```java
public class SLLSparseM implements SparseM {
	//EntryNode class
	public static class ENode{
		private int row,col;
		private int val;
		private ENode next;

		public ENode(int r, int c, int v, ENode n){
			row = r;
			col = c;
			val = v;
			next = n;
		}

		//copy constructor
		public ENode(ENode copy){
			this(copy.getRow(),copy.getCol(),copy.getVal(),copy.getNext());
		}

		public int getRow(){
			return row;
		}
		public int getCol(){
			return col;
		}
		public int getVal(){
			return val;
		}
		public ENode getNext(){
			return next;
		}
		public void setVal(int nVal){
			val = nVal;
		}
		public void setNext(ENode n){
			next = n;
		}
	}//End of EntryNode class

	//RowHead class
	public static class RNode{
		private int ridx;
		private int numElem;
		private ENode first;
		private RNode nextRow;


		public RNode(int r,int n, ENode e, RNode nr){
			ridx = r;
			numElem = n;
			first = e;
			nextRow = nr;
		}

		public RNode(int r, ENode f, RNode nr){
			this(r,0,f,nr);
		}

		//copy constructor
		public RNode(RNode copy){
			this(copy.getRidx(),copy.getNumElem(),copy.getFirst(),copy.getNextRow());
		}

		public int getRidx(){ //get row index
			return ridx;
		}
		public int getNumElem(){ // get total number of elements in the row
			return numElem;
		}
		public ENode getFirst(){ // get first element of the row
			return first;
		}
		public RNode getNextRow(){
			return nextRow;
		}
		public void setFirstNext(ENode f){
			first = f;
		}
		public void setNextRow(RNode nr){
			nextRow = nr;
		}

	}//End of RowHead Node class

	private int nrows,ncols;
	private int numElements = 0;
	private RNode firstRow;

	//Singly linked list sparse matrix class
	public SLLSparseM(int nr, int nc){
		if(nr <= 0) nr = 1;
		if(nc <= 0) nc = 1;
		nrows = nr;
		ncols = nc;
		firstRow = null;
	}

	@Override
	public int nrows() { //get number of rows
		return nrows;
	}
	@Override
	public int ncols() { // get number of columns
		return ncols;
	}
	@Override
	public int numElements() { // get number of nonzero entries
		return numElements;
	}
	public RNode firstRow(){ // get first row
		return firstRow;
	}

	@Override
	// get element at (ridx,cidx)
	// if out of bounds, return -1
	public int getElement(int ridx, int cidx) {
		if(outOfBounds(ridx,cidx))
			return -1;

		RNode row = findRow(ridx);
		ENode col = searchRow(row,cidx);
		if((row != null) && (col != null))
			return col.getVal();

		return 0;
	}

	@Override
	// clear element at (ridx,cidx)
	public void clearElement(int ridx, int cidx) {
		if(outOfBounds(ridx,cidx))
			return;

		if(getElement(ridx,cidx) == 0)
			return;

		else {
			RNode row = findRow(ridx);
			ENode col = searchRow(row,cidx);
			int numElemInRow = row.getNumElem();

			if(numElemInRow == 1)
				removeRow(row);
			else
				removeEntry(row,col);
		}
	}

	@Override
	// set the element at (ridx,cidx) to val
	public void setElement(int ridx, int cidx, int val) {
		if(outOfBounds(ridx,cidx))
			return;

		if(val == 0){
			clearElement(ridx,cidx);
			return;
		}

		if (firstRow == null) {
			ENode entry = new ENode(ridx, cidx, val, null);
			insertRow(entry);
		}

		else {
			RNode isRValid = findRow(ridx);
			ENode isCValid = searchRow(isRValid, cidx);

			if ((isRValid != null) && (isCValid != null))
				isCValid.setVal(val);

			else if ((isRValid != null) && (isCValid == null)) {
				ENode entry = new ENode(isRValid.getRidx(), cidx, val, null);
				insertEntry(isRValid, entry);
			} else {
				ENode entry = new ENode(ridx, cidx, val, null);
				insertRow(entry);
			}
		}
	}//setElement

	@Override
	// get all nonzero elements, row by row, col by col
```

```java
// stores the row indices, column indices and values in the three arrays
public void getAllElements(int[] ridx, int[] cidx, int[] val) {
              int counter = 0;
              RNode curRow = firstRow;
              while(curRow != null){
                            ENode curCol = curRow.getFirst();
                            for(int i = 0; i < curRow.getNumElem(); i++){
                                          ridx[counter] = curCol.getRow();
                                          cidx[counter] = curCol.getCol();
                                          val[counter] = curCol.getVal();
                                          curCol = curCol.getNext();
                                          ++counter;
                            }
                            curRow = curRow.getNextRow();
              }
              return;
}

@Override
// adding otherM into the current matrix
public void addition(SparseM otherM) {
              if((otherM.nrows() != nrows) || (otherM.ncols() != ncols))
                            return;

              RNode rptr1 = firstRow;
              RNode prev1 = null;
              RNode rptr2 = ((SLLSparseM) otherM).firstRow();

              if(rptr2 == null)                              // 2nd(otherM) matrix is empty
                            return;

              else if(rptr1 == null)                         // 1st(this) matrix is empty
                            insertRowAfter(rptr1,rptr2);

              else {
                while (rptr1 != null) {
                            if (rptr1.getRidx() < rptr2.getRidx()){            // rowM1 < rowM2
                                          prev1 = rptr1;
                                          rptr1 = rptr1.getNextRow();
                            }

                            else if (rptr1.getRidx() > rptr2.getRidx()) {       //rowM1 > rowM2
                               prev1 = insertRowBefore(prev1,rptr1,rptr2);
                               rptr2 = rptr2.getNextRow();
                               if(rptr2 == null     //M2 reaches the end of the list before the M1
                                          return;
                            }

                            else {                                        //rowM1 = rowM2
                               addNodes(rptr1,rptr2);

                               if(rptr1 == firstRow)
                                          prev1 = rptr1;
                               else
                                   prev1 = prev1.getNextRow();
                                   rptr1 = rptr1.getNextRow();
                                   rptr2 = rptr2.getNextRow();
                                   if(rptr2 == null)       //M2 reaches the end of the list before the M1
                                          return;
                     }
               } // while
               insertRowAfter(prev1,rptr2); //M1 reaches the end of the list before the M2
```

```
        }//else

    }//addition


    //outOfBounds
    private boolean outOfBounds(int r, int c){
            return ((r < 0) || (r >= nrows) || (c < 0) || (c >= ncols));
    }

    //findRow
    private RNode findRow(int r){
            RNode temp = firstRow;
            RNode foundR = null;            //return value

            while(temp != null && (temp.getRidx() <= r)){
                    if(temp.getRidx() == r){
                            foundR = temp;
                            break;
                    }
                    temp = temp.getNextRow();
            }
            return foundR;
    }

    //searchRow
    private ENode searchRow(RNode fRow, int c){
            if(fRow == null) return null;

            ENode temp = fRow.getFirst();
            ENode foundC = null;                //return value

            while(temp != null && (temp.getCol() <= c)){
                    if(temp.getCol() == c){
                            foundC = temp;
                            break;
                    }
                    temp = temp.getNext();
            }
            return foundC;
    }

    //insert entry node
    private void insertEntry(RNode fRow, ENode entry){
            ENode curr = fRow.getFirst();
            ENode prev = null;

            if(entry.getCol() < curr.getCol()){
                    entry.setNext(curr);
                    fRow.setFirstNext(entry);
            }
            else{
                    while((curr != null) && (entry.getCol() > curr.getCol())){
                            prev = curr;
                            curr = curr.getNext();
                    }
                    entry.setNext(curr);
                    prev.setNext(entry);
            }

            (fRow.numElem)++;
            numElements++;
```

```
}

//insert Head of row node
private void insertRow(ENode entry){
		RNode rowHead = new RNode(entry.getRow(),entry,null);
		(rowHead.numElem)++;

		if(firstRow == null)
			firstRow = rowHead;

		else if((rowHead.getRidx() < firstRow.getRidx())){
			rowHead.setNextRow(firstRow);
			firstRow = rowHead;
		}

		else {
			RNode curr = firstRow;
			RNode prev = null;

			while((curr!= null) && (rowHead.getRidx() > curr.getRidx())){
				prev = curr;
				curr = curr.getNextRow();
			}
			rowHead.setNextRow(curr);
			prev.setNextRow(rowHead);
		}

		numElements++;
}

//remove row
private void removeRow(RNode row){
		if(row == firstRow){
			firstRow = firstRow.getNextRow();
			row = null;
		}

		else {
			RNode curr = firstRow;
			RNode prev = null;

			while((curr != null) && (curr.getRidx() != row.getRidx())){
				prev = curr;
				curr = curr.getNextRow();
			}
			prev.setNextRow(curr.getNextRow());
			row = null;
		}

		numElements--;
}

//remove entry node
private void removeEntry(RNode row, ENode entry){
		if(entry == row.getFirst()){
			row.setFirstNext(entry.getNext());
			entry = null;
		}

		else {
			ENode curr = row.getFirst();
			ENode prev = null;
```

```
				while((curr != null) && entry.getCol() != curr.getCol()){
					prev = curr;
					curr = curr.getNext();
				}
				prev.setNext(curr.getNext());
				entry = null;
			}
	(row.numElem)--;
			numElements--;
		}

		private void insertRowAfter(RNode rptr1, RNode rptr2){
			RNode temp = new RNode(rptr2);

			if(rptr1 == null){
				rptr1 = temp;
				firstRow = rptr1;

				while(rptr1 != null){
					numElements += rptr1.getNumElem();
					rptr1 = rptr1.getNextRow();
				}
			}
			else {
				rptr1.setNextRow(temp);

				while(temp != null){
					numElements += temp.getNumElem();
					temp = temp.getNextRow();
				}
			}
		}

		private RNode insertRowBefore(RNode prev1, RNode rptr1, RNode rptr2){
			RNode temp = new RNode(rptr2);

			if(rptr1 == firstRow){
				temp.setNextRow(rptr1);
				firstRow = temp;
			}

			else{
				temp.setNextRow(rptr1);
				prev1.setNextRow(temp);
			}

			numElements += temp.getNumElem();

			return temp;
		}

		private void addNodes(RNode rptr1, RNode rptr2){
			ENode c1 =  rptr1.getFirst();
			ENode c2 = new ENode(rptr2.getFirst());
			ENode prev = null;
			while(c1 != null){

				if(c1.getCol() < c2.getCol()){

					prev = c1;
					c1 = c1.getNext();
```

```
                }//if

                else if(c1.getCol() > c2.getCol()){
                    ENode temp = new ENode(c2);

                    if(c1.getCol() == rptr1.getFirst().getCol()){
                        temp.setNext(c1);
                        rptr1.setFirstNext(temp);
                    }

                    else {
                        temp.setNext(c1);
                        prev.setNext(temp);
                    }
                    (rptr1.numElem)++;
                    numElements++;
                    prev = temp;
                    c2 = c2.getNext();
                    if(c2==null)
                        return;

                }//else if

                else {
                    c1.setVal(c1.getVal()+c2.getVal());
                    prev = c1;
                    c1 = c1.getNext();
                    c2 = c2.getNext();
                    if(c2 == null)
                        return;
                }//else

            }//while
            ENode temp = new ENode(c2);
            prev.setNext(temp);
            while(temp!= null){
                (rptr1.numElem)++;
                numElements++;
                temp=temp.getNext();
            }
        }
}//SSLSparseM
```