# Assignment 1

Contents

[Assignment 1 1](#_Toc34394896)

[Question A 2](#_Toc34394897)

[Question B 4](#_Toc34394898)

[Question C 8](#_Toc34394899)

[Trains.java class 8](#_Toc34394900)

[TrainTrack.java class 11](#_Toc34394901)

[Activity.java 29](#_Toc34394902)

[CDS.java 32](#_Toc34394903)

[MageeSemaphore.java 33](#_Toc34394904)

[TrainA.java 34](#_Toc34394905)

[TrainB.java 35](#_Toc34394906)

[Console Output 36](#_Toc34394907)

## Question A

// TrainA

**process** TrainA (string name) {

// limit trains on track to max 8

<**await** (trainsOnTrack < 8) trainsOnTrack++;>

// wait for slots 4 and 5 to be free and then move into slot[5]

<**await** ( (slots[4]==”[...]”) && (slot[5]==”[...]”) slots[5] = "[" + trainName + "]";

// train moves around from section 5 to section 8

**int** currentPosition = 5;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 8)

// wait for the junction to be clear and then grab access to it

<**await** (!junctionOccupied) junctionOccupied = true;>

// progress through the junction (first time)

slots[9] = slots[8]; slots[8] = “[…]”;

slots[0] = slots[9]; slots[9] = “[…]”;

slots[10] = slots[0]; slots[0] = “[…]”;

slots[11] = slots[10]; slots[10] = “[…]”;

// release access to the junction

< junctionOccupied = false;>

// move around B loop from junction to junction

**int** currentPosition = 11;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 17)

// wait for the junction to be clear and then grab access to it

<**await** (!junctionOccupied) junctionOccupied = true;>

// progress through the junction (second time)

slots[18] = slots[17]; slots[17] = “[…]”;

slots[0] = slots[18]; slots[18] = “[…]”;

slots[1] = slots[0]; slots[0] = “[…]”;

slots[2] = slots[1]; slots[1] = “[…]”;

// release access to the junction

< junctionOccupied = false;>

// move around A loop from junction and exit

**int** currentPosition = 2;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 5)

// exit the track

slots[5] = “[…]”;

<trainsOnTrack--;> // leaving the track and make space for another train

} // end TrainA

// TrainB

**process** TrainB (string name) {

// limit trains on track to max 8

<**await** (trainsOnTrack < 8) trainsOnTrack++;>

// wait for slots 13 and 14 to be free and then move into slot[14]

<**await** ( (slots[13]==”[...]”) && (slot[14]==”[...]”) slots[5] = "[" + trainName + "]";

// train moves around from section 14 to section 17

**int** currentPosition = 14;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 17)

// wait for the junction to be clear and then grab access to it

<**await** (!junctionOccupied) junctionOccupied = true;>

// progress through the junction (first time)

slots[18] = slots[17]; slots[17] = “[…]”;

slots[0] = slots[18]; slots[18] = “[…]”;

slots[1] = slots[0]; slots[0] = “[…]”;

slots[2] = slots[1]; slots[1] = “[…]”;

// release access to the junction

< junctionOccupied = false;>

// move around A loop from junction to junction

**int** currentPosition = 2;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 8)

// wait for the junction to be clear and then grab access to it

<**await** (!junctionOccupied) junctionOccupied = true;>

// progress through the junction (second time)

slots[9] = slots[8]; slots[8] = “[…]”;

slots[0] = slots[9]; slots[9] = “[…]”;

slots[10] = slots[0]; slots[0] = “[…]”;

slots[11] = slots[10]; slots[10] = “[…]”;

// release access to the junction

< junctionOccupied = false;>

// move around B loop from junction and exit

**int** currentPosition = 11;

**do**  {

// wait until the position ahead is empty and then move into it

<**await** (slots[currentPosition+1]==”[...]”) slots[currentPosition+1]=slots[currentPosition];

slots[currentPosition]=”[...]”; **>**

currentPosition++;

} **while** (currentPosition < 14)

// exit the track

slots[14] = “[…]”;

<trainsOnTrack--;> // leaving the track and make space for another train

} // end TrainB

## Question B

sharedTrackLock = new Semaphore(1); //initialize shared track as open

slotsSem[] = new Semaphore(19); //initialize slots semaphores

slots[] = {"[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]",

"[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]"}; // Array of slots

// TrainA

**process** TrainA (string name) {

// limit trains on track to max 4 A trains

trainASem = new Semaphore(4);

aMutexSem = new Semaphore(1); //initialize mutually exclusive semaphore

aUsingSharedTrack = new AtomicInteger(0); //set using shared track to false;

trainASem.P();

// wait for slots 4 and 5 to be free and then move into slot[5]

slotSem[4].P();

slotSem[4].V();

slotSem[5].P();

// train moves around from section 5 to section 8

**int** currentPosition = 5;

**do**  {

// wait until the position ahead is empty and then move into it

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 8)

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTrack

if (aUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move on to shared track

slotSem[9].P(); //Check 9 free

slotSem[18].P(); //Check 18 free

slotSem[18].V(); //Release 18

slots[9] = slots[8];

slots[8] = "[..]";

slotSem[8].V(); //move from slot[8] to slot[9]

// move along shared track

slotSem[0].P();

slots[0] = slots[9];

slots[9] = "[..]";

slotSem[9].V(); //move from slot[9] to slot[0]

// move along shared track

slotSem[10].P();

slots[10] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[10]

// Move off shared track

slotSem[11].P();

slots[11] = slots[10];

slots[10] = "[..]";

slotSem[10].V(); //move from slot[10] to slot[11]

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTracK

if (aUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move around B loop from junction to junction

**int** currentPosition = 11;

**do**  {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 17)

// wait for the junction to be clear and then grab access to it

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTrack

if (aUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move on to shared track

slotSem[18].P();

slotSem[9].P();

slotSem[9].V();

slots[18] = slots[17];

slots[17] = "[..]";

slotSem[17].V(); //move from slot[17] to slot[18]

// move along shared track

slotSem[0].P();

slots[0] = slots[18];

slots[18] = "[..]";

slotSem[18].V(); //move from slot[18] to slot[0]

// move along shared track

slotSem[1].P();

slots[1] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[1]

// Move off shared track

slotSem[2].P();

slots[2] = slots[1];

slots[1] = "[..]";

slotSem[1].V(); //move from slot[1] to slot[2]

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTracK

if (aUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move around A loop from junction and exit

**int** currentPosition = 2;

**do**  {

// wait until the position ahead is empty and then move into it

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 5)

// exit the track

slots[5] = “[…]”;

slotSem[5].V();// signal slot 5 to be free

trainASem.V(); // signal space for another A train

} // end TrainA

// TrainB

**process** TrainB (string name) {

// limit trains on track to max 4 B trains

trainBSem = new Semaphore(4);

bMutexSem = new Semaphore(1); //initialize mutually exclusive semaphore

bUsingSharedTrack = new AtomicInteger(0); //set using shared track to false;

trainBSem.P();

// wait for slots 13 and 14 to be free and then move into slot[5]

slotSem[13].P();

slotSem[13].V();

slotSem[14].P();

// train moves around from section 14 to section 17

**int** currentPosition = 14;

**do**  {

// wait until the position ahead is empty and then move into it

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainBctivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 17)

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTrack

if (bUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTracK

if (bUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

// move on to shared track

slotSem[18].P();

slotSem[9].P();

slotSem[9].V();

slots[18] = slots[17];

slots[17] = "[..]";

slotSem[17].V(); //move from slot[17] to slot[18]

// move along shared track

slotSem[0].P();

slots[0] = slots[18];

slots[18] = "[..]";

slotSem[18].V(); //move from slot[18] to slot[0]

// move along shared track

slotSem[1].P();

slots[1] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[1]

// Move off shared track

slotSem[2].P();

slots[2] = slots[1];

slots[1] = "[..]";

slotSem[1].V(); //move from slot[1] to slot[2]

// move around A loop from junction to junction

**int** currentPosition = 2;

**do**  {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainBctivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 8)

// wait for the junction to be clear and then grab access to it

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTrack

if (bUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

// move on to shared track

slotSem[9].P(); //Check 9 free

slotSem[18].P(); //Check 18 free

slotSem[18].V(); //Release 18

slots[9] = slots[8];

slots[8] = "[..]";

slotSem[8].V(); //move from slot[8] to slot[9]

// move along shared track

slotSem[0].P();

slots[0] = slots[9];

slots[9] = "[..]";

slotSem[9].V(); //move from slot[9] to slot[0]

// move along shared track

slotSem[10].P();

slots[10] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[10]

// Move off shared track

slotSem[11].P();

slots[11] = slots[10];

slots[10] = "[..]";

slotSem[10].V(); //move from slot[10] to slot[11]

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTracK

if (bUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

// move around B loop from junction and exit

**int** currentPosition = 11;

**do**  {

// wait until the position ahead is empty and then move into it

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainBctivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} **while** (currentPosition < 14)

// exit the track

slots[14] = “[…]”;

slotSem[14].V();// signal slot 14 to be free

bCountSem.V(); // signal space for another A train

} // end TrainB

## Question C

### Trains.java class

public class Trains {

// Note. You can assuming that trains approaching the track will

// adhere to normal protocol.

static final int NUM\_OF\_A\_TRAINS = 10;

static final int NUM\_OF\_B\_TRAINS = 10;

static TrainTrack theTrainTrack;

public static void main(String[] args) {

// create a train track

theTrainTrack = new TrainTrack();

System.out.println("STARTED");

// create arrays to hold the trains

TrainA[] trainA = new TrainA[NUM\_OF\_A\_TRAINS];

TrainB[] trainB = new TrainB[NUM\_OF\_B\_TRAINS];

// create trains to enter the track

for (int i = 0; i < NUM\_OF\_A\_TRAINS; i++) {

CDS.idleQuietly((int) (Math.random() \* 500));

trainA[i] = new TrainA("A" + i, theTrainTrack);

}

for (int i = 0; i < NUM\_OF\_B\_TRAINS; i++) {

CDS.idleQuietly((int) (Math.random() \* 500));

trainB[i] = new TrainB("B" + i, theTrainTrack);

}

// set the train processes running

for (int i = 0; i < NUM\_OF\_A\_TRAINS; i++) {

trainA[i].start();

} // end for

for (int i = 0; i < NUM\_OF\_B\_TRAINS; i++) {

trainB[i].start();

} // end for

// trains now travelling

// wait for all the train threads to finish before printing out final message.

for (int i = 0; i < NUM\_OF\_A\_TRAINS; i++) {

try {

trainA[i].join();

} catch (InterruptedException ex) {

}

} // end for

for (int i = 0; i < NUM\_OF\_B\_TRAINS; i++) {

try {

trainB[i].join();

} catch (InterruptedException ex) {

}

} // end for

// Display all the train activity that took place

theTrainTrack.theTrainActivity.printActivities();

// Final message

System.out.println("All trains have successfully travelled 1 circuits of their track loop ");

} // end main

} // end Trains class

### TrainTrack.java class

import java.util.concurrent.atomic.\*;

public class TrainTrack {

private void Idle(int time) {

CDS.idleQuietly((int) (Math.random() \* time));

}

private final String[] slots = {"[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]",

"[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]", "[..]"};

// declare array to hold the Binary Semaphores for access to track slots (sections)

private final MageeSemaphore slotSem[] = new MageeSemaphore[19];

// reference to train activity record

Activity theTrainActivity;

// global count of trains on shared track

AtomicInteger aUsingSharedTrack;

AtomicInteger bUsingSharedTrack;

// counting semaphore to limit number of trains on track

MageeSemaphore aCountSem;

MageeSemaphore bCountSem;

// declare Semaphores for mutually exclusive access to aUsingSharedTrack

private final MageeSemaphore aMutexSem;

// declare Semaphores for mutually exclusive access to bUsingSharedTrack

private final MageeSemaphore bMutexSem;

// shared track lock

MageeSemaphore sharedTrackLock;

/\* Constructor for TrainTrack \*/

public TrainTrack() {

// record the train activity

theTrainActivity = new Activity(slots);

// create the array of slotSems and set them all free (empty)

for (int i = 0; i <= 18; i++) {

slotSem[i] = new MageeSemaphore(1);

}

// create semaphores for mutually exclusive access to global count

aMutexSem = new MageeSemaphore(1);

bMutexSem = new MageeSemaphore(1);

// create global AtomicInteger count variables

aUsingSharedTrack = new AtomicInteger(0);

bUsingSharedTrack = new AtomicInteger(0);

// create semaphores for limiting number of trains on track

aCountSem = new MageeSemaphore(4);

bCountSem = new MageeSemaphore(4);

// initially shared track is accessible

sharedTrackLock = new MageeSemaphore(1);

} // constructor

public void trainA\_MoveOnToTrack(String trainName) {

Idle(100);

aCountSem.P(); // limit number of trains on track to avoid deadlock

// record the train activity

slotSem[5].P();// wait for slot 5 to be free

slots[5] = "[" + trainName + "]"; // move train type A on to slot zero

theTrainActivity.addMovedTo(5); // record the train activity

}// end trainA\_movedOnToTrack

public void trainB\_MoveOnToTrack(String trainName) {

// record the train activity

bCountSem.P(); // limit number of trains on track to avoid deadlock

Idle(100);

slotSem[14].P();// wait for slot 14 to be free

slots[14] = "[" + trainName + "]"; // move train type B on to slot sixteen

theTrainActivity.addMovedTo(14); // record the train activity

}// end trainB\_movedOnToTrack

public void trainA\_MoveOffTrack(String trainName) {

Idle(100);

// record the train activity

slots[5] = "[..]"; // move train type A off slot zero

slotSem[5].V();// signal slot 5 to be free

Idle(100);

aCountSem.V(); // signal space for another A train

theTrainActivity.addMovedOff(trainName);

}// end trainA\_movedOffTrack

public void trainB\_MoveOffTrack(String trainName) {

Idle(100);

// record the train activity

slots[14] = "[..]"; // move train type A off slot zero

slotSem[14].V();// signal slot 0 to be free

Idle(100);

bCountSem.V(); // signal space for another B train

theTrainActivity.addMovedOff(trainName);

}// end trainB\_movedOffTrack

public void trainA\_MoveAroundToSharedTrackPart1(String trainName) {

Idle(100);

int currentPosition = 5;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 8);

Idle(100);

} // end trainA\_MoveAroundToSharedTrackPart1

public void trainA\_MoveAlongSharedTrackPart1(String trainName) {

// wait for the necessary conditions to get access to shared track

Idle(100);

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTrack

if (aUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move on to shared track

slotSem[9].P();

slotSem[18].P();

slotSem[18].V();

slots[9] = slots[8];

slots[8] = "[..]";

slotSem[8].V(); //move from slot[8] to slot[9]

theTrainActivity.addMovedTo(9); //record the train activity

// move along shared track

slotSem[0].P();

slots[0] = slots[9];

slots[9] = "[..]";

slotSem[9].V(); //move from slot[9] to slot[0]

theTrainActivity.addMovedTo(0); // record the train activity

// Move off shared track

slotSem[10].P();

slots[10] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[10]

theTrainActivity.addMovedTo(10); // record the train activity

Idle(100);

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTracK

if (aUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

Idle(100);

}// end trainA\_MoveAlongSharedTrackPart1

public void trainA\_MoveAroundToSharedTrackPart2(String trainName) {

Idle(100);

int currentPosition = 10;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 17);

Idle(100);

} // end trainA\_MoveAroundToSharedTrackPart2

public void trainA\_MoveAlongSharedTrackPart2(String trainName) {

// wait for the necessary conditions to get access to shared track

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTrack

if (aUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

// move on to shared track

slotSem[18].P();

slotSem[9].P();

slotSem[9].V();

slots[18] = slots[17];

slots[17] = "[..]";

slotSem[17].V(); //move from slot[17] to slot[18]

theTrainActivity.addMovedTo(18); //record the train activity

Idle(100);

// move along shared track

slotSem[0].P();

slots[0] = slots[18];

slots[18] = "[..]";

slotSem[18].V(); //move from slot[18] to slot[0]

theTrainActivity.addMovedTo(0); // record the train activity

// Move off shared track

slotSem[1].P();

slots[1] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[1]

theTrainActivity.addMovedTo(1); // record the train activity

Idle(100);

aMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTracK

if (aUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

aMutexSem.V(); // release mutually exclusive access to global variable aUsingSharedTrack

Idle(100);

}// end trainA\_MoveAlongSharedTrackPart1

public void trainA\_MoveAroundToSharedTrackPart3(String trainName) {

Idle(100);

int currentPosition = 1;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 5);

Idle(100);

} // end trainA\_MoveAroundToSharedTrackPart3

public void trainB\_MoveAroundToSharedTrackPart1(String trainName) {

Idle(100);

int currentPosition = 14;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 17);

Idle(100);

} // end trainB\_MoveAroundToSharedTrackPart1

public void trainB\_MoveAlongSharedTrackPart1(String trainName) {

// wait for the necessary conditions to get access to shared track

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTrack

if (bUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

// move on to shared track

slotSem[9].P();

slotSem[9].V();

slotSem[18].P();

slots[18] = slots[17];

slots[17] = "[..]";

slotSem[17].V(); //move from slot[17] to slot[18]

theTrainActivity.addMovedTo(18); //record the train activity

Idle(100);

// move along shared track

slotSem[0].P();

slots[0] = slots[18];

slots[18] = "[..]";

slotSem[18].V(); //move from slot[18] to slot[0]

theTrainActivity.addMovedTo(0); // record the train activity

// Move off shared track

slotSem[1].P();

slots[1] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[1]

theTrainActivity.addMovedTo(1); // record the train activity

Idle(100);

bMutexSem.P(); // obtain mutually exclusive access to global variable aUsingSharedTracK

if (bUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

Idle(100);

}// end trainB\_MoveAlongSharedTrackPart1

public void trainB\_MoveAroundToSharedTrackPart2(String trainName) {

Idle(100);

int currentPosition = 1;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 8);

Idle(100);

} // end trainB\_MoveAroundToSharedTrackPart1

public void trainB\_MoveAlongSharedTrackPart2(String trainName) {

// wait for the necessary conditions to get access to shared track

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTrack

if (bUsingSharedTrack.incrementAndGet() == 1)// if first A train joining shared track

{

sharedTrackLock.P(); // grab lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

// move on to shared track

slotSem[18].P();

slotSem[18].V();

slotSem[9].P();

slots[9] = slots[8];

slots[8] = "[..]";

slotSem[8].V(); //move from slot[8] to slot[9]

theTrainActivity.addMovedTo(9); //record the train activity

Idle(100);

// move along shared track

slotSem[0].P();

slots[0] = slots[9];

slots[9] = "[..]";

slotSem[9].V(); //move from slot[9] to slot[0]

theTrainActivity.addMovedTo(0); // record the train activity

// Move off shared track

slotSem[10].P();

slots[10] = slots[0];

slots[0] = "[..]";

slotSem[0].V(); //move from slot[0] to slot[10]

theTrainActivity.addMovedTo(10); // record the train activity

Idle(100);

bMutexSem.P(); // obtain mutually exclusive access to global variable bUsingSharedTracK

if (bUsingSharedTrack.decrementAndGet() == 0) // if last A train leaving shared track

{

sharedTrackLock.V(); // release lock to shared track

}

bMutexSem.V(); // release mutually exclusive access to global variable bUsingSharedTrack

Idle(100);

}// end trainA\_MoveAlongSharedTrackPart1

public void trainB\_MoveAroundToSharedTrackPart3(String trainName) {

Idle(100);

int currentPosition = 10;

do {

/\* wait until the position ahead is empty and then move into it\*/

slotSem[currentPosition + 1].P(); // wait for the slot ahead to be free

slots[currentPosition + 1] = slots[currentPosition]; // move train forward

slots[currentPosition] = "[..]"; //clear the slot the train vacated

theTrainActivity.addMovedTo(currentPosition + 1); //record the train activity

slotSem[currentPosition].V(); //signal slot you are leaving

currentPosition++;

} while (currentPosition < 14);

Idle(100);

} // end trainB\_MoveAroundToSharedTrackPart3

} // end Train track

### Activity.java

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.Iterator;

// - Represents the train track activity in a thread-safe CopyOnWriteArrayList<String>

// - called theActivities

// - addMovementTo(<Integer>) adds a train movement (destination) activity to the record

// - addMessage(<String>) adds a message to the record

// - printActivities() display all the activity history of the train movement

// - trackString() takes a snapshot of the traintrack (with trains) for printing

public class Activity {

private final CopyOnWriteArrayList<String> theActivities;

private final String[] trainTrack;

// Constructor for objects of class Activity

// A reference to the track is passed as a parameter

public Activity(String[] trainTrack) {

theActivities = new CopyOnWriteArrayList<>();

this.trainTrack = trainTrack;

}

public void addMovedTo(int section) {

// add an activity message to the activity history

String tempString1 = "Train " + trainTrack[section] + " moving/moved to [" + section + "]";

theActivities.add(tempString1);

// add the current state of the track to the activity history

theActivities.add(trackString());

}// end addMovedTo

public void addMovedOff(String trainName) {

// add an activity message to the activity history

String tempString1 = "Train " + trainName + " has left the track";

theActivities.add(tempString1);

// add the current state of the track to the activity history

theActivities.add(trackString());

}// end addMovedTo

public void addMessage(String message) {

// add an activity message to the activity history

String tempString1 = message;

theActivities.add(tempString1);

}// end addMessage

public void printActivities() {

// print all the train activity history

System.out.println("TRAIN TRACK ACTIVITY(Tracks [0..18])");

Iterator<String> iterator = theActivities.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

}// end printActivities

// Utility method to represent the track as a string for printing/display

public String trackString() {

String trackStateAsString = " " + trainTrack[5] + "\n"

+ " " + trainTrack[4] + " " + trainTrack[6] + "\n"

+ " " + trainTrack[3] + " " + trainTrack[7] + "\n"

+ " " + trainTrack[2] + " " + trainTrack[8] + "\n"

+ " " + trainTrack[1] + " " + trainTrack[9] + "\n"

+ " " + trainTrack[0] + "\n"

+ " " + trainTrack[10] + " " + trainTrack[18] + "\n"

+ " " + trainTrack[11] + " " + trainTrack[17] + "\n"

+ " " + trainTrack[12] + " " + trainTrack[16] + "\n"

+ " " + trainTrack[13] + " " + trainTrack[15] + "\n"

+ " " + trainTrack[14] + "\n";

return trackStateAsString;

}// end trackString

} // end Activity

### CDS.java

public class CDS {

public static void idle(int millisecs) { // with messages

Thread mainThread = Thread.currentThread();

System.out.println(mainThread.getName() + ": About to sleep");

try {

Thread.sleep(millisecs);

} catch (InterruptedException e) {

}

System.out.println(mainThread.getName() + ": Woken up");

} // end idle

public static void idleQuietly(int millisecs) { // idle with no messages

try {

Thread.sleep(millisecs);

} catch (InterruptedException e) {

}

} // end idleQuietly

} // end CDS

### MageeSemaphore.java

import java.util.concurrent.\*;

//MageeSemaphore.java

//This is an implementation of the traditional (counting) Semaphore with P() and V() operations

class MageeSemaphore

{

private Semaphore sem;

public MageeSemaphore (int initialCount)

{

sem = new Semaphore(initialCount);

} // end constructor

public void P()

{

try {

sem.acquire();

} catch (InterruptedException ex) {System.out.println("Interrupted when waiting");}

} // end P()

public void V()

{

sem.release();

} // end V()

} // end MageeSemaphore

### TrainA.java

public class TrainA extends Thread {

String trainName;

TrainTrack theTrack;

//initialise (constructor)

public TrainA(String trainName, TrainTrack theTrack) {

this.trainName = trainName;

this.theTrack = theTrack;

}

@Override

public void run() { // start train Process

// wait for clearance before moving on to the track

theTrack.trainA\_MoveOnToTrack(trainName); // move on to track A

int circuitCount = 0;

while (circuitCount < 1) { // keep cycling the A track loop

theTrack.trainA\_MoveAroundToSharedTrackPart1(trainName);

theTrack.trainA\_MoveAlongSharedTrackPart1(trainName);

theTrack.trainA\_MoveAroundToSharedTrackPart2(trainName);

theTrack.trainA\_MoveAlongSharedTrackPart2(trainName);

theTrack.trainA\_MoveAroundToSharedTrackPart3(trainName);

circuitCount++;

}

theTrack.trainA\_MoveOffTrack(trainName); // move off the track

} // end run } // end trainAProcess

### TrainB.java

public class TrainB extends Thread {

String trainName;

TrainTrack theTrack;

//initialise (constructor)

public TrainB(String trainName, TrainTrack theTrack) {

this.trainName = trainName;

this.theTrack = theTrack;

}

@Override

public void run() { // start train Process

// wait for clearance before moving on to the track

theTrack.trainB\_MoveOnToTrack(trainName); // move on to track B

int circuitCount = 0;

while (circuitCount < 1) { // keep cycling the B track loop

theTrack.trainB\_MoveAroundToSharedTrackPart1(trainName);

theTrack.trainB\_MoveAlongSharedTrackPart1(trainName);

theTrack.trainB\_MoveAroundToSharedTrackPart2(trainName);

theTrack.trainB\_MoveAlongSharedTrackPart2(trainName);

theTrack.trainB\_MoveAroundToSharedTrackPart3(trainName);

circuitCount++;

}

theTrack.trainB\_MoveOffTrack(trainName); // move off the track \*/

} // end run } // end trainBProcess

### Console Output

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated