

```
1: []
```

```
1: package museumvisit
2:
3: fun createArtGallery(): Museum {
4:     val entrance = MuseumRoom("Entrance hall", 20)
5:     val result = Museum("Art gallery", entrance)
6:     val exhibitionRoom = MuseumRoom("Exhibition room", 10)
7:     result.addRoom(exhibitionRoom)
8:     result.connectRoomTo(entrance, exhibitionRoom)
9:     result.connectRoomToExit(exhibitionRoom)
10:    return result
11: }
12:
13: fun createAnimalSanctuary(): Museum {
14:     val entrance = MuseumRoom("Entrance hall", 20)
15:     val bats = MuseumRoom("Bats", 10)
16:     val lizards = MuseumRoom("Lizards", 10)
17:     val insects = MuseumRoom("Insects", 10)
18:     val giftShop = MuseumRoom("Gift shop", 10)
19:     val snakes = MuseumRoom("Snakes", 10)
20:     val result = Museum("Animal sanctuary", entrance)
21:     result.addRoom(bats)
22:     result.addRoom(lizards)
23:     result.addRoom(insects)
24:     result.addRoom(giftShop)
25:     result.addRoom(snakes)
26:     result.connectRoomTo(entrance, bats)
27:     result.connectRoomTo(bats, lizards)
28:     result.connectRoomTo(lizards, insects)
29:     result.connectRoomTo(insects, snakes)
30:     result.connectRoomTo(snakes, entrance)
31:     result.connectRoomTo(lizards, giftShop)
32:     result.connectRoomTo(insects, giftShop)
33:     result.connectRoomToExit(giftShop)
34:    return result
35: }
36:
37: fun createAnimalSanctuaryWithUnreachableRooms(): Museum {
38:     val entrance = MuseumRoom("Entrance hall", 20)
39:     val bats = MuseumRoom("Bats", 10)
40:     val lizards = MuseumRoom("Lizards", 10)
41:     val insects = MuseumRoom("Insects", 10)
42:     val giftShop = MuseumRoom("Gift shop", 10)
43:     val snakes = MuseumRoom("Snakes", 10)
44:     val result = Museum("Animal sanctuary", entrance)
45:     result.addRoom(bats)
46:     result.addRoom(lizards)
47:     result.addRoom(insects)
48:     result.addRoom(giftShop)
49:     result.addRoom(snakes)
50:     result.connectRoomTo(bats, lizards)
51:     result.connectRoomTo(lizards, insects)
52:     result.connectRoomTo(insects, snakes)
53:     result.connectRoomTo(snakes, entrance)
54:     result.connectRoomTo(lizards, giftShop)
55:     result.connectRoomTo(insects, giftShop)
56:     result.connectRoomToExit(giftShop)
57:    return result
58: }
59:
60: fun createAnimalSanctuaryWithRoomsThatDoNotLeadToExit(): Museum {
61:     val entrance = MuseumRoom("Entrance hall", 20)
62:     val bats = MuseumRoom("Bats", 10)
63:     val lizards = MuseumRoom("Lizards", 10)
64:     val insects = MuseumRoom("Insects", 10)
65:     val giftShop = MuseumRoom("Gift shop", 10)
66:     val snakes = MuseumRoom("Snakes", 10)
67:     val result = Museum("Animal sanctuary", entrance)
68:     result.addRoom(bats)
```

../solution/src/main/kotlin/museumvisit/ExampleMuseums.kt

Thu Feb 08 22:06:26 2024

2

```
69:     result.addRoom(lizards)
70:     result.addRoom(insects)
71:     result.addRoom(giftShop)
72:     result.addRoom(snakes)
73:     result.connectRoomTo(entrance, bats)
74:     result.connectRoomTo(bats, lizards)
75:     result.connectRoomTo(lizards, insects)
76:     result.connectRoomTo(insects, snakes)
77:     result.connectRoomTo(lizards, giftShop)
78:     result.connectRoomToExit(giftShop)
79:     return result
80: }
```

../solution/src/main/kotlin/museumvisit/Exceptions.kt

Thu Feb 08 22:06:26 2024

1

```
1: package museumvisit
2:
3: private fun getRoomNames(rooms: Set<MuseumRoom>): String =
4:     rooms.map { it.name }
5:         .sorted().joinToString()
6:
7: class UnreachableRoomsException(private val unreachable: Set<MuseumRoom>) :
Exception() {
8:     override fun toString(): String = "Unreachable rooms: " +
getRoomNames(unreachable)
9: }
10:
11: class CannotExitMuseumException(private val roomsThatDoNotLeadToExit:
Set<MuseumRoom>) : Exception() {
12:     override fun toString(): String = "Impossible to leave museum from: " +
getRoomNames(roomsThatDoNotLeadToExit)
13: }
```

```

1: package museumvisit
2:
3: fun exploreMuseum(museum: Museum) {
4:     println("Welcome to ${museum.name}! Let's explore.")
5:     var currentSite: MuseumSite = museum.enterIfPossible()!!
6:     currentSite.enter()
7:     while (currentSite is MuseumRoom) {
8:         println("You are in ${currentSite.name}")
9:         println("Have a good look around. Bored yet? Where do you want to go?")
10:        println("From here, you can go to:")
11:        val nextRoomNames = currentSite.exitTurnstiles.map { it ->
it.destination.name }
12:        for (name in nextRoomNames) {
13:            println(" $name")
14:        }
15:        val choice = readlnOrNull()
16:        if (choice == null) {
17:            println("You have had enough of this museum - taking you back to the
main menu.")
18:            return
19:        } else {
20:            if (choice in nextRoomNames) {
21:                currentSite = currentSite.exitTurnstiles.map { it.destination
}.filter { it.name == choice }[0]
22:            } else {
23:                println("I'm sorry, but that's not one of the next places you can
go. Let's try again.")
24:            }
25:        }
26:    }
27:    assert(currentSite == museum.outside)
28:    println("We hope you had a good time in the ${museum.name} museum - goodbye!!")
29: }
30:
31: val museumsToExplore = listOf(createArtGallery(), createAnimalSanctuary())
32:
33: fun main() {
34:     while (true) {
35:         println("Which museum would you like to explore?")
36:         println(" " + museumsToExplore.map { it.name }.joinToString())
37:         val choice = readlnOrNull()
38:         if (choice == null) {
39:             println("You have had enough of this game - what is wrong with you?
Goodbye.")
40:             return
41:         }
42:         val maybeMuseum = museumsToExplore.find { it.name == choice }
43:         if (maybeMuseum != null) {
44:             exploreMuseum(maybeMuseum)
45:         } else {
46:             println("I don't know that museum, sorry.")
47:         }
48:     }
49: }

```

```

1: package museumvisit
2:
3: import java.io.PrintStream
4:
5: class ImpatientVisitor(name: String, printStream: PrintStream, museum: Museum) :
Visitor(name, printStream, museum) {
6:     override fun enterMuseum(): MuseumRoom {
7:         while (true) {
8:             val maybeResult: MuseumRoom? = museum.enterIfPossible()
9:             if (maybeResult != null) {
10:                return maybeResult
11:            }
12:            printStream.println("$name could not get into ${museum.name} but will
try again soon.")
13:            waitSomeTimeBeforeRetrying()
14:            printStream.println("$name is ready to try again.")
15:        }
16:    }
17:
18:    override fun leaveRoom(currentRoom: MuseumRoom): MuseumSite {
19:        while (true) {
20:            val turnstile = currentRoom.exitTurnstiles.random()
21:            val maybeNextSite = turnstile.passToNextSiteIfPossible()
22:            if (maybeNextSite != null) {
23:                return maybeNextSite
24:            }
25:            printStream.println("$name failed to leave ${currentRoom.name} but will
try again soon.")
26:            waitSomeTimeBeforeRetrying()
27:            printStream.println("$name is ready to try leaving ${currentRoom.name}
again.")
28:        }
29:    }
30:
31:    private fun waitSomeTimeBeforeRetrying() {
32:        Thread.sleep(10)
33:    }
34: }

```

```

1: package museumvisit
2:
3: fun main() {
4:     val visitorNames = listOf(
5:         "Neha",
6:         "Alex",
7:         "Yi",
8:         "Jianyi",
9:         "Felix",
10:        "Oscar",
11:        "Amelia",
12:        "Noah",
13:        "Prakesh",
14:        "Satnam",
15:        "Susan",
16:        "Poppy",
17:        "Jaya",
18:        "Indy",
19:        "Lula",
20:        "Maximilian",
21:        "Minimilian",
22:        "Jacub",
23:        "Donald",
24:        "Liz",
25:        "Teresa",
26:        "Julia",
27:        "Parminda",
28:        "Xi",
29:    )
30:    val museum = createAnimalSanctuary()
31:    val visitors = visitorNames.mapIndexed { index, person ->
32:        Thread(ImpatientVisitor(person, System.out, museum))
33:    }
34:    visitors.forEach { it.start() }
35:    visitors.forEach { it.join() }
36: }

```

```

1: package museumvisit
2:
3: import kotlin.IllegalArgumentException
4: import kotlin.concurrent.withLock
5:
6: class Museum(val name: String, private val entrance: MuseumRoom) {
7:
8:     var admitted: Int = 0
9:     private set
10:
11:     val outside: OutsideMuseum = OutsideMuseum()
12:
13:     private val rooms: MutableSet<MuseumRoom> = mutableSetOf(entrance)
14:
15:     fun addRoom(room: MuseumRoom) {
16:         if (room in rooms) {
17:             throw IllegalArgumentException("A room should not be added to the
museum multiple times.")
18:         }
19:         rooms.add(room)
20:     }
21:
22:     fun connectRoomTo(startRoom: MuseumRoom, destinationRoom: MuseumRoom) {
23:         if (startRoom === destinationRoom) {
24:             throw IllegalArgumentException("Cannot connect a room to itself.")
25:         }
26:         if (startRoom !in rooms || destinationRoom !in rooms) {
27:             throw IllegalArgumentException("To add a connection between rooms, they
must both be rooms of the museum.")
28:         }
29:         if (destinationRoom in startRoom.exitTurnstiles.map { it.destination }) {
30:             throw IllegalArgumentException("A turnstile from ${startRoom.name} to
${destinationRoom.name} already exists.")
31:         }
32:         startRoom.addExitTurnstile(destinationRoom)
33:     }
34:
35:     fun connectRoomToExit(room: MuseumRoom) {
36:         if (room !in rooms) {
37:             throw IllegalArgumentException("To connect a room to the exit, the room
must be part of the museum.")
38:         }
39:         if (outside in room.exitTurnstiles.map { it.destination }) {
40:             throw IllegalArgumentException("An exit turnstile from ${room.name} to
already exists.")
41:         }
42:         room.addExitTurnstile(outside)
43:     }
44:
45:     fun enterIfPossible(): MuseumRoom? {
46:         entrance.lock.withLock {
47:             if (entrance.hasCapacity()) {
48:                 entrance.enter()
49:                 admitted++
50:                 return entrance
51:             }
52:             return null
53:         }
54:     }
55:
56:     fun entranceHasCapacity(): Boolean = entrance.hasCapacity()
57:
58:     fun enter() {
59:         if (!entrance.hasCapacity()) {
60:             throw UnsupportedOperationException("The museum entrance is full.")
61:         }
62:         admitted++
63:         entrance.enter()

```

```

64:     }
65:
66:     // This is part of the extension
67:     fun enterByWaiting(): MuseumRoom {
68:         while (true) {
69:             entrance.lock.withLock {
70:                 while (!entranceHasCapacity()) {
71:                     entrance.hasCapacityCondition.await()
72:                 }
73:                 enter()
74:                 return entrance
75:             }
76:         }
77:     }
78:
79:     fun checkWellFormed() {
80:         val reachableFrom = mutableMapOf<MuseumRoom, MutableSet<MuseumSite>>()
81:         for (room in rooms) {
82:             reachableFrom[room] = mutableSetOf(room)
83:         }
84:         var changed = true
85:         while (changed) {
86:             changed = false
87:             for (room in rooms) {
88:                 for (turnstile in room.exitTurnstiles) {
89:                     changed = if (turnstile.destination is OutsideMuseum) {
90:                         changed or reachableFrom[room]!!.add(turnstile.destination)
91:                     } else {
92:                         changed or
reachableFrom[room]!!.addAll(reachableFrom[turnstile.destination as MuseumRoom]!!)
93:                     }
94:                 }
95:             }
96:         }
97:
98:         val unreachableRooms: Set<MuseumRoom> = rooms -
reachableFrom[entrance]!!.filterIsInstance<MuseumRoom>()
99:         if (unreachableRooms.isNotEmpty()) {
100:             throw UnreachableRoomsException(unreachableRooms)
101:         }
102:         val roomsThatDoNotReachExit: Set<MuseumRoom> =
103:             rooms.filter { outside !in reachableFrom[it]!! }.toSet()
104:         if (roomsThatDoNotReachExit.isNotEmpty()) {
105:             throw CannotExitMuseumException(roomsThatDoNotReachExit)
106:         }
107:     }
108:
109:     override fun toString(): String {
110:         val result = StringBuilder()
111:         result.append("$name\n")
112:         val alreadyConsidered = mutableSetOf(entrance)
113:         val roomsToProcess = mutableListOf(entrance)
114:         while (roomsToProcess.isNotEmpty()) {
115:             val room = roomsToProcess.removeAt(0)
116:             result.append("${room.name} leads to: ")
117:             result.append(room.exitTurnstiles.map { it.destination.name
}.joinToString())
118:             result.append("\n")
119:             for (nextRoom in room.exitTurnstiles.map { it.destination
}.filterIsInstance<MuseumRoom>()) {
120:                 if (nextRoom !in alreadyConsidered) {
121:                     alreadyConsidered.add(nextRoom)
122:                     roomsToProcess.add(nextRoom)
123:                 }
124:             }
125:         }
126:         return result.toString()
127:     }

```

128: }

```
../solution/src/main/kotlin/museumvisit/MuseumRoom.kt      Thu Feb 08 22:06:26 2024      1
1: package museumvisit
2:
3: class MuseumRoom(override val name: String, private val capacity: Int) :
MuseumSite() {
4:
5:     init {
6:         if (capacity < 1) {
7:             throw IllegalArgumentException("A room must capacity for at least one
visitor.")
8:         }
9:     }
10:
11:     val exitTurnstiles: List<Turnstile>
12:         get() = turnstileList
13:
14:     private val turnstileList: MutableList<Turnstile> = mutableListOf()
15:
16:     override fun hasCapacity(): Boolean = occupancy < capacity
17:
18:     override fun enter() {
19:         if (!hasCapacity()) {
20:             throw UnsupportedOperationException("Room should never exceed its
capacity.")
21:         }
22:         occupancy++
23:     }
24:
25:     fun exit() {
26:         if (occupancy == 0) {
27:             throw UnsupportedOperationException("Room should never go below
capacity 0.")
28:         }
29:         occupancy--
30:     }
31:
32:     fun addExitTurnstile(destination: MuseumSite) {
33:         turnstileList.add(Turnstile(this, destination))
34:     }
35: }
```

```
../solution/src/main/kotlin/museumvisit/MuseumSite.kt      Thu Feb 08 22:06:26 2024      1
1: package museumvisit
2:
3: import java.util.concurrent.locks.Condition
4: import java.util.concurrent.locks.Lock
5: import java.util.concurrent.locks.ReentrantLock
6:
7: abstract class MuseumSite {
8:     val lock: Lock = ReentrantLock()
9:     val hasCapacityCondition: Condition = lock.newCondition()
10:
11:     abstract val name: String
12:     var occupancy: Int = 0
13:     protected set
14:
15:     abstract fun hasCapacity(): Boolean
16:
17:     abstract fun enter()
18: }
```

../solution/src/main/kotlin/museumvisit/OutsideMuseum.kt Thu Feb 08 22:06:26 2024 1

```
1: package museumvisit
2:
3: class OutsideMuseum : MuseumSite() {
4:
5:     override val name: String = "Outside"
6:
7:     override fun hasCapacity(): Boolean = true
8:
9:     override fun enter() {
10:         occupancy++
11:     }
12: }
```

../solution/src/main/kotlin/museumvisit/PatientVisitor.kt Thu Feb 08 22:06:26 2024 1

```
1: package museumvisit
2:
3: import java.io.PrintStream
4:
5: class PatientVisitor(name: String, printStream: PrintStream, museum: Museum) :
Visitor(name, printStream, museum) {
6:     override fun enterMuseum(): MuseumRoom = museum.enterByWaiting()
7:
8:     override fun leaveRoom(currentRoom: MuseumRoom): MuseumSite {
9:         while (true) {
10:             val turnstile = currentRoom.exitTurnstiles.random()
11:             val maybeNextSite =
turnstile.passToNextSiteWithPatienceAndPersistence(5, 100)
12:             if (maybeNextSite != null) {
13:                 return maybeNextSite
14:             }
15:         }
16:     }
17: }
```

```

../solution/src/main/kotlin/museumvisit/Turnstile.kt      Thu Feb 08 22:06:26 2024      1
1: package museumvisit
2:
3: import java.util.concurrent.TimeUnit
4: import kotlin.concurrent.withLock
5:
6: class Turnstile(val origin: MuseumRoom, val destination: MuseumSite) {
7:
8:     fun passToNextSiteIfPossible(): MuseumSite? {
9:         // The following condition establishes a total order relation over the
MuseumSites: assuming the names of the
10:         // rooms are unique, it is possible with the following condition to
establish an order such that rooms with
11:         // larger names (in alphabetical order) are always locked first. Any
ordering can be used (including comparing
12:         // the hashcodes of the MuseumSite instances) as long as it is total, i.e.,
can order any possible pair of
13:         // MuseumSites. Establishing an order is critical to avoid circular
dependencies (i.e., deadlocks).
14:         val (firstLock, secondLock) = if (destination.name.compareTo(origin.name)
>= 0) {
15:             Pair(destination.lock, origin.lock)
16:         } else {
17:             Pair(origin.lock, destination.lock)
18:         }
19:         firstLock.withLock {
20:             secondLock.withLock {
21:                 if (destination.hasCapacity()) {
22:                     origin.exit()
23:                     destination.enter()
24:                     origin.hasCapacityCondition.signalAll()
25:                     return destination
26:                 }
27:                 return null
28:             }
29:         }
30:     }
31:
32:     // This is part of the extension.
33:     fun passToNextSiteWithPatienceAndPersistence(
34:         attempts: Int,
35:         timeoutPerAttempt: Long,
36:     ): MuseumSite? {
37:         for (attempt in 1..attempts) {
38:             val maybeNextSite = passToNextSiteIfPossible()
39:             if (maybeNextSite != null) {
40:                 return maybeNextSite
41:             }
42:             destination.lock.withLock {
43:                 if (!destination.hasCapacity()) {
44:                     destination.hasCapacityCondition.await(timeoutPerAttempt,
TimeUnit.MILLISECONDS)
45:                 }
46:             }
47:         }
48:         return null
49:     }
50: }

```

```

../solution/src/main/kotlin/museumvisit/Visitor.kt      Thu Feb 08 22:06:26 2024      1
1: package museumvisit
2:
3: import java.io.PrintStream
4:
5: abstract class Visitor(protected val name: String, protected val printStream:
PrintStream, protected val museum: Museum) : Runnable {
6:
7:     abstract fun enterMuseum(): MuseumRoom
8:
9:     abstract fun leaveRoom(currentRoom: MuseumRoom): MuseumSite
10:
11:     override fun run() {
12:         var currentSite: MuseumSite = enterMuseum()
13:         printStream.println("$name has entered ${museum.name}.")
14:         while (currentSite != museum.outside) {
15:             assert(currentSite is MuseumRoom)
16:             printStream.println("$this has entered ${currentSite.name}.")
17:             val randomVisitTimeInMillis = (Math.random() * 200).toInt() + 1
18:             Thread.sleep(randomVisitTimeInMillis.toLong())
19:             printStream.println("$this wants to leave ${currentSite.name}.")
20:             val newSite = leaveRoom(currentSite as MuseumRoom)
21:             printStream.println("$this has left ${currentSite.name}.")
22:             currentSite = newSite
23:         }
24:         printStream.println("$this has left ${museum.name}.")
25:     }
26:
27:     override fun toString(): String {
28:         return name
29:     }
30: }

```



```

../solution/src/test/kotlin/museumvisit/ExtensionTest.kt    Thu Feb 08 22:06:26 2024    1
1: package museumvisit
2:
3: import org.junit.Test
4: import java.io.ByteArrayOutputStream
5: import java.io.PrintStream
6: import kotlin.test.assertEquals
7:
8: class ExtensionTest {
9:
10:     private fun testMuseumVisit(museumUnderTest: MuseumUnderTest, people:
List<String>, allPatient: Boolean) {
11:         val byteOutputStreams = people.map {
12:             ByteArrayOutputStream()
13:         }
14:         val printStreams = byteOutputStreams.map {
15:             PrintStream(it)
16:         }
17:         assertEquals(0, museumUnderTest.museum.admitted)
18:         assertEquals(0, museumUnderTest.museum.outside.occupancy)
19:         for (room in museumUnderTest.rooms) {
20:             assertEquals(0, room.occupancy)
21:         }
22:         val visitors = people.mapIndexed { index, person ->
23:             Thread(
24:                 if (allPatient || (index % 2 == 0)) {
25:                     PatientVisitor(person, printStreams[index],
museumUnderTest.museum)
26:                 } else {
27:                     ImpatientVisitor(person, printStreams[index],
museumUnderTest.museum)
28:                 },
29:             )
30:         }
31:         visitors.forEach { it.start() }
32:         visitors.forEach { it.join() }
33:         assertEquals(people.size, museumUnderTest.museum.admitted)
34:         assertEquals(people.size, museumUnderTest.museum.outside.occupancy)
35:         for (room in museumUnderTest.rooms) {
36:             assertEquals(0, room.occupancy)
37:         }
38:         byteOutputStreams.forEachIndexed { index, byteArrayOutputStream ->
39:             if (allPatient || (index % 2 == 0)) {
40:                 checkPatientOutput(people[index], byteArrayOutputStream.toString(),
museumUnderTest)
41:             } else {
42:                 checkImpatientOutput(people[index],
byteArrayOutputStream.toString(), museumUnderTest)
43:             }
44:         }
45:     }
46:
47:     @Test
48:     fun 'two patient visitors to small museum'() {
49:         testMuseumVisit(createSmallMuseumUnderTest(), listOf("Ally", "Chris"), true)
50:     }
51:
52:     @Test
53:     fun 'one patient and one impatient visitor to small museum'() {
54:         testMuseumVisit(createSmallMuseumUnderTest(), listOf("Ally", "Chris"),
false)
55:     }
56:
57:     @Test
58:     fun 'many patient visitors to small museum'() {
59:         testMuseumVisit(createSmallMuseumUnderTest(), listOfPeople, true)
60:     }
61:
62:     @Test

```

```

../solution/src/test/kotlin/museumvisit/ExtensionTest.kt    Thu Feb 08 22:06:26 2024    2
63:     fun 'many mixed-patience visitors to small museum'() {
64:         testMuseumVisit(createSmallMuseumUnderTest(), listOfPeople, false)
65:     }
66:
67:     @Test
68:     fun 'two patient visitors to aquarium'() {
69:         testMuseumVisit(createAquariumMuseumUnderTest(), listOf("Ally", "Chris"),
true)
70:     }
71:
72:     @Test
73:     fun 'one patient and one impatient visitor to aquarium'() {
74:         testMuseumVisit(createAquariumMuseumUnderTest(), listOf("Ally", "Chris"),
false)
75:     }
76:
77:     @Test
78:     fun 'many patient visitors to aquarium'() {
79:         testMuseumVisit(createAquariumMuseumUnderTest(), listOfPeople, true)
80:     }
81:
82:     @Test
83:     fun 'many mixed-patience visitors to aquarium'() {
84:         testMuseumVisit(createAquariumMuseumUnderTest(), listOfPeople, false)
85:     }
86: }

```

```

../solution/src/test/kotlin/museumvisit/ImpatientVisitorsTest.kt    Thu Feb 08 22:06:26 2024    1
1: package museumvisit
2:
3: import org.junit.Test
4: import java.io.ByteArrayOutputStream
5: import java.io.PrintStream
6: import kotlin.test.assertEquals
7:
8: class ImpatientVisitorsTest {
9:
10:     private fun testMuseumVisit(museumUnderTest: MuseumUnderTest, people:
List<String>) {
11:         val byteOutputStreams = people.map {
12:             ByteArrayOutputStream()
13:         }
14:         val printStreams = byteOutputStreams.map {
15:             PrintStream(it)
16:         }
17:         assertEquals(0, museumUnderTest.museum.admitted)
18:         assertEquals(0, museumUnderTest.museum.outside.occupancy)
19:         for (room in museumUnderTest.rooms) {
20:             assertEquals(0, room.occupancy)
21:         }
22:         val visitors = people.mapIndexed { index, person ->
23:             Thread(ImpatientVisitor(person, printStreams[index],
museumUnderTest.museum))
24:         }
25:         visitors.forEach { it.start() }
26:         visitors.forEach { it.join() }
27:         assertEquals(people.size, museumUnderTest.museum.admitted)
28:         assertEquals(people.size, museumUnderTest.museum.outside.occupancy)
29:         for (room in museumUnderTest.rooms) {
30:             assertEquals(0, room.occupancy)
31:         }
32:         byteOutputStreams.forEachIndexed { index, byteArrayOutputStream ->
33:             checkImpatientOutput(people[index], byteArrayOutputStream.toString(),
museumUnderTest)
34:         }
35:     }
36:
37:     @Test
38:     fun 'two visitors to small museum'() {
39:         testMuseumVisit(createSmallMuseumUnderTest(), listOf("Ally", "Chris"))
40:     }
41:
42:     @Test
43:     fun 'many visitors to small museum'() {
44:         testMuseumVisit(createSmallMuseumUnderTest(), lotsOfPeople)
45:     }
46:
47:     @Test
48:     fun 'two visitors to aquarium'() {
49:         testMuseumVisit(createAquariumMuseumUnderTest(), listOf("Ally", "Chris"))
50:     }
51:
52:     @Test
53:     fun 'many visitors to aquarium'() {
54:         testMuseumVisit(createAquariumMuseumUnderTest(), lotsOfPeople)
55:     }
56: }

```

```

../solution/src/test/kotlin/museumvisit/MuseumRoomTest.kt    Thu Feb 08 22:06:26 2024    1
1: package museumvisit
2:
3: import kotlin.test.Test
4: import kotlin.test.assertEquals
5: import kotlin.test.assertFalse
6: import kotlin.test.assertTrue
7: import kotlin.test.fail
8:
9: class MuseumRoomTest {
10:
11:     @Test
12:     fun 'test name'() {
13:         assertEquals("Dali paintings", MuseumRoom("Dali paintings", 10).name)
14:     }
15:
16:     @Test
17:     fun 'test capacity positive'() {
18:         try {
19:             MuseumRoom("Dali paintings", 0)
20:             fail("An IllegalArgumentException should have been thrown.")
21:         } catch (exception: IllegalArgumentException) {
22:             // Good: exception expected
23:         }
24:         try {
25:             MuseumRoom("Dali paintings", -100)
26:             fail("An IllegalArgumentException should have been thrown.")
27:         } catch (exception: IllegalArgumentException) {
28:             // Good: exception expected
29:         }
30:     }
31:
32:     @Test
33:     fun 'successful enter and exit'() {
34:         val room = MuseumRoom("Sharks", 5)
35:         assertTrue(room.hasCapacity())
36:         room.enter()
37:         assertTrue(room.hasCapacity())
38:         room.enter()
39:         assertTrue(room.hasCapacity())
40:         room.enter()
41:         assertTrue(room.hasCapacity())
42:         room.enter()
43:         assertTrue(room.hasCapacity())
44:         room.enter()
45:         assertFalse(room.hasCapacity())
46:         room.exit()
47:         assertTrue(room.hasCapacity())
48:         room.exit()
49:         assertTrue(room.hasCapacity())
50:         room.exit()
51:         assertTrue(room.hasCapacity())
52:         room.exit()
53:         assertTrue(room.hasCapacity())
54:         room.exit()
55:         assertTrue(room.hasCapacity())
56:     }
57:
58:     @Test
59:     fun 'exception on enter if room gets full'() {
60:         val room = MuseumRoom("Sharks", 3)
61:         assertTrue(room.hasCapacity())
62:         room.enter()
63:         assertTrue(room.hasCapacity())
64:         room.enter()
65:         assertTrue(room.hasCapacity())
66:         room.enter()
67:         assertFalse(room.hasCapacity())
68:         try {

```

```

../solution/src/test/kotlin/museumvisit/MuseumRoomTest.kt      Thu Feb 08 22:06:26 2024      2
69:         room.enter()
70:         fail("An UnsupportedOperationException should have been thrown.")
71:     } catch (exception: UnsupportedOperationException) {
72:         // Good: exception expected
73:     }
74: }
75:
76: @Test
77: fun 'exception on exit if room is empty'() {
78:     val room = MuseumRoom("Sharks", 3)
79:     assertTrue(room.hasCapacity())
80:     try {
81:         room.exit()
82:         fail("An UnsupportedOperationException should have been thrown.")
83:     } catch (exception: UnsupportedOperationException) {
84:         // Good: exception expected
85:     }
86: }
87: }

```

```

../solution/src/test/kotlin/museumvisit/MuseumTest.kt          Thu Feb 08 22:06:26 2024      1
1: package museumvisit
2:
3: import org.junit.Test
4: import kotlin.test.assertEquals
5: import kotlin.test.assertFalse
6: import kotlin.test.assertTrue
7: import kotlin.test.fail
8:
9: class MuseumTest {
10:
11:     @Test
12:     fun 'test toString art gallery'() {
13:         assertEquals(
14:             """
15:             Art gallery
16:             Entrance hall leads to: Exhibition room
17:             Exhibition room leads to: Outside
18:
19:             """.trimIndent(),
20:             createArtGallery().toString(),
21:         )
22:     }
23:
24:     @Test
25:     fun 'test toString animal sanctuary'() {
26:         assertEquals(
27:             """
28:             Animal sanctuary
29:             Entrance hall leads to: Bats
30:             Bats leads to: Lizards
31:             Lizards leads to: Insects, Gift shop
32:             Insects leads to: Snakes, Gift shop
33:             Gift shop leads to: Outside
34:             Snakes leads to: Entrance hall
35:
36:             """.trimIndent(),
37:             createAnimalSanctuary().toString(),
38:         )
39:     }
40:
41:     @Test
42:     fun 'test well formed art gallery'() {
43:         createArtGallery().checkWellFormed()
44:     }
45:
46:     @Test
47:     fun 'test well formed animal sanctuary'() {
48:         createAnimalSanctuary().checkWellFormed()
49:     }
50:
51:     @Test
52:     fun 'test animal sanctuary with unreachable rooms'() {
53:         try {
54:             createAnimalSanctuaryWithUnreachableRooms().checkWellFormed()
55:             fail("An UnreachableRoomException should have been thrown")
56:         } catch (exception: UnreachableRoomsException) {
57:             assertEquals(
58:                 """
59:                 Unreachable rooms: Bats, Gift shop, Insects, Lizards, Snakes
60:                 """.trimIndent(),
61:                 exception.toString(),
62:             )
63:         }
64:     }
65:
66:     @Test
67:     fun 'test animal sanctuary with rooms that do not lead to exit'() {
68:         try {

```

```

./solution/src/test/kotlin/museumvisit/MuseumTest.kt      Thu Feb 08 22:06:26 2024      2
69:         createAnimalSanctuaryWithRoomsThatDoNotLeadToExit().checkWellFormed()
70:         fail("An CannotExitMuseumException should have been thrown")
71:     } catch (exception: CannotExitMuseumException) {
72:         assertEquals(
73:             """
74:             Impossible to leave museum from: Insects, Snakes
75:             """.trimIndent(),
76:             exception.toString(),
77:         )
78:     }
79: }
80:
81: @Test
82: fun 'cannot connect unknown room to exit'() {
83:     val museum = Museum("Some museum", MuseumRoom("Entrance", 5))
84:     try {
85:         museum.connectRoomToExit(MuseumRoom("Some room", 3))
86:         fail("Expected IllegalArgumentException to be thrown")
87:     } catch (exception: IllegalArgumentException) {
88:         // Good: exception expected
89:     }
90: }
91:
92: @Test
93: fun 'cannot connect rooms if first is unknown'() {
94:     val entrance = MuseumRoom("Entrance", 5)
95:     val museum = Museum("Some museum", entrance)
96:     try {
97:         museum.connectRoomTo(MuseumRoom("Some room", 3), entrance)
98:         fail("Expected IllegalArgumentException to be thrown")
99:     } catch (exception: IllegalArgumentException) {
100:         // Good: exception expected
101:     }
102: }
103:
104: @Test
105: fun 'cannot connect rooms if second is unknown'() {
106:     val entrance = MuseumRoom("Entrance", 5)
107:     val museum = Museum("Some museum", entrance)
108:     try {
109:         museum.connectRoomTo(entrance, MuseumRoom("Some room", 3))
110:         fail("Expected IllegalArgumentException to be thrown")
111:     } catch (exception: IllegalArgumentException) {
112:         // Good: exception expected
113:     }
114: }
115:
116: @Test
117: fun 'cannot add room multiple times'() {
118:     val entrance = MuseumRoom("Entrance", 5)
119:     val museum = Museum("Some museum", entrance)
120:     try {
121:         museum.addRoom(entrance)
122:         fail("Expected IllegalArgumentException to be thrown")
123:     } catch (exception: IllegalArgumentException) {
124:         // Good: exception expected
125:     }
126: }
127:
128: fun 'cannot add room with same name'() {
129:     val entrance = MuseumRoom("Entrance", 5)
130:     val museum = Museum("Some museum", entrance)
131:     try {
132:         museum.addRoom(MuseumRoom("Entrance", 6))
133:         fail("Expected IllegalArgumentException to be thrown")
134:     } catch (exception: IllegalArgumentException) {
135:         // Good: exception expected
136:     }

```

```

./solution/src/test/kotlin/museumvisit/MuseumTest.kt      Thu Feb 08 22:06:26 2024      3
137:     }
138: }
139: @Test
140: fun 'cannot connect room to same room multiple times'() {
141:     val entrance = MuseumRoom("Entrance", 5)
142:     val exhibitionRoom = MuseumRoom("Exhibition room", 3)
143:     val museum = Museum("Some museum", entrance)
144:     museum.addRoom(exhibitionRoom)
145:     museum.connectRoomTo(entrance, exhibitionRoom)
146:     try {
147:         museum.connectRoomTo(entrance, exhibitionRoom)
148:         fail("Expected IllegalArgumentException to be thrown")
149:     } catch (exception: IllegalArgumentException) {
150:         // Good: exception expected
151:     }
152: }
153:
154: @Test
155: fun 'cannot connect room to exit multiple times'() {
156:     val entrance = MuseumRoom("Entrance", 5)
157:     val museum = Museum("Some museum", entrance)
158:     museum.connectRoomToExit(entrance)
159:     try {
160:         museum.connectRoomToExit(entrance)
161:         fail("Expected IllegalArgumentException to be thrown")
162:     } catch (exception: IllegalArgumentException) {
163:         // Good: exception expected
164:     }
165: }
166:
167: @Test
168: fun 'cannot connect room to self'() {
169:     val entrance = MuseumRoom("Entrance", 5)
170:     val museum = Museum("Some museum", entrance)
171:     try {
172:         museum.connectRoomTo(entrance, entrance)
173:         fail("Expected IllegalArgumentException to be thrown")
174:     } catch (exception: IllegalArgumentException) {
175:         // Good: exception expected
176:     }
177: }
178:
179: @Test
180: fun 'test museum has capacity'() {
181:     val museum = createArtGallery()
182:     for (i in 0..<20) {
183:         assertTrue(museum.entranceHasCapacity())
184:         assertEquals(i, museum.admitted)
185:         museum.enter()
186:     }
187:     assertFalse(museum.entranceHasCapacity())
188:     try {
189:         museum.enter()
190:         fail("An UnsupportedOperationException should have been thrown.")
191:     } catch (exception: UnsupportedOperationException) {
192:         // Good: exception expected
193:     }
194: }
195: }

```

```
../solution/src/test/kotlin/museumvisit/TestHelpers.kt
```

Thu Feb 08 22:06:26 2024

1

```

1: package museumvisit
2:
3: import kotlin.test.assertEquals
4: import kotlin.test.assertTrue
5: import kotlin.test.fail
6:
7: class MuseumUnderTest(val museum: Museum, val entrance: MuseumRoom, val rooms:
List<MuseumRoom>){
8:
9: fun createSmallMuseumUnderTest(): MuseumUnderTest {
10:     val entrance = MuseumRoom("Entrance", 1)
11:     val exhibition1 = MuseumRoom("Room 1", 1)
12:     val exhibition2 = MuseumRoom("Room 1", 1)
13:
14:     val rooms = listOf(entrance, exhibition1, exhibition2)
15:
16:     val museum = Museum("Small museum", entrance)
17:     museum.addRoom(exhibition1)
18:     museum.addRoom(exhibition2)
19:     museum.connectRoomTo(entrance, exhibition1)
20:     museum.connectRoomTo(exhibition1, entrance)
21:     museum.connectRoomTo(exhibition1, exhibition2)
22:     museum.connectRoomTo(exhibition2, exhibition1)
23:     museum.connectRoomToExit(exhibition2)
24:     museum.checkWellFormed()
25:
26:     return MuseumUnderTest(museum, entrance, rooms)
27: }
28:
29: fun createAquariumMuseumUnderTest(): MuseumUnderTest {
30:     val entrance = MuseumRoom("Aquarium entrance", 20)
31:     val crustaceans = MuseumRoom("Crabs and lobsters", 4)
32:     val sharks = MuseumRoom("Sharks", 4)
33:     val rays = MuseumRoom("Rays", 6)
34:     val seahorses = MuseumRoom("Seahorses", 3)
35:     val smallFish = MuseumRoom("Small fish", 9)
36:     val bobbits = MuseumRoom("Bobbitt worms", 1)
37:
38:     val rooms = listOf(crustaceans, sharks, rays, seahorses, smallFish, bobbits,
entrance)
39:
40:     val museum = Museum("Ally's Grand Aquarium", entrance)
41:     museum.addRoom(crustaceans)
42:     museum.addRoom(sharks)
43:     museum.addRoom(rays)
44:     museum.addRoom(seahorses)
45:     museum.addRoom(smallFish)
46:     museum.addRoom(bobbits)
47:     museum.connectRoomTo(entrance, crustaceans)
48:     museum.connectRoomTo(crustaceans, sharks)
49:     museum.connectRoomTo(sharks, rays)
50:     museum.connectRoomTo(rays, seahorses)
51:     museum.connectRoomTo(seahorses, smallFish)
52:     museum.connectRoomTo(smallFish, bobbits)
53:     museum.connectRoomTo(bobbits, entrance)
54:     museum.connectRoomTo(sharks, smallFish)
55:     museum.connectRoomTo(smallFish, sharks)
56:     museum.connectRoomToExit(entrance)
57:     museum.connectRoomToExit(rays)
58:     museum.checkWellFormed()
59:
60:     return MuseumUnderTest(museum, entrance, rooms)
61: }
62:
63: val lotsOfPeople = listOf(
64:     "Neha",
65:     "Alex",
66:     "Yi",

```

```
../solution/src/test/kotlin/museumvisit/TestHelpers.kt
```

Thu Feb 08 22:06:26 2024

2

```

67:     "Jianyi",
68:     "Felix",
69:     "Oscar",
70:     "Amelia",
71:     "Noah",
72:     "Prakesh",
73:     "Satnam",
74:     "Susan",
75:     "Poppy",
76:     "Jaya",
77:     "Indy",
78:     "Lula",
79:     "Maximilian",
80:     "Minimilian",
81:     "Jacob",
82:     "Donald",
83:     "Liz",
84:     "Teresa",
85:     "Julia",
86:     "Parminda",
87:     "Xi",
88: )
89:
90: fun checkImpatientOutput(person: String, output: String, museumUnderTest:
MuseumUnderTest) {
91:     val lines = output.split("\n")
92:     var index = 0
93:     while (lines[index] != "$person has entered ${museumUnderTest.museum.name}.") {
94:         assertEquals("$person could not get into ${museumUnderTest.museum.name} but
will try again soon.", lines[index])
95:         index++
96:         assertEquals("$person is ready to try again.", lines[index])
97:         index++
98:     }
99:     index++
100:    while (index < lines.size) {
101:        val personEnteredRegex = "" "$person has entered ([a-zA-Z0-9
]+)\\. "" .toRegex()
102:        val personEnteredRegexMatchResult = personEnteredRegex.find(lines[index])!!
103:        val (roomName) = personEnteredRegexMatchResult.destructured
104:        assertTrue(roomName in museumUnderTest.rooms.map { it.name }, "Unknown room
name $roomName")
105:        assertTrue(index < lines.size - 1)
106:        index++
107:        assertEquals("$person wants to leave $roomName.", lines[index])
108:        assertTrue(index < lines.size - 1)
109:        index++
110:        while (" $person has left $roomName." != lines[index]) {
111:            assertEquals("$person failed to leave $roomName but will try again
soon.", lines[index])
112:            assertTrue(index < lines.size - 1)
113:            index++
114:            assertEquals("$person is ready to try leaving $roomName again.",
lines[index])
115:            assertTrue(index < lines.size - 1)
116:            index++
117:        }
118:        assertTrue(index < lines.size - 1)
119:        index++
120:        if (lines[index] == "$person has left ${museumUnderTest.museum.name}.") {
121:            assertEquals(lines.size - 2, index)
122:            assertEquals("", lines[lines.size - 1])
123:            return
124:        }
125:    }
126:    fail("Expected to see $person leaving the museum.")
127: }
128:

```

```

129: fun checkPatientOutput(person: String, output: String, museumUnderTest:
MuseumUnderTest) {
130:     val lines = output.split("\n")
131:     var index = 0
132:     assertEquals("$person has entered ${museumUnderTest.museum.name}.",
lines[index])
133:     assertTrue(index < lines.size - 1)
134:     index++
135:     while (index < lines.size) {
136:         val personEnteredRegex = """"$person has entered ([a-zA-Z0-9
]+)\. """".toRegex()
137:         val personEnteredRegexMatchResult = personEnteredRegex.find(lines[index])!!
138:         val (roomName) = personEnteredRegexMatchResult.destructured
139:         assertTrue(roomName in museumUnderTest.rooms.map { it.name }, "Unknown room
name $roomName")
140:         assertTrue(index < lines.size - 1)
141:         index++
142:         assertEquals("$person wants to leave $roomName.", lines[index])
143:         assertTrue(index < lines.size - 1)
144:         index++
145:         assertEquals("$person has left $roomName.", lines[index])
146:         assertTrue(index < lines.size - 1)
147:         index++
148:         if (lines[index] == "$person has left ${museumUnderTest.museum.name}.") {
149:             assertEquals(lines.size - 2, index)
150:             assertEquals("", lines[lines.size - 1])
151:             return
152:         }
153:     }
154:     fail("Expected to see $person leaving the museum.")
155: }

```