גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (10 נקודות)

@INV: 0<=size()<=capacity()=to()-from()+1

המחלקה בטוחה כי הגישה לשדות שאינם final מסונכרנת, והתוכנה הנשמרת לעייל תמיד מתקיימת

אי בדיקת ערכי toHeight ו fromHeight בבנאי, והאינדקס i ב setPedestrian(), אינה פוגמת באינווריאנטה, כי יזרק Exception (ביצירת המערך pedestrians_, ובגישה אליו).

אם כוללים באינווריאנטה את האילוץ, שהולך רגל אינו מופיע בשתי מדרגות בו זמנית (V 0<=i,j<capacity(), i != j → getPedestrian(i) != getPedestrian(j)) התכונה לא מתקיימת בהכרח בקוד הנתון.

תנאי התחלה: 5 נקודות

בטיחות: 5 נקודות

סעיף ב (12 נקודות)

```
public synchronized void advance(StairCase stairCase) throws InterruptedException {
//@PRE:   @param staircase != null &&
          (stairCase.getPedestrian(getHeight() - staircase.fromHeight()) == this ||
           getHeight() == @param staircase.fromHeight() ||
           getHeight() == @param staircase.toHeight()) &&
          stairCase.getPedestrian(getStrategy().next(getHeight()) - staircase. fromHeight()) == null
//@POST: getHeight() == getStrategy().next(@pre(getHeight()) &&
          stairCase.getPedestrian(@pre(getHeight() - staircase.fromHeight()) == null  &&
          stairCase.getPedestrian(getHeight() - staircase.fromHeight()) == this

synchronized(staircase) {
      while (!(staircase != null &&
             staircase.getPedestrian(_strategy.next(getHeight()) - staircase. fromHeight()) == null &&
             (_height == staircase.fromHeight() || _height == staircase.toHeight() ||
              staircase.getPedestrian(_height - staircase. fromHeight()) == this)))
          staircase.wait();
       staircase.setPedestrian(null, _height - staircase.fromHeight());
      _height = _strategy.next(_height);
      staircase.setPedestrian(this, _height - staircase.fromHeight());
      staircase.notifyAll();
    }
}
```

הערה: בפיתרון זה קיימת סכנת חבק, בשל תפישת המנעול על this עם היציאה ל wait על staircase (המשחררת רק אותו).

מפתח ניקוד:
תנאי התחלה: 3 נקודות
תנאי סיום: 2 נקודות
מימוש: 7 נקודות
בבדיקת המימוש הושם דגש על הנקודות הבאות
- מנגנון המתנה המבוסס על wait/notify
- שימוש במוניטור מתאים (staircase)
- רצף סנכרון בין בדיקת תנאי התחלה ועד ביצוע הפעולה
- חישוב נכון של האינדקס בגרם המדרגות ביחס לגובה הולך הרגל
- עדכון גובה הולך הרגל
- עדכון המדרגות

סעיף ג (8 נקודות)

חבק של המתנה הדדית עשוי להתרחש, בין שני הולכי רגל האחד עולה והשני יורד, הנפגשים בשתי מדרגות סמוכות. חבק זה – המכונה בשפה המקצועית חמור גֶרֶם – משתחרר בנוכחות ת'רד המבצע את המשימה StairCaseMovementTask, המביאה את הולכי הרגל החבוקים למפלס פתוח.

שאלה 2 (30 נקודות)

סעיף א (18 נקודות)

```
class StairCase
{
public:
  virtual ~StairCase();
   virtual int  fromHeight() const = 0;
   virtual int  toHeight() const = 0;
   virtual Pedestrian *getPedestrian(int i) const = 0;
   virtual void setPedestrian(Pedestrian *pedestrian ,int i) = 0;
   virtual int size() const = 0;
   virtual int capacity() const = 0;
}

class StairCaseImpl : public StairCase
{
 private:
  Pedestrian  **_pedestrians;
  int _fromHeight;
  int _toHeight;

public:
StairCaseImpl(int fromHeight, int  toHeight) {
```

```
      _fromHeight = fromHeight;
      _toHeight = toHeight;
      _size=0;
      _pedestrians  = new Pedestrian*[_toHeight  - _fromHeight + 1] ;
  }
virtual ~StairCaseImpl(){
  delete[] _pedestrians;
}

StairCaseImpl(const StairCase& other){
  _fromHeight = other.fromHeight();
  _toHeight = other. toHeight();
  _pedestrians  = new Pedestrian*[other.capacity()] ;
  for(int i=0; i other.capacity(); i++){
     _pedestrians[i] =  other. getPedestrian(i);
}

StairCaseImpl & operator=(const StairCase& other) {
  _fromHeight = other. fromHeight();
  _toHeight = other. toHeight();
  delete[] __pedestrians;
  _pedestrians  = new Pedestrian*[other.capacity()] ;
  for(int i=0; i other.capacity(); i++){
     _pedestrians[i] =  other. getPedestrian(i);
  }
}

int  fromHeight () const { return _fromHeight; }
int toHeight () const { return _toHeight; }
int capacity() const {  return _toHeight  - _fromHeight + 1;  }
int size() const {
  int size=0;
  for(int i=0; i<_toHeight  - _fromHeight + 1; i++){
    if (_pedestrians[i]!=null)
      size++;
  }
  return size;
}

 Pedestrian* getPedestrian(int i)  const {
     return _pedestrians[i];
 }

void setPedestrian(Pedestrian *p, int i) {
```

```
  _pedestrians[i] = p;
}

};
```

סעיף ב (12 נקודות)

Stack		
address	value	meaning
...		
1040	1020	pointer to upStairCase2
1036	return address from Q	
1032	39	_toHeight
1028	1	_fromHeight
1024	7000	pointer to _pedestrians in heap
1020	*vtable	
1016	6000	pointer to downStairCase in heap
1012	39	_toHeight
1008	1	_fromHeight
1004	5000	pointer to _pedestrians in heap
1000	*vtable	

Heap		
address	value	meaning
...		
7000	_pedestrians[39]	_pedestrians array
...		
6012	_pedestrians[39]	_pedestrians array
6008	39	_toHeight
6004	1	_fromHeight
6000	*vtable	
...		
5000	_pedestrians[39]	_pedestrians array
...		

Grading Key

1. Missing inheritance, missing implementation of StairCase, StairCaseImpl, missing virtual methods -8
2. C++ Syntax, Constructor (ctor), Destructor (dtor) [-1 - -4]
3. Definition, allocation and free of pedestrian array: -4
4. Copy Constructor (copy ctor), Operator= -8
5. Memory diagram
    a. each missing entry -3
    b. missing vtable pointers -4 (no multiple deduction if missing inheritance above)
    c. functions, return address (r.a), parameters on stack -3, stack, heap confusions -3

שאלה 3 (30 נקודות)

סעיף א (10 נקודות)

What are the changes required to make StairCase are remote interface are the following (as documented in **http://www.cs.bgu.ac.il/~spl111/RMI**):

1. On the StairCase interface:
- Mark the interface remote by extending java.rmi.Remote
- Add java.rmi.RemoteException to all methods in the interface

2. On the StairCaseImpl implementation (the remote object):
- Inherit from the java.rmi.server.UnicastRemoteObject
- Add java.rmi.RemoteException to all methods of the class

3. In the process that publishes the remote object (the class Simulation), make the remote object available, by registering it in the rmiRegistry:
- Naming.rebind("132.87.45.3:4004/StairCase1", upstairCase);
- In the specific case, it makes sense to add a method to the interface to indicate it is now "under control" of the remote controller that ensures the staircase movement – but this was not expected as part of this answer.

4. Make sure all the parameters passed to and returned by remote methods are either:
- Primitive types (int, char…)
- Serializable types
- Remote objects
In our case, the parameters passed in the StairCase interface are: int and Pedestrian.
So we had to make sure that class Pedestrian is Serializable.

[Making this list was sufficient to get credit – issue 4 was not taken into account in the grading, any missing indication of 1, 2 or 3 above caused a 3 point deduction]

Most common errors:
- Not listing the RemoteExceptions

- Marking that a class or an interface throws exception (I was shocked how many students wrote something that strange). A class does not throw an exception – only methods do.
- Confuse extends and implements (the remote interface extends java.rmi.Remote, the remote object extends java.rmi.server.UnicastRemoteObject).
- Forget about the Serializable condition on parameters (was not counted in the grading)

In code:

```
interface StairCase extends java.rmi.Remote {
  int  fromHeight() throws java.rmi.RemoteException;
  int toHeight() throws java.rmi.RemoteException;
  Pedestrian getPedestrian(int i) throws java.rmi.RemoteException;
  void setPedestrian(Pedestrian pedestrian ,int i) throws java.rmi.RemoteException;
  int size() throws java.rmi.RemoteException;
  int capacity() throws java.rmi.RemoteException;
}

class StairCaseImpl implements StairCase extends java.rmi.server.UnicastRemoteObject {

  private final Pedestrian[] _pedestrians;
  private final int _fromHeight;
  private final int _toHeight;

  StairCaseImpl(int fromHeight, int  toHeight) throws java.rmi.RemoteException {
     _fromHeight = fromHeight;
     _toHeight = toHeight;
     _pedestrians  = new Pedestrian[_toHeight  - _fromHeight + 1] ;
  }

  public int  fromHeight ()throws java.rmi.RemoteException { return _fromHeight; }
  public int toHeight ()throws java.rmi.RemoteException { return _toHeight; }
  public int capacity()throws java.rmi.RemoteException {  return _pedestrians.length;  }
  public synchronized int size() throws java.rmi.RemoteException {
    int size=0;
    for (Pedestrian p : _pedestrians)
      if (p!=null)
        size++;
    return size;
 }

  public synchronized Pedestrian getPedestrian(int i) throws java.rmi.RemoteException {
    return _pedestrians[i];
  }

  public synchronized void setPedestrian(Pedestrian p, int i) throws java.rmi.RemoteException {
     _pedestrians[i] = p;
```

```
  }
}

class Simulation {
  public static void main(String[] args) {
    StairCase upStairCase = new StairCaseImpl(1, 39);
    Naming.rebind("132.87.45.3:4004/StairCase1", upstairCase);
    Pedestrian pedestrian1 = new Passenger(1,new HurryUp());
    Pedestrian pedestrian2 = new Passenger(39,new HurryDown());
    new Thread(new StairCaseMovementTask (upStairCase,1000)).start();
    // Would make sense to wait until the remote staircase movement task is started – not necessary
    new Thread(new PedestrianMovementTask (pedestrian1,1000, upStairCase)).start();
    new Thread(new PedestrianMovementTask (pedestrian2,1000, upStairCase)).start();
  }
}

interface Pedestrian extends java.io.Serializable {
  int getHeight();
  void setHeight(int height);
  Strategy getStrategy();
  void advance(StairCase stairCase);
}
```

סעיף ב (7 נקודות)

How many communication operations are necessary to move up a staircase by 3 steps?

We look at what the RMI client does:

```
StairCase upStairCase=(StairCase)Naming.lookup("132.87.45.3:4004/StairCase1
new Thread(new StairCaseMovementTask (upStairCase,1000)).start();
```

The Naming.lookup() call causes one round-trip communication with the rmiRegistry server.

The other part is of the process is that the remote object upStairCase is accessed through the run() method of StairCaseMovementTask. So we look at the code of run():

```
public void run() {
    while (true) {
      try {
        for (int i=_staircase .capacity()-1; i>0; i--) {
          _staircase.setPedestrian(_staircase.getPedestrian(i-1),i);
          Pedestrian p = _staircase.getPedestrian(i);
          if (p!=null)
            p.setHeight(p.getHeight()+1);
```

```
        }
        _staircase.setPedestrian(null,0);
        synchromized(_staircase) { _staircase.notifyAll(); }
        Thread.sleep(_speed);
      } catch (InterruptedException e) {}
    }
  }
```

In this code, _staircase is bound to a remote object (that is, it is a stub that sends remote calls to the remote object). This means each call on _staircase is a remote call – and we want to call such remote calls – each call is a round-trip communication operation.

The calls are:
For each iteration in the while(true) loop:
  In the for loop (capacity() times):
    getPedestrian()
    setPedestrian()
  setPedestrian(null)
  [We can either ignore the notifyAll() call as indicated in the question
   or count it as a remote variant]

Total: (capacity() * 2 + 1) calls per iteration.

Each iteration moves the stair by 1 step up.

We want to count 3 steps up:
3 * (capacity() * 2 + 1) + 1 (lookup)

We know that capacity() is 39 in our case – giving:
3 * (39 * 2 + 1) + 1 = 238 round trip operations.

Possible variation:
- Some students assumed that the Pedestrian interface is a remote interface instead of being a serializable interface. This is fine and in this case, p.setHeight() and p.getHeight() must be counted as 2 round-trip calls in the loop.

Most common errors:
- Not counting the operations at all (detail which operations are remote calls).
- Giving a random number as an answer.
- Not taking into account the loop over the capacity().
- Not taking into account the loop for 3 steps
- Not counting the lookup() operation
[Either loops omission was counted as 2 point deduction]

סעיף ג (3 נקודות)

Must the class StairCaseImpl.class be present in the file system side of the process running StairCaseControl?

No it does not – since this is a remote object, which is accessed through the remote interface StairCase. When StairCaseControl executes:

```
StairCase upStairCase=(StairCase)Naming.lookup("132.87.45.3:4004/StairCase1
```

It gets in return an instance of the stub class (StairCaseStub.class) which implements the StairCase interface. It never accesses the stairCaseImpl code, which is only operated on the side of the Simulation process.

סעיף ד (10 נקודות)

Assume the implementation of the skel uses the Reactor pattern.
A student proposes to remove the synchronized directive on the run() method of ProtocolTask, and instead to limit it half of the method (dealing with the tokenizer and protocol and not with the part dealing with the buffers and tokenizer).
- Does the change affect correctness?
- What is the condition that the synchronized run() enforces?

First – why is run() synchronized?
Notes from the reactor pattern:
- A protocolTask instance is associated to a single connectionHandler. That means, a protocolTask will only operate on the same client connection. There is only one protocolTask per connectionHandler.
- The protocolTask is submitted to the thread pool executor each time new data is read by the connectionHandler (in the read() method).

When run() is synchronized, we know that the calls to run() from the executor will be serialized (executed sequentially one after the other) – so that 2 threads from the thread pool cannot execute the same task simultaneously.

If we remove the synchronized directive from run(), then there can be the following situation:
- ConnectionHandler receives a read event from the reactor and posts its task to the thread pool for execution.
- Thread T1 from the thread pool starts running the task.
- Another read event arrives from the reactor to the ConnectionHandler – and the same task is posted again to the thread pool for execution.
- Thread T2 from the thread pool starts running the same task while T1 is still working on the same task.

Can this fact cause correctness problems?
- If the 2 read events come from 2 separate messages m1 and m2 sent from the same client, then this could potentially lead to a situation where the server processes m2 before m1 – and it does not respect the order of messages sent by the client.
- If the 2 read events come from 2 fragments of the same message m1, or from fragments that overlap 2 distinct messages m1 and m2, then potentially, this situation could lead to a situation where the bytes are added to the tokenizer in the wrong order.

Note that the other way to access the protocol task is through the method addBytes() which is invoked by the ConnectionHandler (in the reactor thread), when it receives data in the read() event. But note that addBytes() is NOT synchronized – and this is on purpose. The buffers queue serves as a buffer between the reactor thread (producer) and the thread running the protocolTask (consumer). Between these 2 active objects, only the buffers object needs to be synchronized, and it is properly synchronized in all places where buffers is accessed.

Now we need to look at the details – because the proposed code does not completely remove the synchronization – it moves it from one place to another (smaller scope).
So can we really get the risks highlighted above?

With the synchronization proposed by the student, we have 2 sections – the sync(buffers) part, and the sync(this) part. So we can now get the situation with 2 threads active on the same ProtocolTask instance:
- T1 is inside part1 (moves buffers from buffers to tokenizer)
- T2 is inside part2 (moves messages from tokenizer to protocol, execute protocol and pass return data to connectionHandler).

But we cannot have a situation where T1 and T2 are together in part1 or together in part2.

Risk 1: (protocol processes m2 before m1) cannot happen because the messages are inserted in order from buffers to tokenizer: only the reactor thread adds to buffers so buffers is always ordered, and whether T1 or T2 read from buffers, they always push to the same tokenizer – so tokenizer gets data in the proper order. Since only one thread can read complete messages from tokenizer at a time, and the sequence "read message / execute message" is synchronized, there is no risk of message order inversion.

Risk 2: (buffers with partial messages are pushed into tokenizer in the wrong order) cannot happen because buffers are pushed into tokenizer by one thread at a time and the sequence buffers.remove() / tokenizer.add() is synchronized.

Note that the only element that is shared between part1 and part2 is the tokenizer – and we must make sure that tokenizer is properly fully synchronized. In the reactor code, this is indeed the case.

All in all – the changes proposed by the student are safe.

Note: we do not discuss here whether the changes are beneficial or not to the reactor. They could increase parallelism but they could also reduce scalability – it depends on the pattern of communication between clients and server (many fragmented messages, several messages sent in sequence by the client without getting a response from the server). This is out of the scope of the question.

Grading considered:
- Any mention of message ordering got a credit of 5 points or more.
- Any mention of partial messages ordering got a credit of 3 points or more.
- Any mention that tokenizer must be fully synchronized got a credit of 5 points or more.
- Explanation that "lack of synchronization brings risk" is not sufficient – we expected specific explanation of the risks.

- Explanation that “there is only one client for this server” were wrong. The risks exist even with a single client.

שאלה 4 (30 נקודות)

סעיף א (5 נקודות)

Data Model:

```
create table Location (
 LocationId int primary key,
 LocationName varchar(200),
 City varchar(200),
 Country varchar(100))

create table Staircase (
  StaircaseId int Primary Key,
  LocationId int Foreign Key references Location,
  Orientation int   // 1 = ascending, 2 = descending
)

create table StaircaseOperation (
  StaircaseId int Foreign Key references Staircase,
  OperationDay datetime,
  PassengersNumber int,
  AverageTripDuration int, // average duration in milliseconds
  Primary Key (StaircaseId, OperationDay)
 )
```

Common Errors:

- Types of fields must be specified (int, varchar, date, Boolean) [1 point]
- Primary Keys must be specified for each table [2 to 3 points]
    - Primary Keys must be unique for each table
        [OperationDay alone in StaircaseOperation is not unique, it cannot be PK]
- Foreign Keys must be specified [2 to 3 points]
    - When there is a relation 1-n from T1 to T2, then the foreign key is in table T2.

In our example, the relation Staircase to StaircaseOperations is 1-n (there is one row of Operations for each date for each staircase). Therefore, the foreign key (StaircaseId) is in the table StaircaseOperations (and not the opposite – a date field in the Staircase table).

- Tables must be normalized [Was not penalized]
    - If there is a group of fields that are repeated in many rows, normalization consists of “replacing” these fields with a foreign key to a table where the group of fields are stored. For example, since LocationName, City and Country could possibly be repeated many times for each staircase in each location in the same city, we defined the table Location to normalize the schema.

סעיף ב (5 נקודות)

```
Select
      Staircase.StaircaseId, Location.LocationName,
StaircaseOperation.PassengersNumber
From
      ((Staircase inner join Location
                  on Staircase.locationId = Location.LocationId)
            inner join staircaseOperation
            on staircase.staircaseId = staircaseOperation.staircaseId)
Where
      Location.city = 'Beer Sheva'
      and
      StaircaseOperation.OperationDay = '2011.01.02'
Order by
      StaircaseOperation.PassengersNumber asc
```