

/*

검색어를 기반으로 검색 요청을 하는 서비스 함수를 만들어보자!

행동 분석

1. 검색 파라미터를 넘겨 받는다.
2. 검색 파라미터를 검증한다.
3. 외부 검색 API 를 호출한다.
4. 통계를 위해 검색 결과를 저장한다.
4. 검색 결과를 반환한다.

TC

1. 성공 : (정답) -> (정답) ?
2. 실패
 - page 가 0보다 작으면 검색을 실패한다.
 - pageSize 가 0보다 작으면 검색을 실패한다.
 - pageSize 가 100 보다 크면 실패한다.
 - 키워드가 비어있으면 실패한다.

*/

/*

검색어를 기반으로 검색 요청을 하는 서비스 함수를 만들어보자!
행동 분석

1. 검색 파라미터를 넘겨 받는다.
2. 검색 파라미터를 검증한다.
3. 외부 검색 API 를 호출한다.
4. 통계를 위해 검색 결과를 저장한다.
4. 검색 결과를 반환한다.

TC

1. 성공 : (정답) -> (정답) ?
2. 실패
 - page 가 0보다 작으면 검색을 실패한다.
 - pageSize 가 0보다 작으면 검색을 실패한다.
 - pageSize 가 100 보다 크면 실패한다.
 - 키워드가 비어있으면 실패한다.

*/

```
data class SearchRequest(  
    val keyword: String,  
    val page: Int,  
    val pageSize: Int,  
)  
  
data class SearchResult(  
    val keyword: String,  
    val count: Int,  
)  
  
interface SearchClient {  
    fun search(request: SearchRequest): SearchResult  
}  
  
interface SearchHistoryRepository {  
    fun appendHistory(result: SearchResult)  
}
```

TDD in real world



```
class SearchService(  
    private val searchClient: SearchClient,  
    private val searchHistoryRepository: SearchHistoryRepository,  
) {  
    private val log = LoggerFactory.getLogger(javaClass)  
  
    fun search(request: SearchRequest): SearchResult {  
        val (keyword, page, pageSize) = request  
        if (page < 0) throw RuntimeException("페이지가 너무 작습니다")  
        if (pageSize < 0) throw RuntimeException("조회 갯수가 너무 작습니다")  
        if (pageSize > 100) throw RuntimeException("조회 갯수가 너무 큼니다")  
        if (keyword.isEmpty()) throw RuntimeException("검색어가 없습니다")  
        val result = searchClient.search(request = request)  
        try {  
            searchHistoryRepository.appendHistory(result = result)  
        } catch (e: Throwable) {  
            log.warn("Exception : {}", e.message, e)  
        }  
        return result  
    }  
}
```

TDD in real world

향해+

```
class SearchService(  
    private val searchClient: SearchClient,  
    private val searchHistoryRepository: SearchHistoryRepository,  
) {  
    private val log = LoggerFactory.getLogger(javaClass)  
  
    fun search(request: SearchRequest): SearchResult {  
        val (keyword, page, pageSize) = request  
        if (page < 0) throw RuntimeException("페이지가 너무 작습니다")  
        if (pageSize < 0) throw RuntimeException("조회 갯수가 너무 작습니다")  
        if (pageSize > 100) throw RuntimeException("조회 갯수가 너무 큼니다")  
        if (keyword.isEmpty()) throw RuntimeException("검색어가 없습니다")  
        val result = searchClient.search(request = request)  
        try {  
            searchHistoryRepository.appendHistory(result = result)  
        } catch (e: Throwable) {  
            log.warn("Exception : {}", e.message, e)  
        }  
        return result  
    }  
}
```

```
class SearchClientStub : SearchClient {  
    override fun search(request: SearchRequest): SearchResult {  
        return SearchResult(keyword = "소고기", count = 1)  
    }  
}  
  
class SearchHistoryRepositoryStub : SearchHistoryRepository {  
    override fun appendHistory(result: SearchResult) = Unit  
}  
  
class SearchServiceTest {  
    private val searchService = SearchService(  
        searchClient = SearchClientStub(),  
        searchHistoryRepository = SearchHistoryRepositoryStub(),  
    )  
  
    @Test  
    fun `page가 0보다 작으면 검색을 실패한다`() {  
        assertThatThrownBy {  
            val request = SearchRequest(keyword = "소고기", page = -1, pageSize = 3)  
            searchService.search(request = request)  
        }.isInstanceOf(RuntimeException::class.java)  
    }  
  
    @Test  
    fun `pageSize 가 0보다 작으면 검색을 실패한다`() {  
        assertThatThrownBy {  
            val request = SearchRequest(keyword = "소고기", page = 1, pageSize = -1)  
            searchService.search(request = request)  
        }.isInstanceOf(RuntimeException::class.java)  
    }  
  
    @Test  
    fun `pageSize 가 100보다 크면 실패한다`() {  
        assertThatThrownBy {  
            val request = SearchRequest(keyword = "소고기", page = 1, pageSize = 101)  
            searchService.search(request = request)  
        }.isInstanceOf(RuntimeException::class.java)  
    }  
  
    @Test  
    fun `키워드가 비어있으면 실패한다`() {  
        assertThatThrownBy {  

```




```
class SearchService(  
    private val searchClient: SearchClient,  
    private val searchHistoryRepository: SearchHistoryRepository,  
) {  
    private val log = LoggerFactory.getLogger(javaClass)  
  
    fun search(request: SearchRequest): SearchResult {  
        val (keyword, page, pageSize) = request  
        if (page < 0) throw RuntimeException("페이지가 너무 작습니다")  
        if (pageSize < 0) throw RuntimeException("조회 갯수가 너무 작습니다")  
        if (pageSize > 100) throw RuntimeException("조회 갯수가 너무 큼니다")  
        if (keyword.isEmpty()) throw RuntimeException("검색어가 없습니다")  
        val result = searchClient.search(request = request)  
        try {  
            searchHistoryRepository.appendHistory(result = result)  
        } catch (e: Throwable) {  
            log.warn("Exception : {}", e.message, e)  
        }  
        return result  
    }  
}
```

이 함수의 책임

1. Parameter Object 에 대한 검증
2. 검색 API 에 대한 요청
3. 검색 결과 저장
4. 검색 결과 반환

```
data class SearchRequest(  
    val keyword: String,  
    val page: Int,  
    val pageSize: Int,  
) {  
    @Throws(RuntimeException::class)  
    fun validate() {  
        if (page < 0) throw RuntimeException("페이지가 너무 작습니다")  
        if (pageSize < 0) throw RuntimeException("조회 갯수가 너무 작습니다")  
        if (pageSize > 100) throw RuntimeException("조회 갯수가 너무 큼니다")  
        if (keyword.isEmpty()) throw RuntimeException("검색어가 없습니다")  
    }  
}
```

```
class SearchService(  
    private val searchClient: SearchClient,  
    private val searchHistoryRepository: SearchHistoryRepository,  
) {  
    private val log = LoggerFactory.getLogger(javaClass)  
      
    // 정책 추가 : request 에 대한 validation 은 앞서 진행되어야 함  
    fun search(request: SearchRequest): SearchResult {  
        val result = searchClient.search(request = request)  
        try {  
            searchHistoryRepository.appendHistory(result = result)  
        } catch (e: Throwable) {  
            log.warn("Exception : {}", e.message, e)  
        }  
        return result  
    }  
}
```