중 더 공부해보고 싶은 주제를 골라주세요. (진행 여부는 2주차 온라인 스터디 때 공유드리겠습니다.)** *

- ◯ Properties 파트 (https://medium.com/@kbm1378/코틀린-입문-스터디-10-properties-26d85c745c4c 참고)
- ◉ OOP 파트 (https://medium.com/@kbm1378/코틀린-입문-스터디-11-object-oriented-programming-8e2e8db4dff 참고)
- ◯ Conventions/Operator overloading 파트 (https://medium.com/@kbm1378/코틀린-입문-스터디-12-conventions-a181f6509289 참고)
- ◯ 실습 파트 (https://medium.com/@kbm1378/코틀린-입문-스터디-13-실습-rationals-board-1fc925580abb 참고)

**이번주에 학습하시면서 이해가 안가셨거나 궁금하신 질문들을 모두 적어주세요** *

Kotlin에서의 함수형 연산들(`filter`, `map` 등..)이 내부적으로 Java8의 Stream 연산과 동일하게 동작하는 것인지 궁금합니다.

---