[코틀린 입문] 4주차 주간 질문

1 개의 메일

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

2019년 4월 28일 오전 1:42

[코틀린 입문] 4주차 주간 질문을(를) 작성해 주셔서 감사합니다.

사용자로부터 다음과 같은 메시지를 받았습니다.

응답 수정

[코틀린 입문] 4주차 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. inline 함수란 무엇이며, non-inline 과 어떤 차이가 있나요? *

Day2 강의 관련

- inline 함수란 : 인라인 함수는 호출된 로직에, 함수의 본문이 inline 되는 함수를 뜻한다. 실제로 함수를 호출하는 것이 아니라 함수의 본문 자체가 호출된 로직에 그대로 붙여넣어진다고 생각하면 된다. (바이트코드로 컴파일 시 해당 본문의 바이트코드가 붙여넣어진다)
- inline과 non-inline의 차이점 : inline을 사용했을때의 장점들을 먼저 살펴보자면
장점1. 오버헤드가 줄어든다 : 함수를 인라인으로 구현하지 않고 일반 함수로 구현한 경우에는 해당 함수 호출 시, Function Call Interface나 해당 Lambda에 대응되는 익명 클래스 개체(anonymous class object)가 생성되기 때문에 그에 따른 오버헤드가 생길 수 있다. 하지만 함수를 인라인으로 구현하게되면 함수 호출의 형태가 아니라, 호출된 논리 흐름의 일부로 동작할 수 있게 된다.
장점2. 간결성 + 가독성 향상 : 반복되는 구문을 줄여 조건문에만 집중할 수 있도록 함으로써 간결성과 가독성을 증가시킬 수 있다. 구문을 줄이되 성능은 유지하는 효과를 볼 수 있다.
단점1. 인라인 함수는 결과적으로 함수의 본문 전체가 그대로 붙여넣어지기 때문에 남용하면 오히려 프로젝트의 크기가 커질 수 있다. 그렇기 때문에, 혹시 인라인 함수를 구현하게 된다면, 작은 크기의 함수만 인라인 함수로 변경하는 것이 좋을 것이다.

2. Inline functions 단원에서 배운 내용 중 이론적 또는 실무적 관점에서 가장 흥미로웠던 부분 1가지를 골라 조금 더 공부해보고 이해하신 내용을 적어주세요. *

Day 1~2 강의 관련

-

3. Collection과 Sequence란 무엇이며, 어떻게 다르고, Sequence는 어떠한 경우에 유용한가요? *

Day 3 강의 관련

- Collections : Eager Evaluation
`filter`, `map`, `find`, `groupBy` 등의 Collections 확장 함수들은 `inline` 으로 정의되어있어 익명 클래스 객체 생성이 필요없기 때문에 관련 퍼포먼스 오베헤드가 줄어들 수 있다.
하지만, 또 또 다른 관점에서는 퍼포먼스 측면 문제가 발생한다. 이러한 확장 함수들은 매 실행 시 마다 그 함수가 실행된 결과를 가지는 Collections 를 반환한다. 그렇기 때문에 확장 함수들을 체이닝하여 호출하게 되면 체이닝 한 횟수만큼 Collections가 생성된다. 그로인해 *최종 결과물을 도출할 때 까지 불필요한 중간결과물을 생성* 한다는 비효율성이 나타난다.
- Sequences : Lazy Evaluation
Kotlin의 Sequence 는 Java8의 Stream 에 대응되는 개념이다. 즉, Lazy 관점으로 연산을 접근한다. 중간 결과물을 만들지 않고 쌓아두다가 마지막 'action' 연산시에 수행되는 것을 뜻한다.
불필요한 중간 결과물을 내지 않는다는 점에서 퍼포먼스 오버헤드가 줄어든다. 또한, terminal operation을 부르기 전까지는 실제 로직 동작이 이루어지지 않기 때문에 효율적인 연산 처리가 가능하다. 당장 연산할 필요 없이 필요할 때에만 연산하는 경우 유용하다.

4. Sequences단원에서 배운 내용 중 이론적 또는 실무적 관점에서 가장 흥미로웠던 부분 1가지를 골라 조금 더 공부해보고 이해하신 내용을 적어주세요. *

Day 3~4 강의 관련

Deferred Execution vs Lazy Evaluation (Lazy Execution)
두 용어가 거의 동일한 느낌인데 각각의 용어가 존재하는 이유가 있을 것 같아 좀 더 찾아봤습니다. 실제로 본질적으로 동일한 것을 가르키지만 뉘앙스의 차이가 있는 것으로 보입니다.
- Lazy means "don't do the work until you absolutely have to." 즉, Lazy는 절대적으로 해야만 하는 경우에 수행된다는 뜻을 가집니다.
- Deferred means "don't compute the result until the caller actually uses it." 즉, Deferred는 호출자가 그것을 실제로 사용할때까지 결과 연산을 하지 않는다는 뜻을 가집니다.
Deferred 가 좀 더 명확한 느낌이라 그런지 Deferred 를 사용하는 것이 더 좋다고 하는 사람들도 있는 것 같습니다만 결국 두 용어가 지향하는 바가 동일하므로 문맥에 따라, 뉘앙스에 따라 용어를 선택하면 될 것 같습니다. (그리고 Haskell이나 LISP와 같은 함수형 언어의 경우에 Lazy라는 표현을 더 자주한다고 하며 C계열에서는 Deferred라는 표현을 자주한다고 합니다. 함수형 패러다임으로 인해 최신 모던 언어들이 LISP의 영향를 많이 받고 있기 때문에 Lazy라는 용어가 더 자주 쓰이는 것으로 보입니다 - 뇌피셜.....)
https://stackoverflow.com/questions/2530755/difference-between-deferred-execution-and-lazy-evaluation-in-c-sharp

5. Lambda with Receiver란 무엇이며, 어떠한 이점이 있나요? *

Day 5 강의 관련

Lambda with receiver 는 Kotlin의 강력한 기능 중 하나이다. 이 기능은 Extension Function과 Lambda에 대한 아이디어의 결합(union)으로 볼 수 있다. Extension Lambda 라고도 불린다. *특정 멤버(receiver)를 비번하게 사용하는 로직* 일수록 lambda with receiver를 사용하는 것이 *간결성* 과 *가독성* 을 높힐 수 있는 방법이 될 수 있다.

6. let함수, with 함수, run 함수, apply 함수, also 함수가 무엇이며 어떠한 경우에 사용하면 좋을지 각각 이해한 내용을 적어주세요. *

Day 1, Day 5 강의 관련

- `with` : receiver 에 대한 멤버참조를 `this` 로 접근할 수 있어 특정 객체에 여러번 접근하며 멤버함수나 멤버변수에 접근하는 경우 로직을 간결하게 표현할 수 있다. `block`의 반환값을 반환하므로 다양하게 활용 가능하다.
- `iet` : `with` 과 매우 비슷하다. `iet`은 *객체를 receiver로 받기때문에 `?` 연산자를 통해 안전한 접근(safe access)이 가능* 하다. `block` 을 일반 Lambda로 받기때문에 객체를 접근할때에 꼭 `it`으로 접근해야 한다. 객체를 다른 함수의 argument로 여러번 사용하는 로직의 경우 유용하다.

- `run` : `with` 과 매우 비슷하지만 `with` 보다 좀 더 확장된 버전이다. `run` 은 *객체를 receiver로 받기 때문에 `?` 연산자를 통해 안전한 접근(safe access)이 가능* 하다!
- `apply` : 객체를 receiver로 받고 해당 객체의 타입을 반환하는 함수이다. *해당 객체의 타입을 반환하기 때문에 추가적인 연산을 체이닝 형태로 수행 할 수 있다*
- `also` : `apply` 와 유사하다. 하지만 `also` 는 * `block` 을 일반 Lambda로 받는다. 그렇기 때문에 꼭 객체에 `it` 으로 접근해주어야 한다. *

7. Lambda with Receiver 단원에서 배운 내용 중 이론적 또는 실무적 관점에서 가장 흥미로웠던 부분 1가지를 골라 조금 더 공부해보고 이해하신 내용을 적어주세요. *

Day 5 강의 관련

* 객체(Receiver) 전달 방식이 Lambda with Receiver인 경우
- 객체에 `this` 로 접근해야 한다. 물론 `this`는 생략 가능하다.
- 객체에 대한 상호작용이 있을 것임을 명시하는 경우이다.
- 객체의 멤버변수(프로퍼티)에 대한 setting을 권장한다. (`this`를 생략 가능하다는 점이 이를 표현한다)
* 객체(Receiver) 전달 방식이 일반 Lambda인 경우
- 객체에 `it`으로만 접근해야 한다. 생략 불가능하다.
- 객체에 대한 상호작용이 없을 수도 있다는 것을 암시한다.
- 객체의 멤버변수(프로퍼티)에 대한 setting을 지양한다. (`it`를 명시적으로 적어줘야 한다는 점이 이를 표현한다)

8. 이번 주차는 실습 과제가 없었습니다. 앞서 1~3주차에서 주어졌던 실습 과제 중 한번 더 풀어보고 싶거나, 혹은 시간관계로 진행하지 못하였던 문제가 있다면 1가지를 골라 풀어보고 작성하신 코드의 캡쳐 또는 .kt파일을 업로드해주세요.

제출된 파일:

토론시간에 주로 다뤄줬으면 하는 문제(중복체크가능) *

- [ ] 문제 1
- [x] 문제 2
- [ ] 문제 3
- [x] 문제 4
- [x] 문제 5
- [ ] 문제 6
- [x] 문제 7
- [ ] 문제 8
- [ ] 기타:

이번주에 학습하시면서 이해가 안가셨거나 궁금하신 질문들을 모두 적어주세요 *

Lambda with the receiver 는 객체에 대한 멤버참조를 하는 경우 `this` 참조를 생략할 수 있으므로 유용하다. 하지만 객체가 특정 함수의 인수(argument)로 전달되어야 하는 경우에는 일반 Lambda 가 더 유용하다.
=> 그냥 Lambda with the receiver에서 this를 써서 수행해도 가능은 하지 않나요? 가능은 하지만 지양하는 건지 궁금합니다!
