

```

../solution/src/main/kotlin/journeyplan/RoutePlanner.kt    Fri Nov 19 21:26:20 2021    1
1: package journeyplan
2:
3: class Route(val segments: List<Segment>) {
4:
5:     override fun toString(): String {
6:         val result = StringBuilder()
7:
8:         result.append("${segments.first().from} to ${segments.last().to} - ${duration()}
minutes, ${numChanges()} changes\n")
9:
10:        var current = segments.first()
11:        result.append(" - ${current.from} to ")
12:
13:        for (seg in segments) {
14:            if (seg.line != current.line) {
15:                result.append("${seg.from} by ${current.line}\n")
16:                result.append(" - ${seg.from} to ")
17:                current = seg
18:            }
19:        }
20:
21:        result.append("${segments.last().to} by ${current.line}")
22:
23:        return result.toString()
24:    }
25:
26:    fun duration(): Int = segments.sumOf(Segment::minutes)
27:
28:    fun numChanges(): Int = changes().size
29:
30:    fun allInterchangesOpen(): Boolean = changes().all(Station::canInterchange)
31:
32:    private fun changes(): List<Station> =
33:        segments.zipWithNext().filter { (l1, l2) -> l1.line != l2.line }.map { (l1, _) ->
l1.to }
34: }
35:
36: class SubwayMap(private val segments: List<Segment>) {
37:
38:     private val nowhereToGo = listOf(Route(emptyList()))
39:
40:     fun routesFrom(
41:         startingPoint: Station,
42:         destination: Station,
43:         optimisingFor: (Route) -> Int = Route::duration
44:     ): List<Route> =
45:         routesFrom(startingPoint, destination, alreadyVisited = setOf(startingPoint))
46:         .filter(Route::allInterchangesOpen)
47:         .sortedBy(optimisingFor)
48:
49:     private fun routesFrom(startingPoint: Station, destination: Station, alreadyVisited:
Set<Station>): List<Route> {
50:
51:         if (startingPoint == destination) {
52:             return nowhereToGo
53:         }
54:
55:         val nextSteps = segments.filter { seg -> seg.from == startingPoint }
56:             .filterNot { seg -> seg.to in alreadyVisited }
57:             .filterNot { seg -> seg.line.isSuspended() }
58:
59:         return nextSteps.flatMap { seg ->
60:             routesFrom(seg.to, destination, alreadyVisited + startingPoint)

```

```

../solution/src/main/kotlin/journeyplan/RoutePlanner.kt    Fri Nov 19 21:26:20 2021    2
61:         .map { r -> Route(listOf(seg) + r.segments) }
62:     }
63: }
64: }

```

```

../solution/src/main/kotlin/journeyplan/TravelModel.kt      Fri Nov 19 21:26:20 2021      1
1: package journeyplan
2:
3: class Station(private val name: String) {
4:
5:     private var open = true
6:
7:     fun canInterchange(): Boolean {
8:         return open
9:     }
10:
11:     fun open() {
12:         open = true
13:     }
14:
15:     fun close() {
16:         open = false
17:     }
18:
19:     override fun toString(): String {
20:         return name
21:     }
22: }
23:
24: class Line(private val name: String) {
25:
26:     private var suspended: Boolean = false
27:
28:     override fun toString(): String {
29:         return "$name Line"
30:     }
31:
32:     fun suspend() {
33:         this.suspended = true
34:     }
35:
36:     fun resume() {
37:         suspended = false
38:     }
39:
40:     fun isSuspended(): Boolean {
41:         return suspended
42:     }
43: }
44:
45: class Segment(val from: Station, val to: Station, val line: Line, val minutes: Int)

```

```

../solution/src/test/kotlin/journeyplan/ExtensionsTest.kt  Fri Nov 19 21:26:20 2021      1
1: package journeyplan
2:
3: import org.junit.Assert.assertEquals
4: import org.junit.Assert.assertNotNull
5: import org.junit.Assert.assertNull
6: import org.junit.Assert.assertTrue
7: import org.junit.Test
8:
9: class ExtensionsTest {
10:
11:     /** Uncomment the code in this file if you do the extensions */
12:
13:     val piccadillyLine = Line("Piccadilly")
14:     val victoriaLine = Line("Victoria")
15:     val districtLine = Line("District")
16:
17:     val southKensington = Station("South Kensington")
18:     val knightsbridge = Station("Knightsbridge")
19:     val hydeParkCorner = Station("Hyde Park Corner")
20:     val greenPark = Station("Green Park")
21:     val oxfordCircus = Station("Oxford Circus")
22:     val victoria = Station("Victoria")
23:     val sloaneSquare = Station("Sloane Square")
24:
25:     fun londonUnderground(): SubwayMap = SubwayMap(
26:         listOf(
27:             Segment(southKensington, knightsbridge, piccadillyLine, 3),
28:             Segment(knightsbridge, hydeParkCorner, piccadillyLine, 4),
29:             Segment(hydeParkCorner, greenPark, piccadillyLine, 2),
30:             Segment(greenPark, oxfordCircus, victoriaLine, 1),
31:             Segment(greenPark, victoria, victoriaLine, 1),
32:             Segment(victoria, greenPark, victoriaLine, 1),
33:             Segment(victoria, sloaneSquare, districtLine, 6),
34:             Segment(sloaneSquare, southKensington, districtLine, 3),
35:             Segment(southKensington, sloaneSquare, districtLine, 6),
36:             Segment(sloaneSquare, victoria, districtLine, 6)
37:         )
38:     )
39:
40:     val map = londonUnderground()
41:
42:     @Test
43:     fun 'can find multiple routes between stations'() {
44:
45:         val routes = map.routesFrom(southKensington, victoria)
46:         assertEquals(2, routes.size)
47:
48:         assertTrue(routes[0].segments.all { s -> s.line in setOf(piccadillyLine, victoriaLine) })
49:         assertTrue(routes[1].segments.all { s -> s.line == districtLine })
50:     }
51:
52:     @Test
53:     fun 'can optimise for number of changes'() {
54:
55:         val routes = map.routesFrom(southKensington, victoria, optimisingFor =
Route:numChanges)
56:         assertEquals(2, routes.size)
57:
58:         assertEquals(0, routes[0].numChanges())
59:         assertEquals(1, routes[1].numChanges())
60:     }
61:

```

```

../solution/src/test/kotlin/journeyplan/ExtensionsTest.kt    Fri Nov 19 21:26:20 2021    2
62:  @Test
63:  fun 'can optimise for duration'() {
64:
65:      val routes = map.routesFrom(southKensington, victoria, optimisingFor =
Route::duration)
66:      assertEquals(2, routes.size)
67:
68:      assertEquals(10, routes[0].duration())
69:      assertEquals(12, routes[1].duration())
70:  }
71:
72:  @Test
73:  fun 'does not offer routes with suspended lines'() {
74:
75:      var routes = map.routesFrom(southKensington, victoria)
76:
77:      assertEquals(2, routes.size)
78:      assertTrue(routes[0].segments.all { s -> s.line in setOf(piccadillyLine,
victoriaLine) })
79:      assertTrue(routes[1].segments.all { s -> s.line == districtLine })
80:
81:      districtLine.suspend()
82:
83:      routes = map.routesFrom(southKensington, victoria)
84:
85:      assertEquals(1, routes.size)
86:      assertTrue(routes[0].segments.none { s -> s.line == districtLine })
87:
88:      districtLine.resume()
89:
90:      routes = map.routesFrom(southKensington, victoria)
91:
92:      assertEquals(2, routes.size)
93:      assertTrue(routes[0].segments.all { s -> s.line in setOf(piccadillyLine,
victoriaLine) })
94:      assertTrue(routes[1].segments.all { s -> s.line == districtLine })
95:  }
96:
97:  @Test
98:  fun 'avoids interchange at closed stations'() {
99:
100:     var routes = map.routesFrom(southKensington, oxfordCircus)
101:     assertEquals(2, routes.size)
102:
103:     victoria.close()
104:
105:     routes = map.routesFrom(southKensington, oxfordCircus)
106:
107:     assertEquals(1, routes.size)
108:     assertDoesNotGoVia(victoria, routes[0])
109:  }
110:
111:  @Test
112:  fun 'does not avoid closed stations if interchange not required'() {
113:
114:     var routes = map.routesFrom(southKensington, oxfordCircus)
115:     assertEquals(2, routes.size)
116:
117:     sloaneSquare.close()
118:
119:     routes = map.routesFrom(southKensington, oxfordCircus)
120:     assertEquals(2, routes.size)
121:     println(routes)

```

```

../solution/src/test/kotlin/journeyplan/ExtensionsTest.kt    Fri Nov 19 21:26:20 2021    3
122:
123:     assertGoesVia(sloaneSquare, routes[1])
124: }
125:
126: fun assertGoesVia(station: Station, route: Route) {
127:     assertNotNull(findIn(route, station))
128: }
129:
130: fun assertDoesNotGoVia(station: Station, route: Route) {
131:     assertNull(findIn(route, station))
132: }
133:
134: fun findIn(route: Route, station: Station) = route.segments.find { s -> s.to ==
station }
135: }

```

../solution/src/test/kotlin/journeyplan/RoutePlannerTest.kt

Fri Nov 19 21:26:20 2021

1

```
1: package journeyplan
2:
3: import org.junit.Test
4: import kotlin.test.assertEquals
5:
6: class RoutePlannerTest {
7:
8:     val northernLine = Line("Northern")
9:     val victoriaLine = Line("Victoria")
10:    val centralLine = Line("Central")
11:
12:    val highgate = Station("Highgate")
13:    val archway = Station("Archway")
14:    val tufnellPark = Station("Tufnell Park")
15:    val kentishTown = Station("Kentish Town")
16:    val camden = Station("Camden Town")
17:    val euston = Station("Euston")
18:    val warrenStreet = Station("Warren Street")
19:    val oxfordCircus = Station("Oxford Circus")
20:    val bondStreet = Station("Bond Street")
21:
22:    val tufnellParkToHighgate =
23:        Route(
24:            listOf(
25:                Segment(tufnellPark, archway, northernLine, 3),
26:                Segment(archway, highgate, northernLine, 3)
27:            )
28:        )
29:
30:    val highgateToOxfordCircus =
31:        Route(
32:            listOf(
33:                Segment(highgate, archway, northernLine, 3),
34:                Segment(archway, kentishTown, northernLine, 3),
35:                Segment(kentishTown, camden, northernLine, 3),
36:                Segment(camden, euston, northernLine, 3),
37:                Segment(euston, warrenStreet, victoriaLine, 3),
38:                Segment(warrenStreet, oxfordCircus, victoriaLine, 3)
39:            )
40:        )
41:
42:    val camdenToBondStreet =
43:        Route(
44:            listOf(
45:                Segment(camden, euston, northernLine, 3),
46:                Segment(euston, warrenStreet, victoriaLine, 3),
47:                Segment(warrenStreet, oxfordCircus, victoriaLine, 3),
48:                Segment(oxfordCircus, bondStreet, centralLine, 2)
49:            )
50:        )
51:
52:    @Test
53:    fun 'can calculate number of changes'() {
54:        assertEquals(0, tufnellParkToHighgate.numChanges())
55:        assertEquals(1, highgateToOxfordCircus.numChanges())
56:        assertEquals(2, camdenToBondStreet.numChanges())
57:    }
58:
59:    @Test
60:    fun 'can calculate total duration'() {
61:        assertEquals(6, tufnellParkToHighgate.duration())
62:        assertEquals(18, highgateToOxfordCircus.duration())
63:        assertEquals(11, camdenToBondStreet.duration())
```

../solution/src/test/kotlin/journeyplan/RoutePlannerTest.kt

Fri Nov 19 21:26:20 2021

2

```
64:     }
65:
66:     @Test
67:     fun 'toString omits calling points'() {
68:         assertEquals(
69:             """
70:                 Tufnell Park to Highgate - 6 minutes, 0 changes
71:                 - Tufnell Park to Highgate by Northern Line
72:             """.trimIndent(),
73:             tufnellParkToHighgate.toString()
74:         )
75:     }
76:
77:     @Test
78:     fun 'toString shows changes'() {
79:         assertEquals(
80:             """
81:                 Highgate to Oxford Circus - 18 minutes, 1 changes
82:                 - Highgate to Euston by Northern Line
83:                 - Euston to Oxford Circus by Victoria Line
84:             """.trimIndent(),
85:             highgateToOxfordCircus.toString()
86:         )
87:     }
88: }
```

```

1: package journeyplan
2:
3: import org.junit.Test
4: import kotlin.test.assertEquals
5:
6: class TravelModelTest {
7:
8:     @Test
9:     fun 'printing stations shows their names'() {
10:         assertEquals("South Kensington", Station("South Kensington").toString())
11:         assertEquals("Knightsbridge", Station("Knightsbridge").toString())
12:     }
13:
14:     @Test
15:     fun 'printing lines shows their names'() {
16:         assertEquals("District Line", Line("District").toString())
17:         assertEquals("Circle Line", Line("Circle").toString())
18:     }
19: }

```