

```
1: import java.util.ArrayList;
2: import java.util.Scanner;
3:
4:
5: /**
6:  * @author 1828799
7:  */
8: public class BigInt
9: {
10:     // Variable Declaration
11:     private ArrayList<Integer> _bigInt = new ArrayList<Integer>();
12:
13:
14:     /**
15:      * Takes a string representation of a number to be used as a BigInt
16:      * @param value the value to be used as a BigInt
17:      */
18:     public BigInt(String value)
19:     {
20:         // Construct the ArrayList representation of the value
21:         for (int i = value.length() - 1; i > -1; i--)
22:         {
23:             _bigInt.add((int) (value.charAt(i)));
24:         }
25:
26:     }
27:
28:
29:     /**
30:      * Takes a long representation of a number to be used as a BigInt
31:      * @param value the value to be used as a BigInt
32:      */
33:     public BigInt(long value)
34:     {
35:         // Create the variable to be manipulated to create a BigInt
36:         long theNum = value;
37:
38:         // Construct the ArrayList representation of the value
39:         while (theNum > 0)
40:         {
41:             _bigInt.add((int) (theNum % 10));
42:             theNum = theNum / 10;
43:         }
44:     }
45:
46:     /**
47:      * Overrides the toString method and gives a string representation
48:      * of the number
49:      * @override toString
50:      */
51:     public String toString()
52:     {
53:         // String representation of BigInt
54:         String bigInt = "";
55:
56:         for (int value: _bigInt)
57:         {
58:             bigInt = String.valueOf(value) + bigInt;
59:         }
60:         return bigInt;
61:     }
62:
63:     /**
64:      * Performs addition
65:      * @param number the number to be added
```

```
66:      * @return the sum of the BigInt and the number passed in
67:      */
68:      public BigInt add(BigInt number)
69:      {
70:          int size;
71:          int remainder;
72:
73:          if(_bigInt.size() < (number.toString()).length())
74:          {
75:              size = _bigInt.size();
76:          }
77:          else
78:          {
79:              size = number.toString().length();
80:          }
81:
82:          for (int i = 0; i < size; i++)
83:          {
84:
85:          }
86:
87:          return null;
88:      }
89:
90:
91:      /**
92:       * Performs multiplication
93:       * @param number the number to be multiplied
94:       * @return the product of the two
95:       */
96:      public BigInt multiply(BigInt number)
97:      {
98:          return null;
99:      }
100:
101:
102:      public static void main(String[] args)
103:      {
104:          Scanner in = new Scanner(System.in);
105:
106:          String firstArg;
107:          String operator;
108:          String secondArg;
109:          boolean repeat = true;
110:
111:          while(repeat)
112:          {
113:              System.out.println("Compute: ");
114:              firstArg = in.next();
115:              operator = in.next();
116:              secondArg = in.next();
117:              in.hasNext();
118:
119:
120:
121:          }
122:      }
123: }
```

```
1:
2: import java.util.Vector;
3: /**
4:  *
5:  * @author 1828799
6:  *
7:  */
8: public class Matrix
9: {
10:     private int _height, _width; // size of matrix
11:     private Vector _rows; // vector of row vectors
12:
13:
14:     public Matrix(int h, int w)
15:         // pre: h >= 0, w >= 0
16:         // post: constructs an h row by w column matrix
17:     {
18:         _height = h; // initialize height and width
19:         _width = w;
20:
21:         // allocate a vector of rows
22:         _rows = new Vector(_height);
23:
24:         for (int r = 0; r < _height; r++)
25:         { // each row is allocated and filled with nulls
26:             Vector theRow = new Vector(_width);
27:             _rows.add(theRow);
28:
29:             for (int c = 0; c < _width; c++)
30:             {
31:                 theRow.add(null);
32:             }
33:         }
34:     }
35:
36:     public Object get(int row, int col)
37:         // pre: 0 <= row < height(), 0 <= col < width()
38:         // post: returns object at (row, col)
39:     {
40:         Vector theRow = null;
41:
42:         if(0 <= row && row < _height)
43:         {
44:             if(0 <= col && col < _width)
45:             {
46:                 theRow = (Vector)_rows.get(row);
47:             }
48:         }
49:         return theRow.get(col);
50:     }
51:
52:     public void set(int row, int col, Object value)
53:         // pre: 0 <= row < height(), 0 <= col < width()
54:         // post: changes location (row, col) to value
55:     {
56:         if(0 <= row && row < _height)
57:         {
58:             if(0 <= col && col < _width)
59:             {
60:                 Vector theRow = (Vector)_rows.get(row);
61:                 theRow.set(col, value);
62:             }
63:         }
64:     }
65:
```

```
66:     public void addRow(int r)
67:         // pre: 0 <= row < height()
68:         // post: inserts row of null values to be row r
69:     {
70:         if(0 <= r && r < height())
71:         {
72:             _height++;
73:             Vector theRow = new Vector(_width);
74:
75:             for (int c = 0; c < _width; c++)
76:             {
77:                 theRow.add(null);
78:             }
79:
80:             _rows.add(r, theRow);
81:         }
82:     }
83:
84:     public void addCol(int c)
85:         // pre: 0 <= col < width()
86:         // post: inserts column of null values to be column c
87:     {
88:         if(0 <= c && c < width())
89:         {
90:             _width++;
91:
92:
93:             // Iterate forward through the rows
94:             for (int i = 0; i < _height; i++)
95:             {
96:                 // Adds a column to each row that is null and shifts the elements
97:                 // to the right over by one
98:                 ((Vector) (_rows.get(i))).add(c, null);
99:             }
100:         }
101:     }
102:
103:     public Vector removeRow(int r)
104:         // pre: 0 <= row < height()
105:         // post: removes row r and returns it as a Vector
106:     {
107:         Vector theRow = null;
108:
109:         if(0 <= r && r < _height)
110:         {
111:             _height--;
112:             theRow = (Vector) _rows.remove(r);
113:         }
114:
115:         return theRow;
116:     }
117:
118:     public Vector removeCol(int c)
119:         // pre: 0 <= col < width()
120:         // post: removes column c and returns it as a Vector
121:     {
122:         Vector theRow = null;
123:
124:         if(0 <= c && c < _width)
125:         {
126:             _width--;
127:
128:             // Iterate forward through the rows
129:             for (int i = 0; i < _height; i++)
130:             {
```

```
131:         theRow = (Vector) ((Vector) (_rows.get(i))).remove(c);
132:     }
133: }
134:
135:     return theRow;
136: }
137:
138:     public int width()
139:     // post: returns number of columns in matrix
140:     {
141:         return _width;
142:     }
143:
144:     public int height()
145:     // post: returns number of rows in matrix
146:     {
147:         return _height;
148:     }
149:
150: }
```

.compile.out **Tue Sep 23 23:26:54 2014** **1**

Note: Matrix.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Author: 1828799

1.) 2.2: What are the pre- and postconditions for String's charAt method?

Preconditions: $0 < \text{index} < \text{length of string}$, not null

Postconditions: it returns a char at the given index

2.) 3.4: The implementation of java.util.Vector provides a method setSize. This method explicitly sets the size of the Vector. Why is this useful? Is it possible to set the size of the Vector without using this method?

Yes. You can use a constructor to set the size of a vector because it accepts an initial capacity.