גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (5 נקודות)

נניח כי ת'רד אחד מבצע את המשימה PedestrianMovementTask ובמסגרתה מבצע את המתודה advance().
המתודה advance() מסונכרנת, כך שהמנעול של this (הולך הרגל) ננעל. מצד שני מתבצעת נעילה של staircase בגוף המתודה. עם היציאה ל wait() על staircase (כאשר תנאי ההתחלה אינם מתקיימים), משתחרר אוטומטית המנעול של staircase, אך לא המנעול של this.
התירד אמור לצאת מהמתנה כאשר ישתנה מצב גרם המדרגות המשוחרר, על ידי תירד אחר, כמו זה המבצע את המשימה StairCaseMovementTask. אולם ת'רד זה עשוי להיתקע בזמן ביצוע p.setHeight(p.getHeight()+1); הדורש נעילה של הולך הרגל, התפוס עך ידי הת'רד הראשון.

סעיף ב (10 נקודות)

יש לדאוג לשחרור הנעילה על this לפני היציאה להמתנה על staircase ולתפוס את המנעול שוב לאחר שההמתנה מסתיימת.
מנגנון ה Semaphore תומך בנעילה ושחרור היזומים על ידי המתכנת. נשתמש במנגנון זה עבור המחלקה Passenger:
- נגדיר שדה במחלקה מסוג Semaphore המאפשר לת'רד אחד בלבד לעבור
- נחליף את סנכרון המתודות בפעולות acquire() ו release(), בתחילתן ובסופן.
- לפני ה wait() נשחרר את המנעול על ידי ביצוע release(), ולאחריו נתפשהו שוב בעזרת acquire().

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

  Passenger (int height, Strategy strategy)  {
    _height  = height;
    _strategy  = strategy;
    _sem = new Semaphore(1);
  }
  public synchronized Strategy getStrategy() {
    _sem.acquire();
    return _strategy;
    _sem.release();
  }
  public synchronized int getHeight() {
    _sem.acquire();
```

```
    int ret = _height;
    _sem.release();
    return ret;
  }
  public synchronized void setHeight(int height) {
    _sem.acquire();
    _height  = height;
    _sem.release();
  }
  public synchronized void advance(StairCase staircase) throws InterruptedException {
    synchronized(staircase) {
      _sem.acquire();
      while (!(staircase != null &&
              staircase.getPedestrian(_strategy.next(getHeight()) - staircase. fromHeight()) == null &&
              (_height == staircase.fromHeight() || _height == staircase.toHeight() ||
               staircase.getPedestrian(_height - staircase. fromHeight()) == this))) {
        _sem.release();
         staircase.wait();
        _sem.acquire();
      }
      staircase.setPedestrian(null, _height - staircase.fromHeight());
      _height = _strategy.next(_height);
      staircase.setPedestrian(this, _height - staircase.fromHeight());
      staircase.notifyAll();
    }
    _sem.release();
  }
}
```

סעיף ג (5 נקודות)

בזמן ביצוע המשימה StairCaseMovementTask על ידי ת'רד אחד, עשוי הת'רד השני, המבצע את PedestrianMovementTask לשנות את מצב המדרגות והולכי הרגל. ובפרט, לקדם את הולך רגל למדרגה הבאה.
כאשר יחזור זמן ה CPU לת'רד הראשון הוא ישים את אותו הולך רגל במדרגה זו.
כלומר, שתי פעולות הקידום הזיזו את הולך הרגל במדרגה אחת. בהרצה סדרתית, בכל סדר שהוא, הולך הרגל יתקדם בשתי מדרוגות, כך שהרצת התוכנית אינה נכונה.

סעיף ד (10 נקודות)

נאלץ לסנכרן את גרם המדרגות לכל משך ביצוע ה run (כל סנכרון קטן מזה, עשוי לגרום לבעיית נכונות):

```
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 {
        synchronized(_staircase) {
          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);
           synchronized(_staircase) { _staircase.notifyAll(); }
        }
        Thread.sleep(_speed);
      } catch (Exception e) {}
    }
  }
}
```

שאלה 2 (30 נקודות)

סעיף א (8 נקודות)

```
void resize(int fromHeight, int toHeight, StairCaseImpl &staircase){
  StairCaseImpl *s = new StairCaseImpl(fromHeight, toHeight);
  int i  = fromHeight;
  while(i<toHeight && i<staircase.toHeight){
    if (i>=staircase.fromHeight)
      s->setPedestrian(staircase.getPedestrian(i), i);
  }
  staircase=*s;
  delete s;
}
StairCaseImpl & operator=(const StairCase& other){
  if (&other==this) return *this
    _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);
    return *this;
}
```

סעיף ב (12 נקודות)

Test1

Stack		
address	value	meaning
1044	78	_toHeight
1040	1	_fromHeight
1036	6000	
1032	*vtable	
1028	39	_toHeight
1024	1	_fromHeight
1020	5000	
1016	*vtable	
1012	39	_toHeight
1008	1	_fromHeight
1004	5000	pointer to _pedestrians in heap
1000	*vtable	
996	39	toHeight
992	1	fromHeight

Heap		
address	value	meaning
6000	_pedestrians[78]	_pedestrians array
...		
5000	_pedestrians[39]	_pedestrians array
...		

Test2

Stack		
address	value	meaning
...		same as test1
1048	7000	pointer to test2

Heap		
address	value	meaning
7012	39	_toHeight
7008	1	_fromHeight
7004	8000	pointer to _pedestrians in heap
7000	*vtable	

סעיף ג (6 נקודות)

The runtime memory error is due to multiple calls to ~StairCaseImpl() everytime foo is exited out from the recursion. This will attempt deleting _pedestrians[] more than once.
A solution is to change the copy constructor to deep copy:

```
StairCaseImpl(const StairCaseImpl& other): _fromHeight(other._fromHeight), _toHeight(other. _toHeight){
   _pedestrians = new Pedestrian*[other.capacity];
  for (int i=0; i<other.capacity(); i++)
     setPedestrians(other.getPedestrian(i), i);
}
```

סעיף ד (4 נקודות)

The problem is that foo gets StairCaseImpl by value and test2 is defined as StairCase*.
A solution is to define a copy constructor that obtains StairCase& and uses only virtual functions:

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

Grading Key

A.

1. Missing deletion of old / creation of new array -2
2. Missing bounded copy of old passengers to new positions -2
3. Return by copy new StairCaseImpl to staircase parameter -2
4. Missing assignemt operator (if required) -2

B.

5. Missing vtable -2
6. Missing pointers to arrays in heap -2-> -4
7. Missing entries from recursion -2
8. On stack instead of heap -2

C.

9. Other memory problems that are not "run-time" -2

D.

10. Casting of non-pointers -4
11. Changing only the prototype of foo and not resize -2

שאלה 3	(30 נקודות)

סעיף א (5 נקודות)

סעיף ב (5 נקודות)

סעיף ג (10 נקודות)

סעיף ד (10 נקודות)

שאלה 4	**(30 נקודות)**

סעיף א (5 נקודות)

סעיף ב (5 נקודות)