A. Compiler of corrections - Giora

Director Examiner: ______________________

Please write your answers only in the answer sheet. Your answers outside the answer sheet will not be checked.

Best wishes


e.

The context of the elevator is a popular railway in Tel Aviv, for example in public places.

In this question, we deal with a simulation of this mechanism. To rise and fall between levels, the stairs move:
The system consists of the passive objects below

- StairCase StairCase StairCase StairCase
- A place for one foot per step.
- A group of pedestrians
- Strategy Strategy Strategy Strategy

The altitude of the lower floor to the altitude of the upper floor, and the number of stairs will be given by N, for example a pedestrian of height 5, and the foot is at 39 stairs above the base floor, say 44.

The strategy of progress is determined by the behavior of a pedestrian on the stairs,

- Three implementations of Strategy Strategy Strategy Strategy, with method progress() which returns the new height to which the pedestrian will arrive. When he stands still and waits for the step, the pedestrian does not go up or down,
- HurryUp HurryUp HurryUp HurryUp
- The pedestrian tries to progress on his own going up
- HurryDown HurryDown HurryDown HurryDown
- The pedestrian tries to progress on his own going down.

In addition to the active objects, two types of threads can be defined in the system:

- Thread StairCaseMovement
- Thread PedestrianMovement

The elevator moves at a given speed. For example, a pedestrian

- Progress StairCaseMovement
- Progress PedestrianMovement

The elevator moves at a given speed. For example, a pedestrian

24.1.2011

Instructor: Michael Alhaddad Prof.

Mani Adler"D

Instructor: Andrew Sherf"D

Course: Programming Systems

Class: 2031

Students: Software, Computer Science

Year: 5762

Semester: A

Date: 

Exam duration: 3 hours

Resources:

1. The context of the elevator is a popular railway in Tel Aviv, for example in public places.

In this question, we deal with a simulation of this mechanism. To rise and fall between levels, the stairs move:
The system consists of the passive objects below

- StairCase StairCase StairCase StairCase
- A place for one foot per step.
- A group of pedestrians
- Strategy Strategy Strategy Strategy

The altitude of the lower floor to the altitude of the upper floor, and the number of stairs will be given by N, for example a pedestrian of height 5, and the foot is at 39 stairs above the base floor, say 44.

The strategy of progress is determined by the behavior of a pedestrian on the stairs,

- Three implementations of Strategy Strategy Strategy Strategy, with method progress() which returns the new height to which the pedestrian will arrive. When he stands still and waits for the step, the pedestrian does not go up or down,
- HurryUp HurryUp HurryUp HurryUp
- The pedestrian tries to progress on his own going up
- HurryDown HurryDown HurryDown HurryDown
- The pedestrian tries to progress on his own going down.

In addition to the active objects, two types of threads can be defined in the system:

- Thread StairCaseMovement
- Thread PedestrianMovement

The elevator moves at a given speed. For example, a pedestrian

- Progress StairCaseMovement
- Progress PedestrianMovement

The elevator moves at a given speed. For example, a pedestrian

24.1.2011

Instructor: Michael Alhaddad Prof.

Mani Adler"D

Instructor: Andrew Sherf"D

Course: Programming Systems

Class: 2031

Students: Software, Computer Science

Year: 5762

Semester: A

Date: 

Exam duration: 3 hours

Resources:
interface Strategy {
    int next(int height);
}

class Relaxed implements Strategy {
    public int next(int height) {
        return height;
    }
}

class HurryUp implements Strategy {
    public int next(int height) {
        return height+1;
    }
}

class HurryDown implements Strategy {
    public int next(int height) {
        return height-1;
    }
}

interface Pedestrian {
    int getHeight();
    void setHeight(int height);
    Strategy getStrategy();
    void advance(StairCase stairCase);
}

class Passenger implements Pedestrian {
    private int _height;
    private final Strategy _strategy;

    Passenger(int height, Strategy strategy) { _height = height; _strategy = strategy; }

    public synchronized Strategy getStrategy() { return _strategy; }
    public synchronized int getHeight() { return _height; }
    public synchronized void setHeight(int height) { _height = height; }
    public void advance(StairCase stairCase) {
        //@TODO
    }
}
interface StairCase {
    int fromHeight();
    int toHeight();
    Pedestrian getPedestrian(int i);
    void setPedestrian(Pedestrian pedestrian, int i);
    int size();
    int capacity();
}

class StairCaseImpl implements StairCase {
    private final Pedestrian[] _pedestrians;
    private final int _fromHeight;
    private final int _toHeight;
    StairCaseImpl(int fromHeight, int toHeight) {
        _fromHeight = fromHeight;
        _toHeight = toHeight;
        _pedestrians = new Pedestrian[_toHeight - _fromHeight + 1];
    }
    public int fromHeight() { return _fromHeight; }
    public int toHeight() { return _toHeight; }
    public int capacity() { return _pedestrians.length; }
    public synchronized int size() {
        int size = 0;
        for (Pedestrian p : _pedestrians)
            if (p!=null)
                size++;
        return size;
    }
    public synchronized Pedestrian getPedestrian(int i) {
        return _pedestrians[i];
    }
    public synchronized void setPedestrian(Pedestrian p, int i) {
        _pedestrians[i] = p;
    }
}

class StairCaseMovementTask implements Runnable {
private final StairCase _staircase;
private final long _speed;
StairCaseMovementTask(StairCase staircase, long speed) { _staircase = staircase; _speed = speed; }
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();
            if (p != null)
                p.setHeight(p.getHeight()+1);
        }
        synchronized(_staircase) {_staircase.notifyAll(); }
        Thread.sleep(_speed);
    } catch (InterruptedException e) { }
} }

class PedestrianMovementTask implements Runnable {
    private final Pedestrian _pedestrian;
    private final StairCase _stairCase;
    private final long _speed;
    PedestrianMovementTask(Pedestrian pedestrian, long speed, StairCase stairCase) { _pedestrian = pedestrian; _speed = speed; _stairCase = stairCase; }
    public void run() {
        while (true) {
            try {
                _pedestrian.advance(_stairCase);
                Thread.sleep(_speed);
            } catch (InterruptedException e) { return; }
        }
    }
}

class Simulation {
    public static void main(String[] args) {
        StairCase upStairCase = new StairCaseImpl(1, 39);
        Pedestrian pedestrian1 = new Passenger(1, new HurryUp());
        Pedestrian pedestrian2 = new Passenger(39, new HurryDown());
        new Thread(new StairCaseMovementTask(upStairCase, 1000)).start();
    }
}
```java
new Thread(new PedestrianMovementTask(pedestrian1,1000, upStairCase)).start();
new Thread(new PedestrianMovementTask(pedestrian2,1000, upStairCase)).start();
}
}

class Simulation {
    public static void main(String[] args) {
        StairCase upStairCase = new StairCaseImpl(1,39);
        Pedestrian pedestrian1 = new Passenger(1, new HurryUp());
        Pedestrian pedestrian2 = new Passenger(39, new HurryDown());
        new Thread(new StairCaseMovementTask(upStairCase, 1000)).start();
        new Thread(new PedestrianMovementTask(pedestrian1,1000, upStairCase)).start();
        new Thread(new PedestrianMovementTask(pedestrian2,1000, upStairCase)).start();
    }
}
```

**Shaal 2 (30 קדורות)**

A. פמס ב C++ את המחלקה `StairCaseImpl` ו- `StairCase` שולח אובייקט מטיפוס `by value` של המחלקה שולח את ההחלפה של שנוי לאובייקט нояוס `(Java, StairCaseImpl לשון קרית ב `StairCaseImpl` assignment operator כ `copy constructor`) StairCase

א. עליך לממש את التنורuggage. [18 קדורות]

B. צייר את התמונות והויפוטו, невозיר בآثار המקודים בתוכי код המחלקה Q [12 קדורות]

```java
void Q(const StairCase& staircase) {
    std::cout << "Mr. Memory, What are the 39 steps?"
}
void main() {
    StairCaseImpl upStairCase(1,39);
    StairCase* downStairCase = new StairCaseImpl(1,39);
    StairCaseImpl upStairCase2 = upStairCase;
    Q(upStairCase2);
}
```

A. תוריד תבנית בשפהית (invariant) שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase`

B. תוריד תבנית בשפהית (invariant) שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase` שבורה הממשיכ (invariant)هام המחלקה `StairCaseImpl` ו- `StairCase`

C++

A. פמס ב C++ את המחלקה `StairCaseImpl` ו- `StairCase` שולח אובייקט מטיפוס `by value` של המחלקה שולח את ההחלפה של שנוי לאובייקטesy =(Java, StairCaseImpl לשון קרית ב `StairCaseImpl` assignment operator כ `copy constructor`) StairCase

א. עליך לממש את التنורuggage. [18 קדורות]

B. צייר את התמונות והויפוטו, невозיר בآثار המקודים בתוכי код המחלקה Q [12 קדורות]

```java
```
 Clippers and sliders are used to control the movement of the stairs. In order to control the system, during a meeting of the company's management, it was decided that the task of moving the stairs would be performed remotely through another process.

The process will be run through rmiregistry in order to receive a stub. The process will then run StairCaseMovementTask.

The simulation will be run on page 4, and after that, the stairs will be moved to the new system.

We changed the classes Simulation, StairCase, StairCaseImpl, so that they will fit the new system.

In the second simulation, we will implement the wait/notify mechanism, without using it in other classes.

It can be assumed that under three conditions, after several communication actions, the moving stairs will be paused.

If the file StairCaseImpl.class is in the process that runs in the file system, StairCaseControl will be used.

It is given that the Skel class, based on the Reactor framework, is used. The method run() in the class ProtocolTask is synchronized, and in order to reduce the synchronization of the method, a student in the course suggested:

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.channels.
```

```java
public class StairCaseControl {
    public static void main(String[] args) {
        try {
            StairCase upStairCase = (StairCase) Naming.lookup("132.87.45.3:4004/StairCase1");
            new Thread(new StairCaseMovementTask(upStairCase,1000)).start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.channels.
```

```java
class ProtocolTask implements Runnable {
    private final ServerProtocol _protocol;
    private final StringMessageTokenizer _tokenizer;
    private final ConnectionHandler _handler;
    /* The fifo queue, which holds data coming from the socket. Access to the queue is serialized, to ensure correct
     * processing order - even if more data is received by the reactor while previous data is still being processed.*/
    private final Vector<ByteBuffer> _buffers = new Vector<ByteBuffer>();
```
public synchronized void run() {
    // first, add all the bytes we have to the tokenizer
    synchronized (_buffers) {
        while (_buffers.size() > 0) {
            ByteBuffer buf = _buffers.remove(0);
            this._tokenizer.addBytes(buf);
        }
    }
}

// now, go over all complete messages and process them.
    synchronized (this) {
        while (_tokenizer.hasMoreMessages()) {
            String msg = _tokenizer.nextMessage();
            String response = this._protocol.processMessage(msg);
            if (response != null) {
                try {
                    ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
                    this._handler.addOutData(bytes);
                } catch (CharacterCodingException e) { e.printStackTrace(); }
            }
        }
    }

    // This is invoked by the ConnectionHandler which runs in the reactor main thread.
    public void addBytes(ByteBuffer b) {
        synchronized (_buffers) {
            _buffers.add(b);
        }
    }

נתנה מתבודד. [10 точки]

הехבם: הבוחרלא מע考えて את הלוח שבשתה שעיבוד במקביל על חלקי שיאים של המizacaoPEC, בפרוטוקול זה בא להבטיח את החלק (어서 המבחן) בנספחניתן למצוא, (זה לא בהכרח נדרש) למעוניינים המחלקה ConnectionHandler ?
לאןокоוב בקורה אחר תיוגות התמועלות של המודרנות הونة באתרים שונים, הוחלט להזמין תמיכה למשרדי הבסיסה.

1. נקבע מושטע על המשמשים בתפקיד:

2. שימו על מידע בשתי בחרים או מערכים:

3. רשימה של הימים בהם הופעל

4. תאריך

5. מספר עוברים

ב. הגדרו שאלות SQL המתייחסות לאכפת האנשימים שערורוב. בכל אצטדיון מודרנויות (שם המתחם)

[5 נקודות]
public class ConnectionHandler {
    protected final SocketChannel _sChannel;
    protected final ReactorData _data;
    protected final AsyncServerProtocol _protocol;
    protected final StringMessageTokenizer _tokenizer;
    protected final Vector<ByteBuffer> _outData;
    protected final SelectionKey _skey;
    private ProtocolTask _task;

    // Post data in the pending data queue, so that the connectionHandler will send it through the socket.
    // switchToReadWriteMode() subscribes this handler key to the OP_WRITE event
    // This event will immediately fire because the output buffer of the channel is empty.
    // It will keep firing as long as the output buffer is not filled.
    // When we are done sending pending data, we will unsubscribe from OP_WRITE.
    public synchronized void addOutData(ByteBuffer buf) {
        _outData.add(buf);
        switchToReadWriteMode();
    }

    // Reads incoming data from the client: Reads some bytes from the SocketChannel
    // Create a protocolTask, to process this data, possibly generating an answer
    // Inserts the Task to the ThreadPool
    public void read() {
        // Do not read if protocol has terminated. Only write of pending data is
        // allowed when the protocol asked to close the connection.
        if (_protocol.shouldClose())
            return;
        SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
        logger.info("Reading from " + address);
        ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
        int numBytesRead = 0;
        try {
            numBytesRead = _sChannel.read(buf);
        } catch (IOException e) {
            numBytesRead = -1;
        }
        if (numBytesRead == -1) // Is the channel closed?
            // No more bytes can be read from the channel
            logger.info("client on " + address + " has disconnected");
            closeConnection();
            // tell the protocol that the connection terminated.
            _protocol.connectionTerminated();
    }
}

public class AsyncServerProtocol {
    protected final StringMessageTokenizer _tokenizer;
    protected final SocketChannel _sChannel;
    protected final ReactorData _data;
    protected final SelectionKey _skey;
    protected Vector<ByteBuffer> _outData;
    protected final String _defaultMessage;
    protected final AsyncServerProtocol _next;
    protected final AsyncServerProtocol _previous;

    public AsyncServerProtocol() {
    }

    public void addOutData(ByteBuffer buf) {
    }
}

private ProtocolTask _task;
return;
}
// Add the buffer to the protocol task
buf.flip();
_task.addBytes(buf);
// Add the protocol task to the reactor which will parse and process the data
// when a thread becomes available for it.
_data.getExecutor().execute(_task);
}

// Attempts to send data to the client. If all the data has been successfully sent, the ConnectionHandler will
// automatically switch to read only mode, otherwise it will stay in its current mode (which is read / write).
public synchronized void write() {
    if (_outData.size() == 0) { // if nothing left in the output string, go back to read mode
        switchToReadOnlyMode();
        return;
    }
    // If there is something to send - send the first byte buffer
    // We will return to this write() operation very soon because the selector will keep firing the OP_WRITE event
    // after we are done writing this buffer and check if there are more buffers to be sent.
    ByteBuffer buf = _outData.remove(0);
    if (buf.remaining() != 0) {
        _sChannel.write(buf);
        // Check if the buffer contains more data: we could not send all of the buffer in one write
        // (the output buffer of the socket got full). So we remember that there is more data to be sent.
        // We will receive a new OP_WRITE event when the output buffer of the socket
        // is not full anymore and complete the write operation then.
        if (buf.remaining() != 0)
            _outData.add(0, buf);
    }
    // Check if the protocol asked us to close this connection.
    // If it did, we remain open as long as there are pending data to be sent.
    // As soon as all the data has been sent, we can close the connection.
    if (_protocol.shouldClose()) {
        switchToWriteOnlyMode();
        if (buf.remaining() == 0) {
            logger.info("disconnecting client on " + _sChannel.socket().getRemoteSocketAddress());
            closeConnection();
        }
    }
    ...
}