40009 ExerciseTypes.PPT11

Kotlin Journey Planning

Submitters

anb122	Adithya Narayanan

```
Final Tests: Summary for anb122 of j1
PPT 24
-------------------------------------

  Public Tests:
    Compiles:     1 / 1
    Tests Pass:   1 / 1
    Style Check:  1 / 1 ✓

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_autumn/kotlinjourneyplanner_anb122.git
Commit ID: 986a1
```

Tests: 3/4

Correctness: 3/3

Quality: 1/3

7/10

-1 for not add tests to test all the code you've written

-1 for very long lines. try to stick to the 80 char limit.

-1 for no comments.

can make this private ?

```kotlin
package journeyplan

import java.util.*
import kotlin.collections.ArrayList

class SubwayMap(var subwaySegmentList: List<Segment>) {
  fun routesFrom(origin: Station, destination: Station): List<Route> {
    return recursiveHelper(Route(listOf()), origin, destination).sortedBy { it.duration() }
  }

  private fun recursiveHelper(routeToPass: Route, origin: Station, destination: Station): List<Route> {
    if (origin == destination) {
      return listOf(routeToPass)
    }

    var nextSegments = this.subwaySegmentList
      .filter { x -> x.station1 == origin }
      .filter { x -> !(x in routeToPass.segmentList) }

    if (nextSegments.size == 0) {
      return emptyList()
    }

    return nextSegments.flatMap { x -> recursiveHelper(routeToPass.add(Route(listOf(x))), x.station2, destination) }
  }
}

class Route(var segmentList: List<Segment>) {
  fun add(route2: Route): Route {
    var arraylistcurrentlist = ArrayList<Segment>(segmentList)
    var arraylistroutetoadd = ArrayList<Segment>(route2.segmentList)
    var listsum = arraylistcurrentlist + arraylistroutetoadd
    return Route(listsum.toList())
  }

  fun listOfLines(): ArrayList<Segment> {
    var listOfLines = ArrayList<Segment>()
    listOfLines.add((this.segmentList.first()))
    for (i in 1..this.segmentList.size - 1) {
      if (segmentList[i].lineOfSegment == segmentList[i - 1].lineOfSegment) {
        var segmentToManipulate = Segment(
          listOfLines.last().station1,
          segmentList[i].station2,
          segmentList[i].lineOfSegment,
          listOfLines.last().minutes + segmentList[i].minutes
        )
        listOfLines.removeAt(listOfLines.size - 1)
        listOfLines.add(segmentToManipulate)
      } else {
        listOfLines.add(segmentList[i])
      }
    }
    return listOfLines
  }

  override fun toString(): String {

    var listOfLines = listOfLines()

    var OutputString =
      this.segmentList.first().station1.toString() + " to " + this.segmentList.last().station2.toString() + " - " + this.duration() + " minutes, " + this.numChanges() + " changes"
```

-1 for long lines.

```kotlin

    for (i in listOfLines) {
      OutputString += "\n - " + i.station1.toString() + " to " + i.station2.toString() + " by " + i.lineOfSegment.toString()
    }

    return OutputString
  }

  fun numChanges(): Int {
    return listOfLines().size - 1
  }

  fun duration(): Int {
    var count = 0
    for (i in this.segmentList) {
      count += i.minutes
    }
    return count
  }
}

fun londonUnderground(): SubwayMap {
  val northernLine = Line("Northern")
  val victoriaLine = Line("Victoria")
  val centralLine = Line("Central")

  val highgate = Station("Highgate")
  val archway = Station("Archway")
  val tufnellPark = Station("Tufnell Park")
  val kentishTown = Station("Kentish Town")
  val camden = Station("Camden Town")
  val euston = Station("Euston")
  val warrenStreet = Station("Warren Street")
  val oxfordCircus = Station("Oxford Circus")
  val bondStreet = Station("Bond Street")

  return SubwayMap(
    listOf(
      Segment(highgate, archway, victoriaLine, 4),
      Segment(archway, tufnellPark, victoriaLine, 6),
      Segment(tufnellPark, kentishTown, northernLine, 8),
      Segment(tufnellPark, camden, victoriaLine, 5),
      Segment(camden, euston, centralLine, 6),
      Segment(camden, warrenStreet, northernLine, 7),
      Segment(euston, oxfordCircus, centralLine, 10),
      Segment(camden, bondStreet, victoriaLine, 7),

      Segment(archway, highgate, victoriaLine, 4),
      Segment(tufnellPark, archway, victoriaLine, 6),
      Segment(kentishTown, archway, northernLine, 8),
      Segment(camden, tufnellPark, victoriaLine, 5),
      Segment(euston, camden, centralLine, 6),
      Segment(warrenStreet, camden, northernLine, 7),
      Segment(oxfordCircus, euston, centralLine, 10),
      Segment(bondStreet, camden, victoriaLine, 7)
    )
  )
}

// Garbage code
// for (i in segmentList) {
//   if (i.station1 == origin) {
//     routeList.add(Route(listOf(i)))
//   }
// }
// println(routeList)
// if (routeList[0].segmentList[0].station1 == origin && routeList[0].segmentList[0].station2 == destination) {
//  return routeList.toList()
// } else {
//  return (routeList + routesFrom())
// }
//
// private fun routesFrom2(origin: Station, destination: Station, routesList: ArrayList<Route>): List<Route> {
//  var routeList = ArrayList<Route>()
//  var stationList = ArrayList<Station>()
//  for (i in routesList) {
//    if (i.segmentList)
//  }
//  for (i: Segment in segmentList){
//    if (i.station1 == origin && i.station2 == destination){
//      return listOf(Route(listOf(i)))
//    } else if (i.station1 == origin && (segmentList.map {x -> x.station2 in }).isEmpty()){
//    }
//  }
// }

fun main() {
  var fakeUnderground = londonUnderground()
  println(fakeUnderground.routesFrom(Station("Highgate"), Station("Kentish Town")))
}
```

```kotlin
package journeyplan

class Station(var n: String) {
  override fun toString(): String {
    return n
  }
}

class Line(var lineName: String) {
  override fun toString(): String {
    return lineName + " Line"
  }
}

class Segment(var station1: Station, var station2: Station, var lineOfSegment: Line, var minutes: Int)
```

```kotlin
package journeyplan

import org.junit.Test
import kotlin.test.assertEquals

class RoutePlannerTest {

  val northernLine = Line("Northern")
  val victoriaLine = Line("Victoria")
  val centralLine = Line("Central")

  val highgate = Station("Highgate")
  val archway = Station("Archway")
  val tufnellPark = Station("Tufnell Park")
  val kentishTown = Station("Kentish Town")
  val camden = Station("Camden Town")
  val euston = Station("Euston")
  val warrenStreet = Station("Warren Street")
  val oxfordCircus = Station("Oxford Circus")
  val bondStreet = Station("Bond Street")

  val tufnellParkToHighgate =
    Route(
      listOf(
        Segment(tufnellPark, archway, northernLine, 3),
        Segment(archway, highgate, northernLine, 3)
      )
    )

  val highgateToOxfordCircus =
    Route(
      listOf(
        Segment(highgate, archway, northernLine, 3),
        Segment(archway, kentishTown, northernLine, 3),
        Segment(kentishTown, camden, northernLine, 3),
        Segment(camden, euston, northernLine, 3),
        Segment(euston, warrenStreet, victoriaLine, 3),
        Segment(warrenStreet, oxfordCircus, victoriaLine, 3)
      )
    )

  val camdenToBondStreet =
    Route(
      listOf(
        Segment(camden, euston, northernLine, 3),
        Segment(euston, warrenStreet, victoriaLine, 3),
        Segment(warrenStreet, oxfordCircus, victoriaLine, 3),
        Segment(oxfordCircus, bondStreet, centralLine, 2)
      )
    )

  @Test
  fun `can calculate number of changes`() {
    assertEquals(0, tufnellParkToHighgate.numChanges())
    assertEquals(1, highgateToOxfordCircus.numChanges())
    assertEquals(2, camdenToBondStreet.numChanges())
  }

  @Test
  fun `can calculate total duration`() {
    assertEquals(6, tufnellParkToHighgate.duration())
    assertEquals(18, highgateToOxfordCircus.duration())
    assertEquals(11, camdenToBondStreet.duration())
  }

  @Test
```

```kotlin
  fun `toString omits calling points`() {
    assertEquals(
      """
                Tufnell Park to Highgate - 6 minutes, 0 changes
                 - Tufnell Park to Highgate by Northern Line
      """.trimIndent(),
      tufnellParkToHighgate.toString()
    )
  }

  @Test
  fun `toString shows changes`() {
    assertEquals(
      """
                Highgate to Oxford Circus - 18 minutes, 1 changes
                 - Highgate to Euston by Northern Line
                 - Euston to Oxford Circus by Victoria Line
      """.trimIndent(),
      highgateToOxfordCircus.toString()
    )
  }
}
```

- Add tests for routesFrom.

- Add tests for listOfLines().

```
package journeyplan

import org.junit.Test
import kotlin.test.assertEquals

class TravelModelTest {

  @Test
  fun `printing stations shows their names`() {
    assertEquals("South Kensington", Station("South Kensington").toString())
    assertEquals("Knightsbridge", Station("Knightsbridge").toString())
  }

  @Test
  fun `printing lines shows their names`() {
    assertEquals("District Line", Line("District").toString())
    assertEquals("Circle Line", Line("Circle").toString())
  }
}
```

```
-------- Test Output --------
Running LabTS build... (Fri 24 Nov 18:28:25 UTC 2023)

Submission summary...
You made 6 commits
  - 7675a0c PPT completed [4 files changed, 257 insertions, 97 deletions]
  - 1799567 Style corrections [2 files changed, 12 insertions, 13 deletions]
  - 6ebb700 Style changes [1 file changed, 2 deletions]
  - d2f21f1 Style changed [2 files changed, 11 insertions, 2 deletions]
  - 0674f14 Return sorted list and style changes [1 file changed, 5 insertions, 11 deletions]
  - 986a1f0 Trying to fix style [1 file changed, 1 insertion, 1 deletion]

Preparing...

BUILD SUCCESSFUL in 681ms

Compiling...Path for java installation '/usr/lib/jvm/openjdk-17' (Common Linux Locations) does not contain a java executable

BUILD SUCCESSFUL in 10s

Running tests...Path for java installation '/usr/lib/jvm/openjdk-17' (Common Linux Locations) does not contain a java executable


journeyplan.RoutePlannerTest > toString shows changes PASSED

journeyplan.RoutePlannerTest > toString omits calling points PASSED

journeyplan.RoutePlannerTest > can calculate total duration PASSED

journeyplan.RoutePlannerTest > can calculate number of changes PASSED

journeyplan.TravelModelTest > printing lines shows their names PASSED

journeyplan.TravelModelTest > printing stations shows their names PASSED

BUILD SUCCESSFUL in 1s

Checking code style...
BUILD SUCCESSFUL in 2s
Finished auto test. (Fri 24 Nov 18:29:03 UTC 2023)

-------- Test Errors --------

```

```
Test Preview: Summary for anb122 of j1
PPT 24
--------------------------------------

  Public Tests:
    Compiles:     1 / 1
    Tests Pass:   1 / 1
    Style Check:  1 / 1

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_autumn/kotlinjourneyplanner_anb122.git
Commit ID: 986a1
```

```kotlin
package journeyplan

import java.util.*
import kotlin.collections.ArrayList

class SubwayMap(var subwaySegmentList: List<Segment>) {
  fun routesFrom(origin: Station, destination: Station): List<Route> {
    return recursiveHelper(Route(listOf()), origin, destination).sortedBy { it.duration() }
  }

  private fun recursiveHelper(routeToPass: Route, origin: Station, destination: Station): List<Route> {
    if (origin == destination) {
      return listOf(routeToPass)
    }

    var nextSegments = this.subwaySegmentList
      .filter { x -> x.station1 == origin }
      .filter { x -> !(x in routeToPass.segmentList) }

    if (nextSegments.size == 0) {
      return emptyList()
    }

    return nextSegments.flatMap { x -> recursiveHelper(routeToPass.add(Route(listOf(x))), x.station2, destination) }
  }
}

class Route(var segmentList: List<Segment>) {
  fun add(route2: Route): Route {
    var arraylistcurrentlist = ArrayList<Segment>(segmentList)
    var arraylistroutetoadd = ArrayList<Segment>(route2.segmentList)
    var listsum = arraylistcurrentlist + arraylistroutetoadd
    return Route(listsum.toList())
  }

  fun listOfLines(): ArrayList<Segment> {
    var listOfLines = ArrayList<Segment>()
    listOfLines.add((this.segmentList.first()))
    for (i in 1..this.segmentList.size - 1) {
      if (segmentList[i].lineOfSegment == segmentList[i - 1].lineOfSegment) {
        var segmentToManipulate = Segment(
          listOfLines.last().station1,
          segmentList[i].station2,
          segmentList[i].lineOfSegment,
          listOfLines.last().minutes + segmentList[i].minutes
        )
        listOfLines.removeAt(listOfLines.size - 1)
        listOfLines.add(segmentToManipulate)
      } else {
        listOfLines.add(segmentList[i])
      }
    }
    return listOfLines
  }

  override fun toString(): String {

    var listOfLines = listOfLines()

    var OutputString =
      this.segmentList.first().station1.toString() + " to " + this.segmentList.last().station2.toString() + " - " + this.duration() + " minutes, " + this.numChanges() + " changes"
```

```kotlin

    for (i in listOfLines) {
      OutputString += "\n - " + i.station1.toString() + " to " + i.station2.toString() + " by " + i.lineOfSegment.toString()
    }

    return OutputString
  }

  fun numChanges(): Int {
    return listOfLines().size - 1
  }

  fun duration(): Int {
    var count = 0
    for (i in this.segmentList) {
      count += i.minutes
    }
    return count
  }
}

fun londonUnderground(): SubwayMap {
  val northernLine = Line("Northern")
  val victoriaLine = Line("Victoria")
  val centralLine = Line("Central")

  val highgate = Station("Highgate")
  val archway = Station("Archway")
  val tufnellPark = Station("Tufnell Park")
  val kentishTown = Station("Kentish Town")
  val camden = Station("Camden Town")
  val euston = Station("Euston")
  val warrenStreet = Station("Warren Street")
  val oxfordCircus = Station("Oxford Circus")
  val bondStreet = Station("Bond Street")

  return SubwayMap(
    listOf(
      Segment(highgate, archway, victoriaLine, 4),
      Segment(archway, tufnellPark, victoriaLine, 6),
      Segment(tufnellPark, kentishTown, northernLine, 8),
      Segment(tufnellPark, camden, victoriaLine, 5),
      Segment(camden, euston, centralLine, 6),
      Segment(camden, warrenStreet, northernLine, 7),
      Segment(euston, oxfordCircus, centralLine, 10),
      Segment(camden, bondStreet, victoriaLine, 7),

      Segment(archway, highgate, victoriaLine, 4),
      Segment(tufnellPark, archway, victoriaLine, 6),
      Segment(kentishTown, archway, northernLine, 8),
      Segment(camden, tufnellPark, victoriaLine, 5),
      Segment(euston, camden, centralLine, 6),
      Segment(warrenStreet, camden, northernLine, 7),
      Segment(oxfordCircus, euston, centralLine, 10),
      Segment(bondStreet, camden, victoriaLine, 7)
    )
  )
}

// Garbage code
// for (i in segmentList) {
//  if (i.station1 == origin) {
//    routeList.add(Route(listOf(i)))
//  }
// }
// println(routeList)
// if (routeList[0].segmentList[0].station1 == origin && routeList[0].segmentList[0].station2 == destination) {
//  return routeList.toList()
// } else {
//  return (routeList + routesFrom())
// }
//
// private fun routesFrom2(origin: Station, destination: Station, routesList: ArrayList<Route>): List<Route> {
//  var routeList = ArrayList<Route>()
//  var stationList = ArrayList<Station>()
//  for (i in routesList) {
//    if (i.segmentList)
//  }
//  for (i: Segment in segmentList){
//    if (i.station1 == origin && i.station2 == destination){
//      return listOf(Route(listOf(i)))
//    } else if (i.station1 == origin && (segmentList.map {x -> x.station2 in }).isEmpty()){
//    }
//  }
// }

fun main() {
  var fakeUnderground = londonUnderground()
  println(fakeUnderground.routesFrom(Station("Highgate"), Station("Kentish Town")))
}
```

```
package journeyplan

class Station(var n: String) {
  override fun toString(): String {
    return n
  }
}

class Line(var lineName: String) {
  override fun toString(): String {
    return lineName + " Line"
  }
}

class Segment(var station1: Station, var station2: Station, var lineOfSegment: Line, var minutes: Int)
```

```kotlin
package journeyplan

import org.junit.Test
import kotlin.test.assertEquals

class RoutePlannerTest {

  val northernLine = Line("Northern")
  val victoriaLine = Line("Victoria")
  val centralLine = Line("Central")

  val highgate = Station("Highgate")
  val archway = Station("Archway")
  val tufnellPark = Station("Tufnell Park")
  val kentishTown = Station("Kentish Town")
  val camden = Station("Camden Town")
  val euston = Station("Euston")
  val warrenStreet = Station("Warren Street")
  val oxfordCircus = Station("Oxford Circus")
  val bondStreet = Station("Bond Street")

  val tufnellParkToHighgate =
    Route(
      listOf(
        Segment(tufnellPark, archway, northernLine, 3),
        Segment(archway, highgate, northernLine, 3)
      )
    )

  val highgateToOxfordCircus =
    Route(
      listOf(
        Segment(highgate, archway, northernLine, 3),
        Segment(archway, kentishTown, northernLine, 3),
        Segment(kentishTown, camden, northernLine, 3),
        Segment(camden, euston, northernLine, 3),
        Segment(euston, warrenStreet, victoriaLine, 3),
        Segment(warrenStreet, oxfordCircus, victoriaLine, 3)
      )
    )

  val camdenToBondStreet =
    Route(
      listOf(
        Segment(camden, euston, northernLine, 3),
        Segment(euston, warrenStreet, victoriaLine, 3),
        Segment(warrenStreet, oxfordCircus, victoriaLine, 3),
        Segment(oxfordCircus, bondStreet, centralLine, 2)
      )
    )

  @Test
  fun `can calculate number of changes`() {
    assertEquals(0, tufnellParkToHighgate.numChanges())
    assertEquals(1, highgateToOxfordCircus.numChanges())
    assertEquals(2, camdenToBondStreet.numChanges())
  }

  @Test
  fun `can calculate total duration`() {
    assertEquals(6, tufnellParkToHighgate.duration())
    assertEquals(18, highgateToOxfordCircus.duration())
    assertEquals(11, camdenToBondStreet.duration())
  }

  @Test
```

```kotlin
  fun `toString omits calling points`() {
    assertEquals(
      """
                Tufnell Park to Highgate - 6 minutes, 0 changes
                 - Tufnell Park to Highgate by Northern Line
      """.trimIndent(),
      tufnellParkToHighgate.toString()
    )
  }

  @Test
  fun `toString shows changes`() {
    assertEquals(
      """
                Highgate to Oxford Circus - 18 minutes, 1 changes
                 - Highgate to Euston by Northern Line
                 - Euston to Oxford Circus by Victoria Line
      """.trimIndent(),
      highgateToOxfordCircus.toString()
    )
  }
}
```

```
package journeyplan

import org.junit.Test
import kotlin.test.assertEquals

class TravelModelTest {

  @Test
  fun `printing stations shows their names`() {
    assertEquals("South Kensington", Station("South Kensington").toString())
    assertEquals("Knightsbridge", Station("Knightsbridge").toString())
  }

  @Test
  fun `printing lines shows their names`() {
    assertEquals("District Line", Line("District").toString())
    assertEquals("Circle Line", Line("Circle").toString())
  }
}
```

```
-------- Test Output --------
Running LabTS build... (Fri 24 Nov 18:28:25 UTC 2023)

Submission summary...
You made 6 commits
  - 7675a0c PPT completed [4 files changed, 257 insertions, 97 deletions]
  - 1799567 Style corrections [2 files changed, 12 insertions, 13 deletions]
  - 6ebb700 Style changes [1 file changed, 2 deletions]
  - d2f21f1 Style changed [2 files changed, 11 insertions, 2 deletions]
  - 0674f14 Return sorted list and style changes [1 file changed, 5 insertions, 11 deletions]
  - 986a1f0 Trying to fix style [1 file changed, 1 insertion, 1 deletion]

Preparing...

BUILD SUCCESSFUL in 681ms

Compiling...Path for java installation '/usr/lib/jvm/openjdk-17' (Common Linux Locations) does not contain a java executable

BUILD SUCCESSFUL in 10s

Running tests...Path for java installation '/usr/lib/jvm/openjdk-17' (Common Linux Locations) does not contain a java executable


journeyplan.RoutePlannerTest > toString shows changes PASSED

journeyplan.RoutePlannerTest > toString omits calling points PASSED

journeyplan.RoutePlannerTest > can calculate total duration PASSED

journeyplan.RoutePlannerTest > can calculate number of changes PASSED

journeyplan.TravelModelTest > printing lines shows their names PASSED

journeyplan.TravelModelTest > printing stations shows their names PASSED

BUILD SUCCESSFUL in 1s

Checking code style...
BUILD SUCCESSFUL in 2s
Finished auto test. (Fri 24 Nov 18:29:03 UTC 2023)

-------- Test Errors --------

```