13331231 孙圣 Part5

Set10

1. Where is the isValid method specified? Which classes provide an implementation of this method?

   It is specified in Grid and class BoundedGrid provides an implementatio.

2. Which AbstractGrid methods call the isValid method? Why don't the other methods need to call it?

   getValidAdjacentLocations();
   Because for *getEmptyAdjacentLocations(); getOccupiedAdjacentLocations(); getNeighbors();* they call g*etValidAdjacentLocations();* , so don't have need to call *isValid();*

3. Which methods of the Grid interface are called in the getNeighbors method? Which classes provide implementations of these methods?

   getOccupiedAdjacentLocations(); get();
   getOccupiedAdjacentLocations(); is implemented in AbstractGrid.
   get(); is implemented both in BoundedGrid and UnboundedGrid.

4. Why must the get method, which returns an object of type E, be used in the getEmptyAdjacentLocations method when this method returns locations, not objects of type E?

   If get method is not used, we cannot know whether the location is empty of not, so we are not able to return locations correctly.

5. What would be the effect of replacing the constant Location.HALF_RIGHT with Location.RIGHT in the two places where it occurs in the getValidAdjacentLocations method?

   It would return the north, east, south and west locations, but those in the diagonal are not returned.

Set11

1. What ensures that a grid has at least one valid location?

   In the constructor, if rows and cols are less than or equal to zero, it will throw an exception, so there is at least one valid location in the grid.

2. How is the number of columns in the grid determined by the getNumCols method? What assumption about the grid makes this possible?

   return occupantArray[0].length;

   There is at least one row in the grid and the number of locations in each row is the same.

3. What are the requirements for a Location to be valid in a BoundedGrid?

   Row number and column number must be greater than or equal to zero and smaller to their maximum respectively.

   In the next four questions, let r = number of rows, c = number of columns,

   and n = number of occupied locations.

4. What type is returned by the getOccupiedLocations method? What is the time complexity (Big-Oh) for this method?

   ArrayList<Location>.

   O(cr).

5. What type is returned by the get method? What parameter is needed? What is the time complexity (Big-Oh) for this method?

   Generic type E.

   O(1).

6. What conditions may cause an exception to be thrown by the put method? What is the time complexity (Big-Oh) for this method?

   The location is not valid and the object is null.

   O(1).

7. What type is returned by the remove method? What happens when an attempt is made to remove an item from an empty location? What is the time complexity (Big-Oh) for this method?

   Generic type E.

   Nothing happens because in the occupantArray, before the operation, null is stored, and after the operation, null is still stored there without any change.

   O(1).

8. Based on the answers to questions 4, 5, 6, and 7, would you consider this an efficient implementation? Justify your answer.

   Yes. Methods like get, put, remove are all O(1), and they cannot be more efficient. But *getOccupiedLocations();* requires to loop through all the locations in the grid, so it is a little bit slow.

Set 12

1. Which method must the Location class implement so that an instance of HashMap can be used for the map? What would be required of the Location class if a TreeMap were used instead? Does Location satisfy these requirements?

   Location must override *equals(); and hashCode();*

   compareTo();

   Yes, these three methods are specified in Location.

2. Why are the checks for null included in the get, put, and remove methods? Why are no such checks included in the corresponding methods for the BoundedGrid?

   Because here we cannot use *isValid();* method to check the location since it always returns true, but we need to ensure that the location is not null, otherwise we cannot finish the task. For boundedGrid, before the operation, each method calls *isValid();* to check whether the location is valid or not, which guarantees successful operations.

3. What is the average time complexity (Big-Oh) for the three methods: get, put, and remove? What would it be if a TreeMap were used instead of a HashMap?

   HashMap O(1), O(1), O(1).

   TreeMap O(logn), O(logn), O(logn), where n is the number of locations that are occupied.

4. How would the behavior of this class differ, aside from time complexity, if a TreeMap were used instead of a HashMap?

   Not much difference, because they both act as a map which helps find actors in it, so it does not interferes with actors' behavior.

   There is still a minor difference: when returning some actors, the order of them is not necessarily the same since they use different techniques to store those actors.

5. Could a map implementation be used for a bounded grid? What advantage, if any, would the two-dimensional array implementation that is used by the BoundedGrid class have over a map implementation?

   Yes. If the grid is quite dense, not so sparse, then BoundedGrid class is better. Besides, the array implementation is relatively easier to use than the map implementation because it does not need to care about methods like *equals(); and hashCode();*

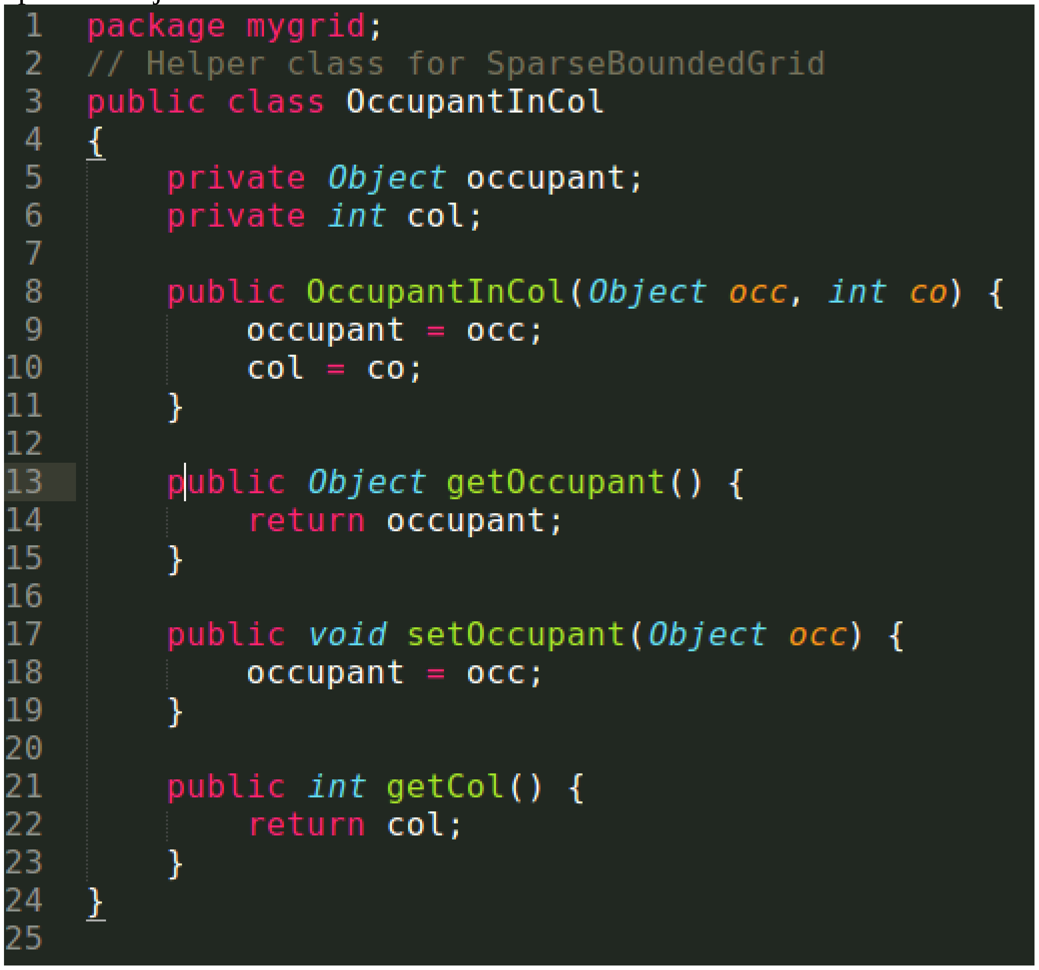
Exercises

1.

Why is this a more time-efficient implementation than BoundedGrid?

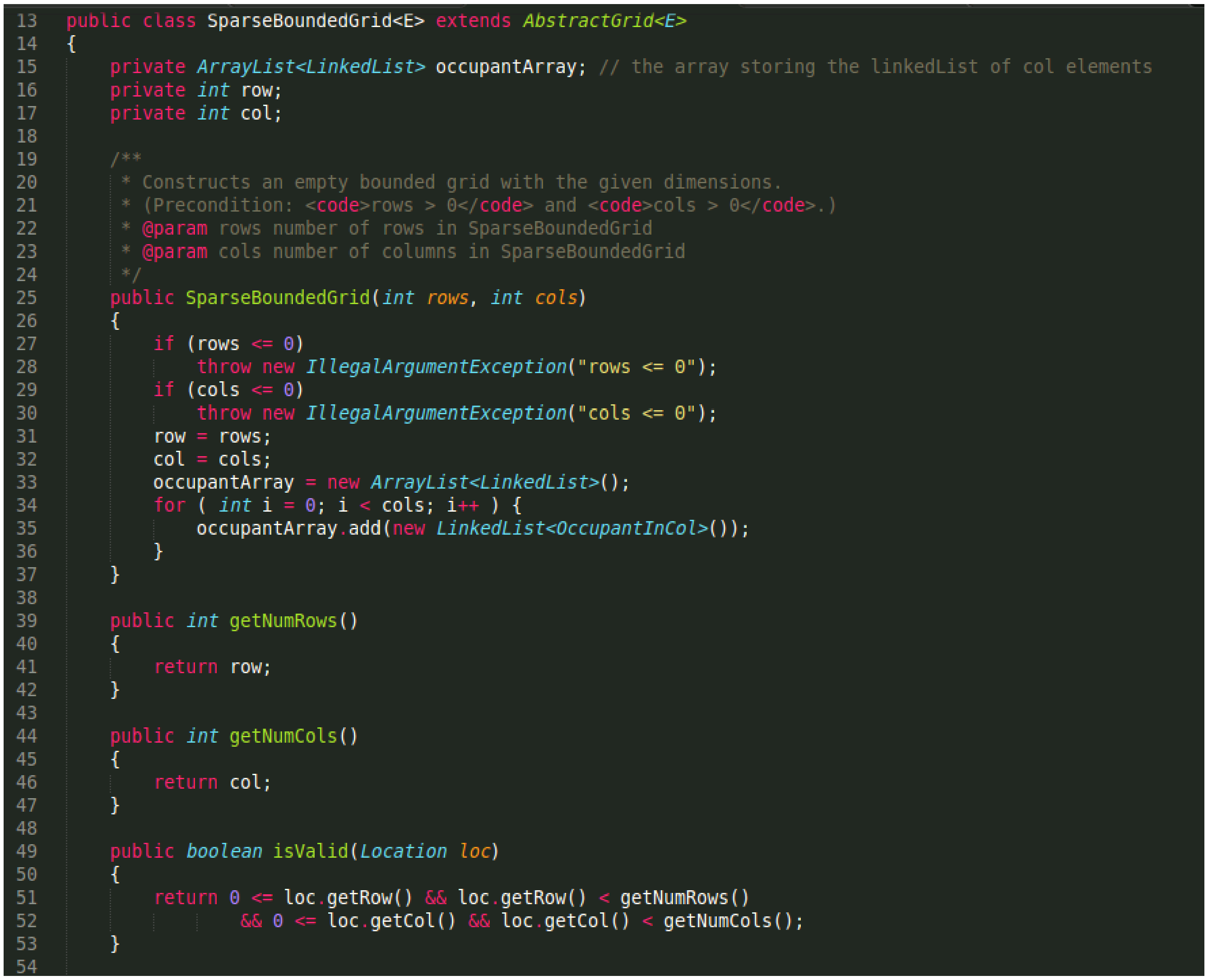
Because the time complexity for getOccupiedLocations(); is O(n) where n is the number of locations that are occupied. Since it only loop through the locations that are occupied instead of all the r * c locations as in BoundedGrid, it is more time-efficient.

OccupantInCol.java

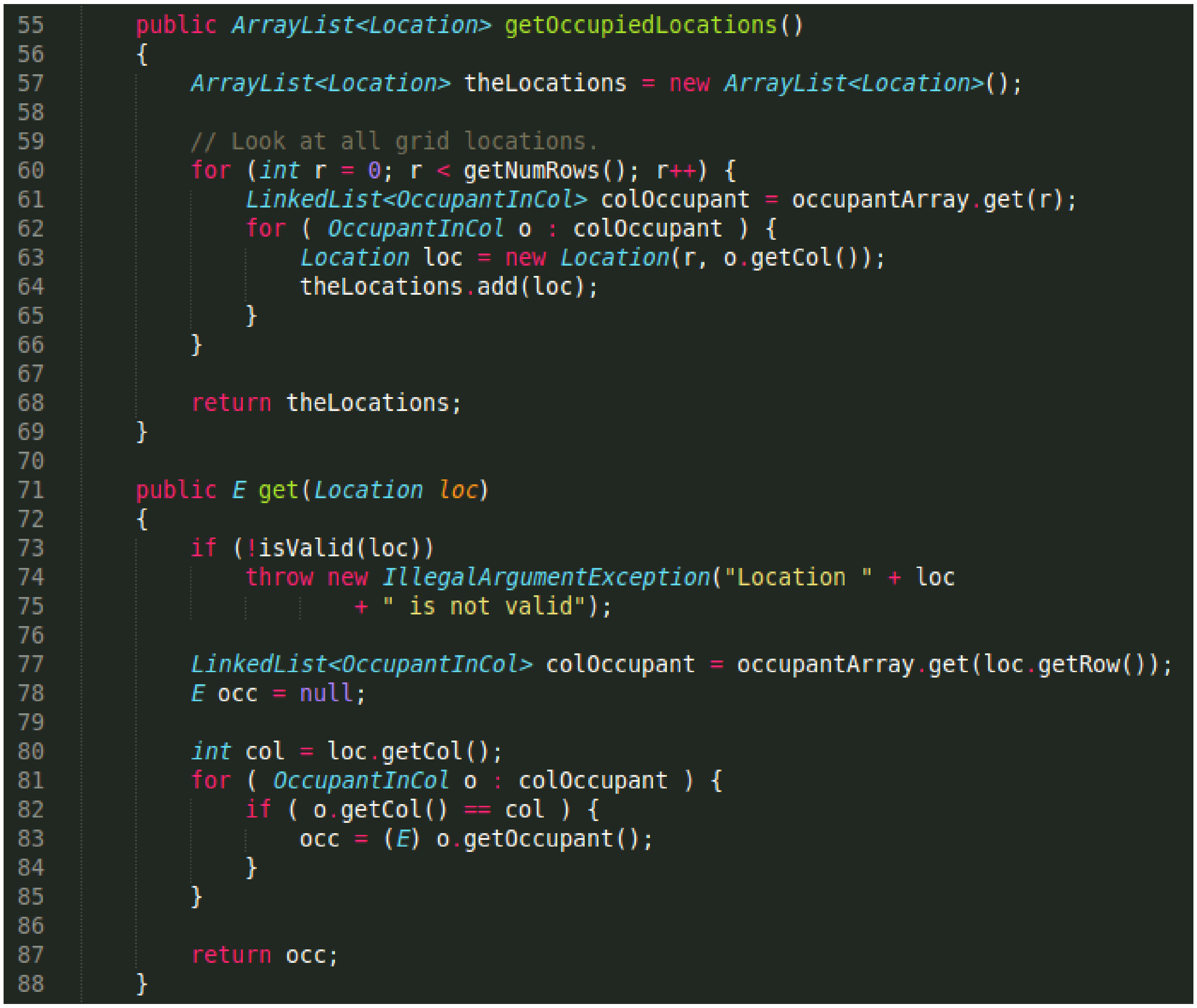


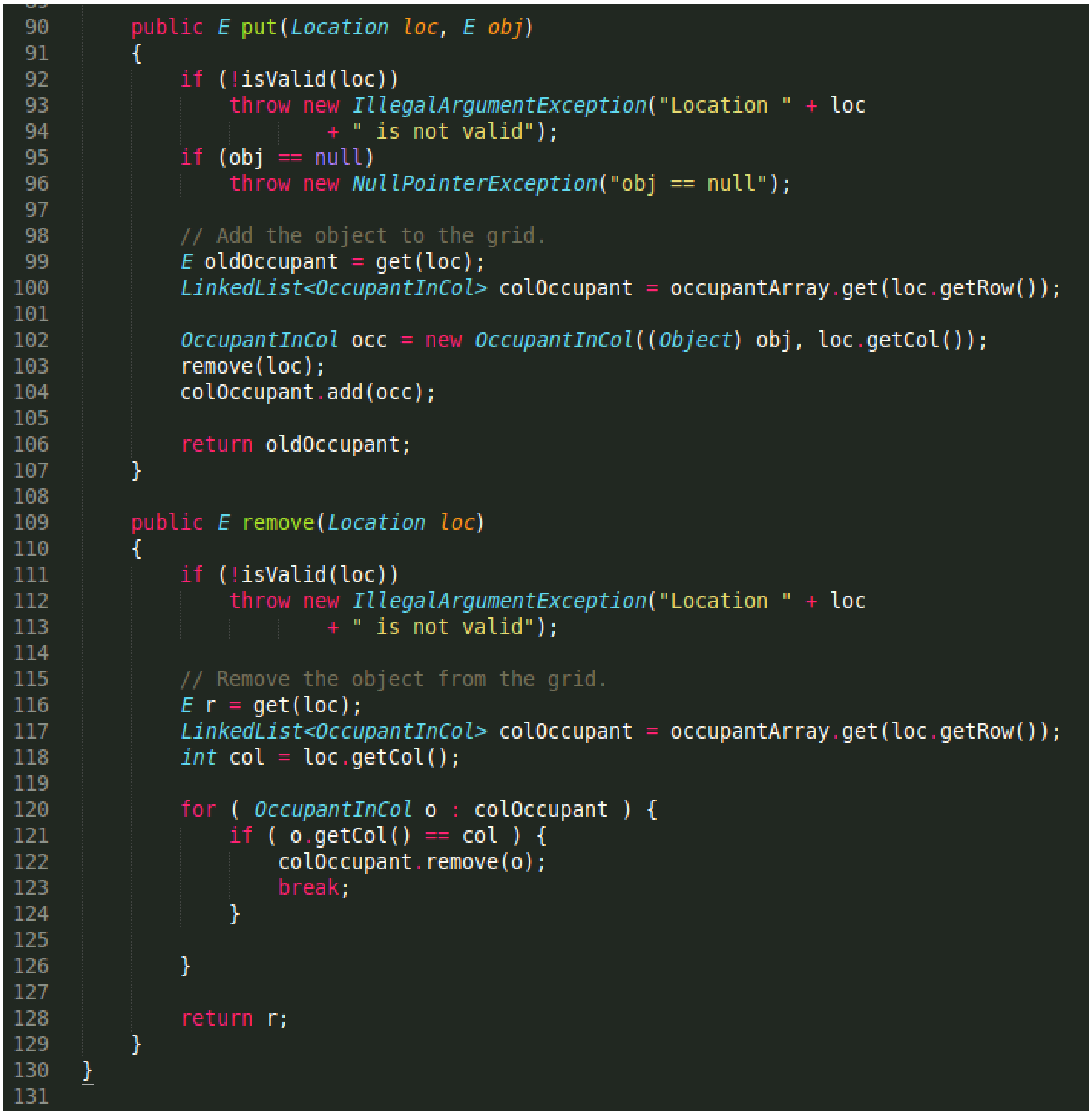
```java
package mygrid;
// Helper class for SparseBoundedGrid
public class OccupantInCol
{
    private Object occupant;
    private int col;

    public OccupantInCol(Object occ, int co) {
        occupant = occ;
        col = co;
    }

    public Object getOccupant() {
        return occupant;
    }

    public void setOccupant(Object occ) {
        occupant = occ;
    }

    public int getCol() {
        return col;
    }
}
```

SparseBoundedGrid.java



```java
public class SparseBoundedGrid<E> extends AbstractGrid<E>
{
    private ArrayList<LinkedList> occupantArray; // the array storing the linkedList of col elements
    private int row;
    private int col;

    /**
     * Constructs an empty bounded grid with the given dimensions.
     * (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.)
     * @param rows number of rows in SparseBoundedGrid
     * @param cols number of columns in SparseBoundedGrid
     */
    public SparseBoundedGrid(int rows, int cols)
    {
        if (rows <= 0)
            throw new IllegalArgumentException("rows <= 0");
        if (cols <= 0)
            throw new IllegalArgumentException("cols <= 0");
        row = rows;
        col = cols;
        occupantArray = new ArrayList<LinkedList>();
        for ( int i = 0; i < cols; i++ ) {
            occupantArray.add(new LinkedList<OccupantInCol>());
        }
    }

    public int getNumRows()
    {
        return row;
    }

    public int getNumCols()
    {
        return col;
    }

    public boolean isValid(Location loc)
    {
        return 0 <= loc.getRow() && loc.getRow() < getNumRows()
                && 0 <= loc.getCol() && loc.getCol() < getNumCols();
    }

```


```java
    public ArrayList<Location> getOccupiedLocations()
    {
        ArrayList<Location> theLocations = new ArrayList<Location>();

        // Look at all grid locations.
        for (int r = 0; r < getNumRows(); r++) {
            LinkedList<OccupantInCol> colOccupant = occupantArray.get(r);
            for ( OccupantInCol o : colOccupant ) {
                Location loc = new Location(r, o.getCol());
                theLocations.add(loc);
            }
        }

        return theLocations;
    }

    public E get(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        LinkedList<OccupantInCol> colOccupant = occupantArray.get(loc.getRow());
        E occ = null;

        int col = loc.getCol();
        for ( OccupantInCol o : colOccupant ) {
            if ( o.getCol() == col ) {
                occ = (E) o.getOccupant();
            }
        }

        return occ;
    }
```



```java
    public E put(Location loc, E obj)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");
        if (obj == null)
            throw new NullPointerException("obj == null");

        // Add the object to the grid.
        E oldOccupant = get(loc);
        LinkedList<OccupantInCol> colOccupant = occupantArray.get(loc.getRow());

        OccupantInCol occ = new OccupantInCol((Object) obj, loc.getCol());
        remove(loc);
        colOccupant.add(occ);

        return oldOccupant;
    }

    public E remove(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        // Remove the object from the grid.
        E r = get(loc);
        LinkedList<OccupantInCol> colOccupant = occupantArray.get(loc.getRow());
        int col = loc.getCol();

        for ( OccupantInCol o : colOccupant ) {
            if ( o.getCol() == col ) {
                colOccupant.remove(o);
                break;
            }

        }

        return r;
    }
}
```

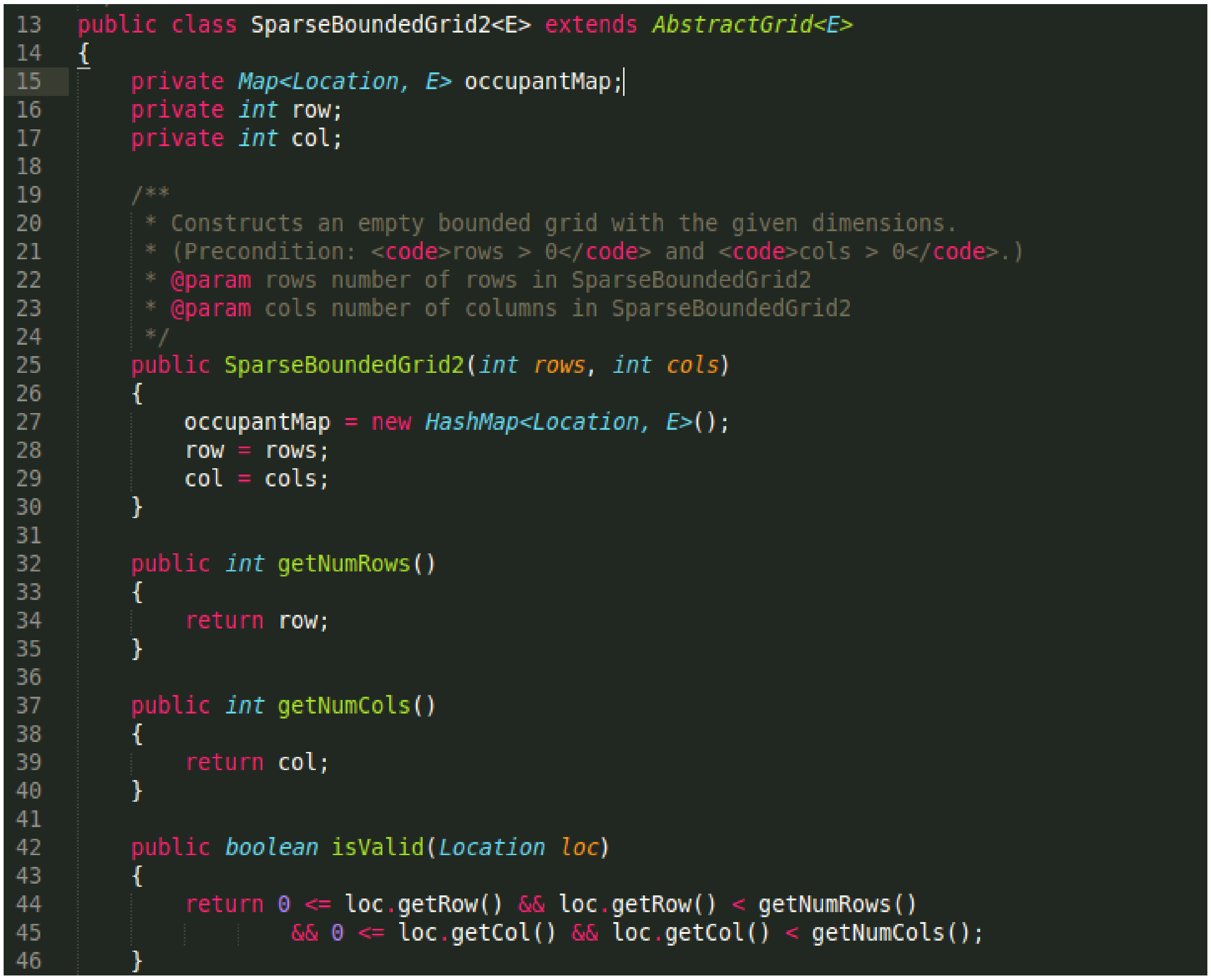
2. How could you use the UnboundedGrid class to accomplish this task? Which methods of UnboundedGrid could be used without change?

I use the same private variable: *private Map<Location, E> occupantMap;*
Method *getOccupiedLocations(); get(); put(); and remove();* can be used without change.

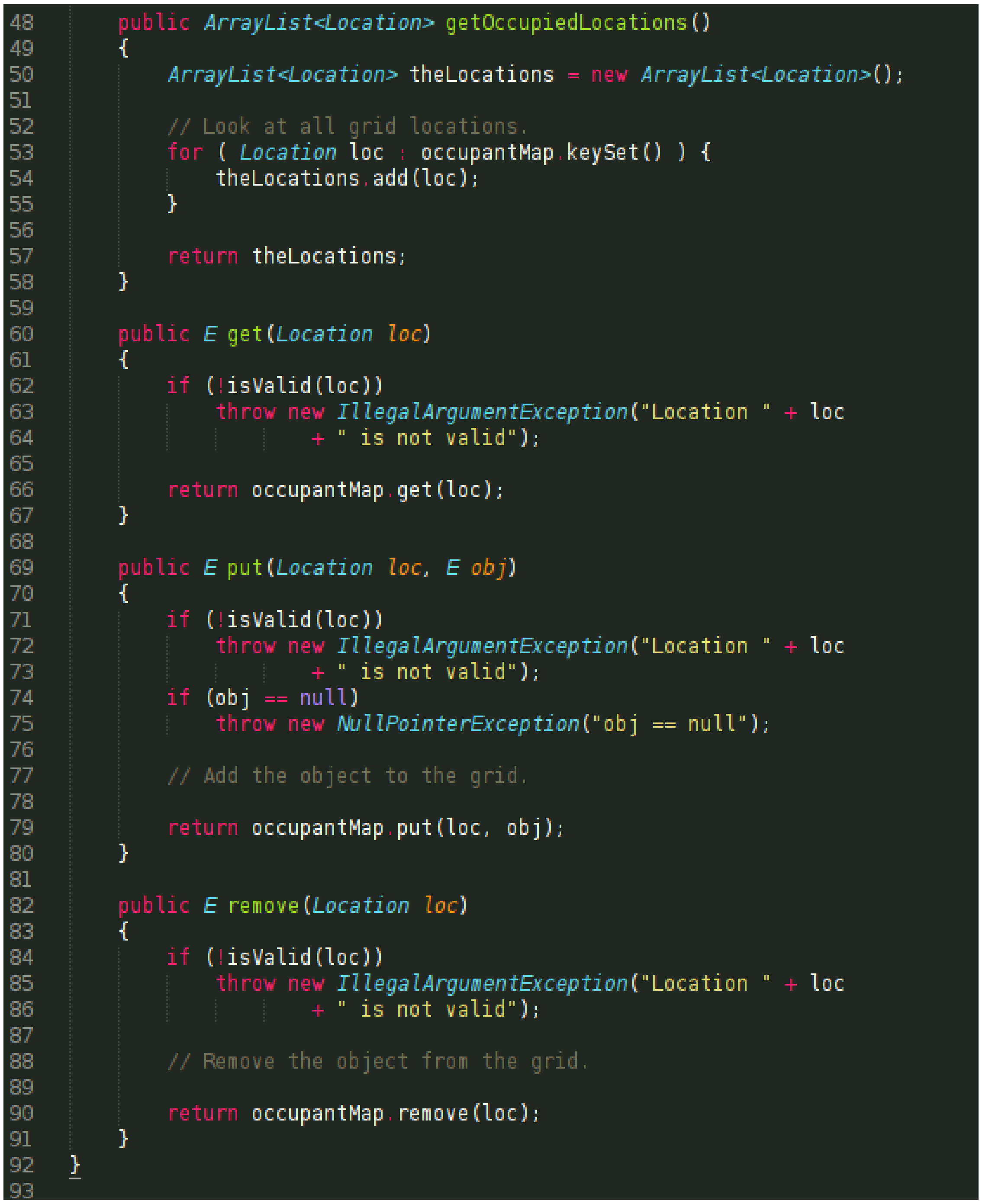
For the following, n is the number of locations that are occupied.

Methods	SparseGridNode version	LinkedList<OccupantInCol> version	HashMap version	TreeMap version
getNeighbors	O(c)	O(c)	O(1)	O(logn)
getEmptyAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedLocations	O(n)	O(n)	O(n)	O(n)
get	O(c)	O(c)	O(1)	O(logn)
put	O(c)	O(c)	O(1)	O(logn)
remove	O(c)	O(c)	O(1)	O(logn)

SparseBoundedGrid2.java



```java
public class SparseBoundedGrid2<E> extends AbstractGrid<E>
{
    private Map<Location, E> occupantMap;
    private int row;
    private int col;

    /**
     * Constructs an empty bounded grid with the given dimensions.
     * (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.)
     * @param rows number of rows in SparseBoundedGrid2
     * @param cols number of columns in SparseBoundedGrid2
     */
    public SparseBoundedGrid2(int rows, int cols)
    {
        occupantMap = new HashMap<Location, E>();
        row = rows;
        col = cols;
    }

    public int getNumRows()
    {
        return row;
    }

    public int getNumCols()
    {
        return col;
    }

    public boolean isValid(Location loc)
    {
        return 0 <= loc.getRow() && loc.getRow() < getNumRows()
                && 0 <= loc.getCol() && loc.getCol() < getNumCols();
    }
```



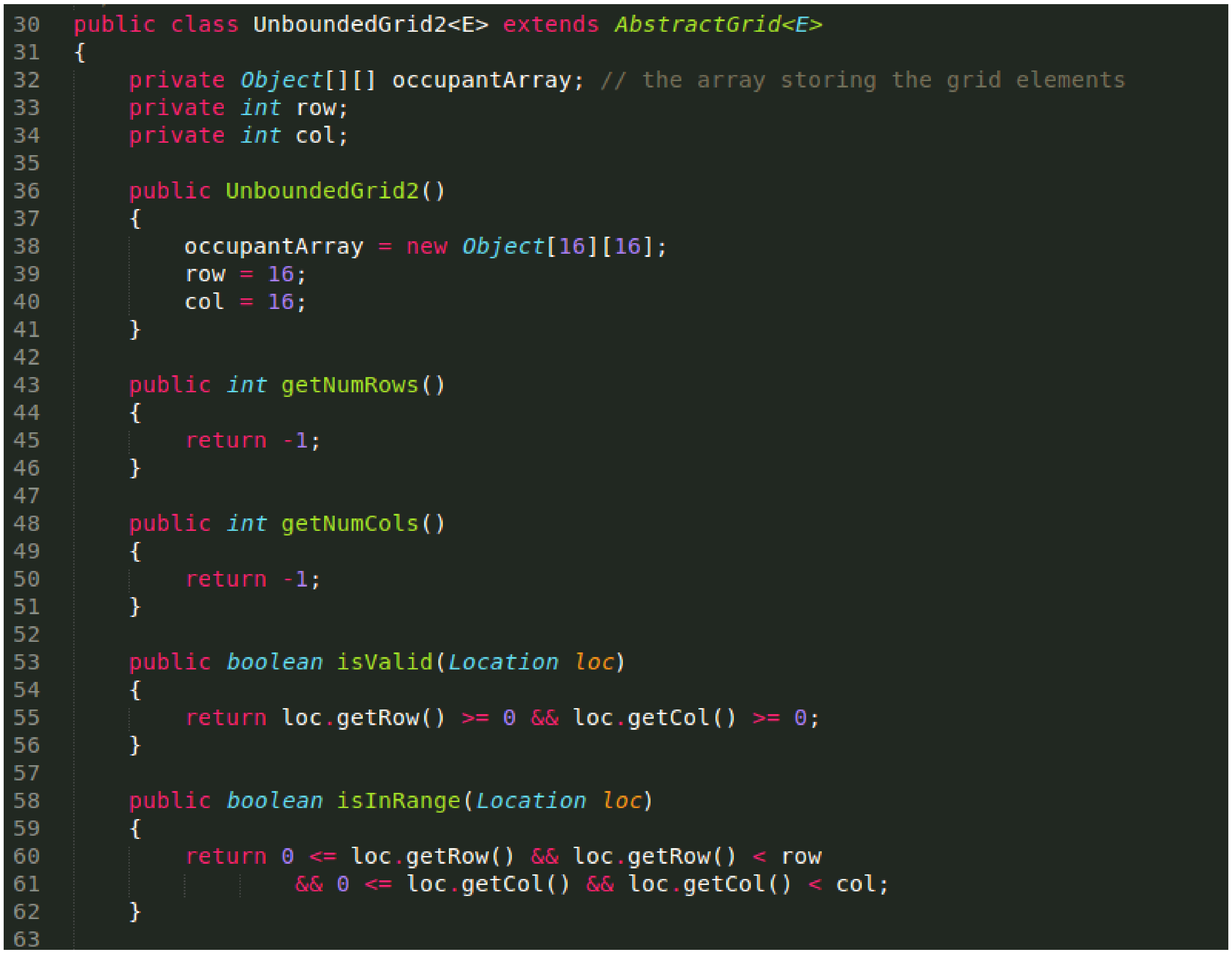
```java
    public ArrayList<Location> getOccupiedLocations()
    {
        ArrayList<Location> theLocations = new ArrayList<Location>();

        // Look at all grid locations.
        for ( Location loc : occupantMap.keySet() ) {
            theLocations.add(loc);
        }

        return theLocations;
    }

    public E get(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        return occupantMap.get(loc);
    }

    public E put(Location loc, E obj)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");
        if (obj == null)
            throw new NullPointerException("obj == null");

        // Add the object to the grid.

        return occupantMap.put(loc, obj);
    }

    public E remove(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        // Remove the object from the grid.

        return occupantMap.remove(loc);
    }
}
```

3.

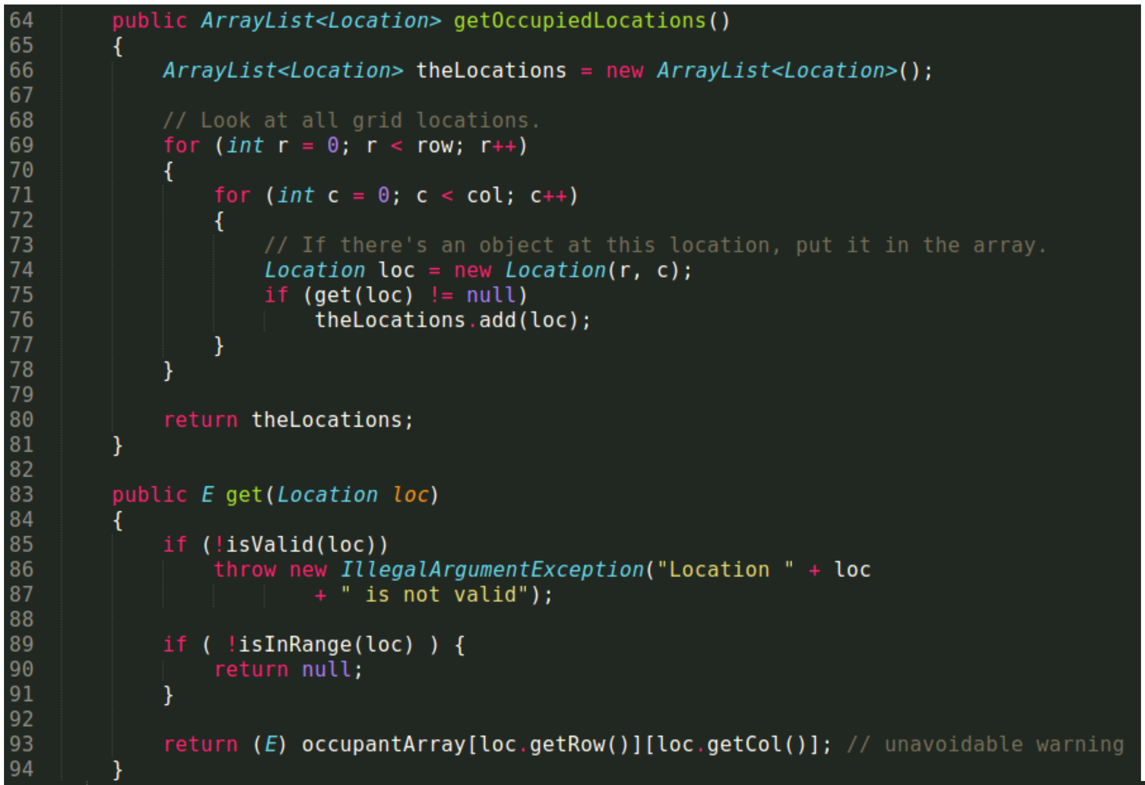
What is the Big-Oh efficiency of the get method? What is the efficiency of the put method when the row and column index values are within the current array bounds? What is the efficiency when the array needs to be resized?

O(1).
O(1).
O(n ^ 2).

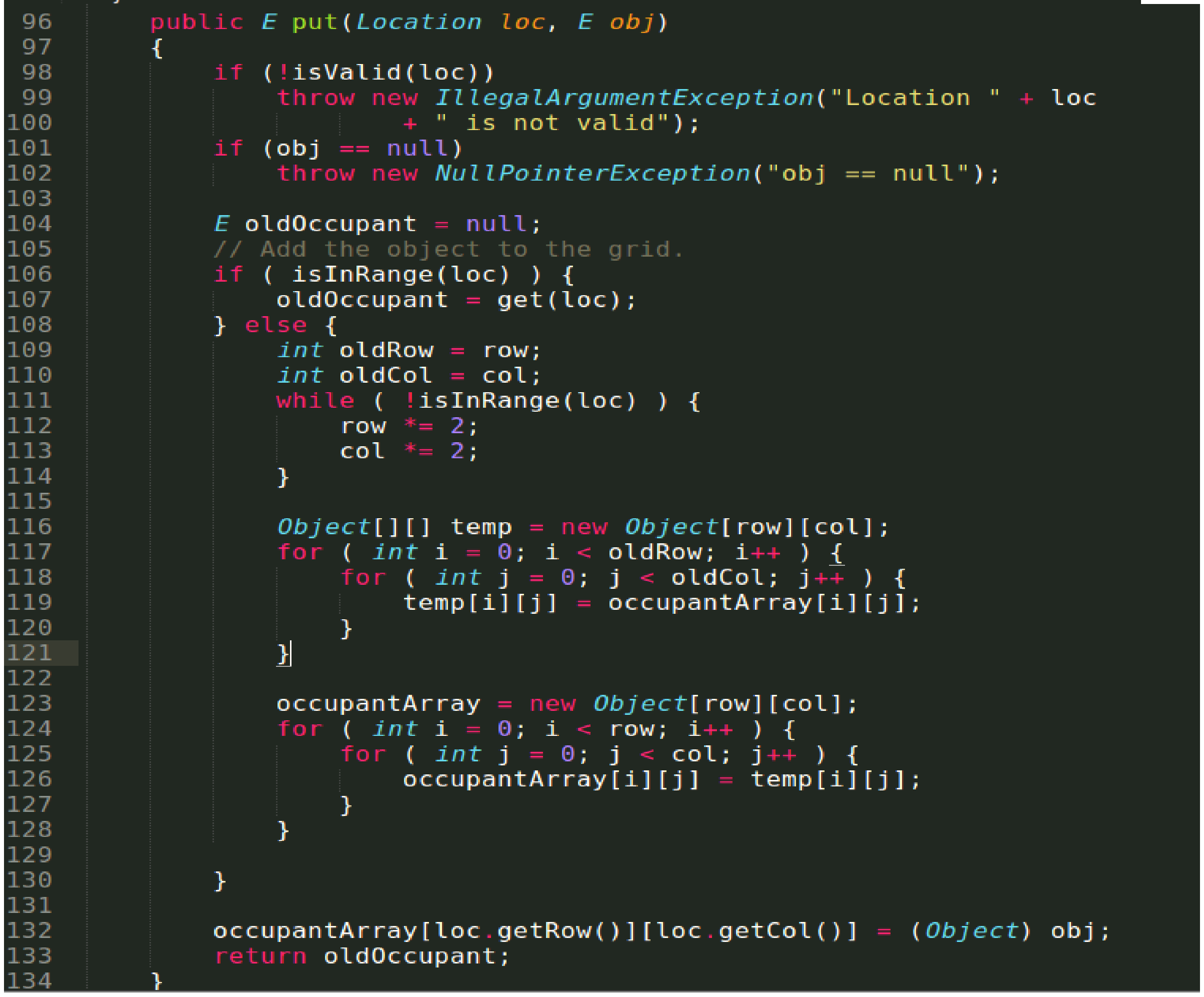
UnboundedGrid2.java



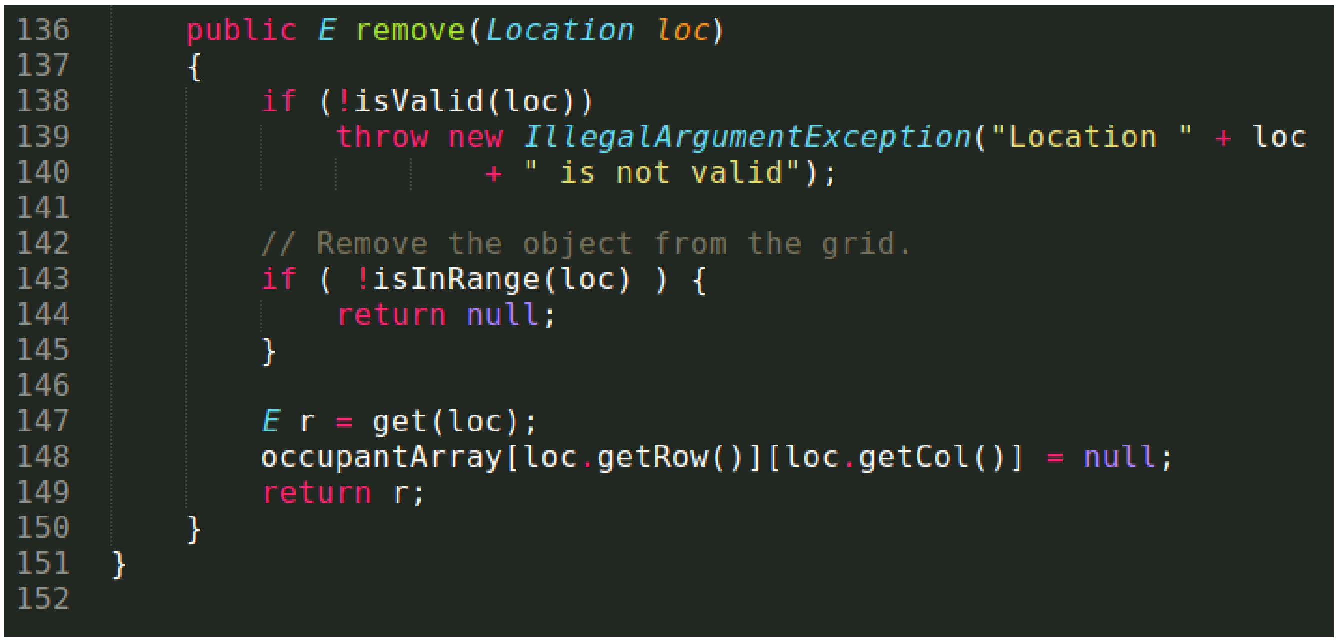
```java
public class UnboundedGrid2<E> extends AbstractGrid<E>
{
    private Object[][] occupantArray; // the array storing the grid elements
    private int row;
    private int col;

    public UnboundedGrid2()
    {
        occupantArray = new Object[16][16];
        row = 16;
        col = 16;
    }

    public int getNumRows()
    {
        return -1;
    }

    public int getNumCols()
    {
        return -1;
    }

    public boolean isValid(Location loc)
    {
        return loc.getRow() >= 0 && loc.getCol() >= 0;
    }

    public boolean isInRange(Location loc)
    {
        return 0 <= loc.getRow() && loc.getRow() < row
                && 0 <= loc.getCol() && loc.getCol() < col;
    }

```



```java
    public ArrayList<Location> getOccupiedLocations()
    {
        ArrayList<Location> theLocations = new ArrayList<Location>();

        // Look at all grid locations.
        for (int r = 0; r < row; r++)
        {
            for (int c = 0; c < col; c++)
            {
                // If there's an object at this location, put it in the array.
                Location loc = new Location(r, c);
                if (get(loc) != null)
                    theLocations.add(loc);
            }
        }

        return theLocations;
    }

    public E get(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        if ( !isInRange(loc) ) {
            return null;
        }

        return (E) occupantArray[loc.getRow()][loc.getCol()]; // unavoidable warning
    }
```



```java
    public E put(Location loc, E obj)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");
        if (obj == null)
            throw new NullPointerException("obj == null");

        E oldOccupant = null;
        // Add the object to the grid.
        if ( isInRange(loc) ) {
            oldOccupant = get(loc);
        } else {
            int oldRow = row;
            int oldCol = col;
            while ( !isInRange(loc) ) {
                row *= 2;
                col *= 2;
            }

            Object[][] temp = new Object[row][col];
            for ( int i = 0; i < oldRow; i++ ) {
                for ( int j = 0; j < oldCol; j++ ) {
                    temp[i][j] = occupantArray[i][j];
                }
            }

            occupantArray = new Object[row][col];
            for ( int i = 0; i < row; i++ ) {
                for ( int j = 0; j < col; j++ ) {
                    occupantArray[i][j] = temp[i][j];
                }
            }

        }

        occupantArray[loc.getRow()][loc.getCol()] = (Object) obj;
        return oldOccupant;
    }
```



```java
    public E remove(Location loc)
    {
        if (!isValid(loc))
            throw new IllegalArgumentException("Location " + loc
                    + " is not valid");

        // Remove the object from the grid.
        if ( !isInRange(loc) ) {
            return null;
        }

        E r = get(loc);
        occupantArray[loc.getRow()][loc.getCol()] = null;
        return r;
    }
}
```