Collections vs Sequences

Extensions on collections are inlined

```
inline fun <T> Iterable<T>.filter(predicate: (T) -> Boolean): List<T>
inline fun <T, R> Iterable<T>.map(transform: (T) -> R): List<R>
inline fun <T> Iterable<T>.any(predicate: (T) -> Boolean): Boolean
inline fun <T> Iterable<T>.find(predicate: (T) -> Boolean): T?
inline fun <T, K> Iterable<T>.groupBy(keySelector: (T) -> K): Map<K, List<T>>
...
```

Intermediate operations return new collections as a result

```
inline fun <T> Iterable<T>.filter(predicate: (T) -> Boolean): List<T>
inline fun <T, R> Iterable<T>.map(transform: (T) -> R): List<R>
inline fun <T, K> Iterable<T>.groupBy(keySelector: (T) -> K): Map<K, List<T>>
...
```

How many collections are created while running the code below?

```
val list = listOf(1, 2, -3)
val maxOddSquare = list
        .map { it * it }
        .filter { it % 2 == 1 }
        .max()
```

?

How many collections are created while running the code below?

```
val list = listOf(1, 2, -3)
val maxOddSquare = list
        .map { it * it }
        .filter { it % 2 == 1 }
        .max()
```

3

Counting intermediate collections…

```
val list = listOf(1, 2, 3)                    // [1, 2, 3]

val mapResult = list.map { it * it }          // [1, 4, 9]

val filterResult = mapResult
        .filter { it % 2 == 1 }               // [1, 9]

val maxOddSquare = filterResult.max()         // 9
```

Operations on collections

- lambdas are inlined
  (no performance overhead)
- *but:* intermediate collections
  are created for chained calls

Sequences

Collections vs Sequences

eager vs lazy evaluation

From List to Sequence

```
val list = listOf(1, 2, -3)
val maxOddSquare = list
        .map { it * it }
        .filter { it % 2 == 1 }
        .max()
```

From List to Sequence

```
val list = listOf(1, 2, -3)
val maxOddSquare = list
        .asSequence()
        .map { it * it }
        .filter { it % 2 == 1 }
        .max()
```

Operations on sequences

```
val sequence = listOf(1, 2, 3).asSequence()

val mapResult = sequence.map { it * it }
```

kotlin.sequences.TransformingSequence@65b54208

```
val filterResult = mapResult
        .filter { it % 2 == 1 }
```

kotlin.sequences.FilteringSequence@38af3868

```
val maxOddSquare = filterResult.max()
```

9