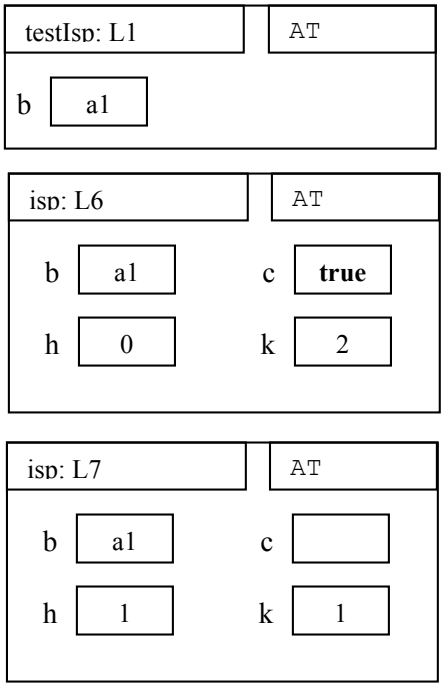
Question 1.



Question 2.

```
/** = Sum of all integer values in obj
    Pre: obj is an object of one of the classes:
      Integer, Integer[], Integer[][], Integer[][][], ….
  */
public static int intDeepSum(Object obj) {
    if (obj instanceof Integer) {
        return ((Integer)obj).intValue();
    }
    Object[] ob= (Object[]) obj;
    int s= 0;
    // inv: s = sum of values in ob[0..k-1]
    for (int k= 0; k < ob.length; k= k+1) {
        s= s + intDeepSum(ob[k]);
    }
    return s;
}
```

Question 3. See final for the specification.

```
public static int fillRow(int[][] b, int r, int c,
                          int num, int  direction, int v) {
   /* inv: The first k values have been placed,
           v+k is the next one to place, and
           it goes in b[r][c]. */
   for (int k= 0; k < num; k= k+1) {
       b[r][c]= v+k;
       if (direction > 0) { r= r+1; c= c+1;}
       else if (direction == 0) { c= c-1; }
         else {  r= r-1; }
     }
     return + num;
   }
```

Question 4.

```
import java.util.*;
/** An instance is a Ring: a Vector in which the
      elements are Integers and they wrap around
      --after the last element comes the first. */
public class Ring extends Vector<Integer> {

   /** Constructor: a Ring with one value: 0 */
   public Ring() { super(); add(0); }

   /** Constructor: ring whose values are the
        digits of n. If n=0, ring contains only 0. */
   public Ring(int n) {
      super();
      if (n > 0)  putIn(this, n);
      else add(0);
   }

   /** = element i of this ring.
      Precondition: i >= 0. */
   public Integer get(int i) {
      return super.get(i%size());
   }

/** Store digits of n in v, with most signifi-
      cant first. If n = 0, v will be empty.
      Precondition. v is not null and is empty. */
   public static void putIn(
                        Vector<Integer> v, int n) {
      Vector<Integer> rev=
                          new Vector<Integer>();
      // Add digits of d, least sig. first, to rev.
      while (n > 0)
         { rev.add(n%10);  n= n/10; }

      // Put the reverse of rev in v.
      int size= rev.size();
      for (int k= 0; k < size; k= k+1)
         { v.add(rev.get(size-1-k)); }
   }
}
```

Question 5.

```
import java.util.*;

/** An instance represents a non-negative repeating
    decimal that is less than 1. */
public class RepeatingDecimal {
   /** The digits of the non-repeating part.
       of the decimal. May be empty. */
   private Vector<Integer> nonrep;

   /** The repeating part of the decimal. Is not empty*/
   private Ring rep;

   /** Constructor: a repeating decimal with
       non-repeating part n and repeating r.
       Precondition. n and r are >= 0.  */
  public RepeatingDecimal(int n, int r) {
```

```
      nonrep= new Vector();
      Ring.putIn(nonrep, n);
      rep= new Ring(r);
   }
   /** Digit number i of this repeating decimal.
      Note: The first digit is number 0.
      Precondition: i >= 0. */
   public int digit(int i) {
      if (i < nonrep.size())
         return nonrep.get(i).intValue();

      i= i - nonrep.size();
      return rep.get(i).intValue();
   }

  /** repr. of this repeating decimal in the form
       non-repeating part [ repeating part ] */
   public String toString() {
      String nr= "" + nonrep;
      nr= nr.substring(1,nr.length()-1) + rep;
      return nr.replaceAll(", ", "");
   }
}
```

Question 6. (a)

```
/** An instance is an Exception */
public class OverflowException
                          extends RuntimeException {
   /** Constructor: instance with empty message. */
   public OverflowException() {
   }

   /** Constructor: an instance with detail message m. */
   public OverflowException(String m) {
      super(m);
   }
}
```

(b)

```
/** An instance maintains a list of
        primes, with duplicates allowed */
public class Primes {
   public static final int MAX= 20; // max no of primes

   private int[] plist= new int[MAX];  //primes are in
   private int n= 0;   // plist[0..n-1. 0 <= n <= MAX.

   /** Throw an IllegalArgumentException with
      suitable message if p is not a prime */
   private static void check(int p) {
      if (p < 2)
         throw new IllegalArgumentException(
                                    p + " is not a prime");
      // throw an exception if some int in 2..p-1 divides p
      // inv: no number in 0..k-1 divides p
      for (int k= 2; k < p; k= k+1) {
         if (p%k == 0)
            throw new IllegalArgumentException(
                                    p + " is not a prime");
      }
   }

   /** Constructor: instance with empty list of primes */
   public Primes() {
      nonrep= new Vector();
      Ring.putIn(nonrep, n);
      rep= new Ring(r);
   }

   /** Add p to the list. Throw an IllegalArgument-
       Exception with suitable message if p is not a prime.
      Throw an OverflowException if no room. */
   public void add(int p) {
      check(p);
      if (n == MAX)
         throw new OverflowException(
                                    "no room for a prime");
      plist[n]= p;
      n= n+1;
   }

   /** If p is a prime, add it to the list; otherwise, print
      "Mistake: p is not a prime".
      Throw an OverflowException if there is no
            room for this prime.*/
   public void addPrime(int p) {
      try {
         add(p);
      } catch (IllegalArgumentException e) {
         System.out.println("Mistake: " + p +
                                 " is not a prime.");
      }
   }
}
```

Question 7. We use formulas instead of pictures for assertions because pictures are too hard to put in here.

```
/**  Precondition: b[p] is some value x, and p <= q.
     Swap the values of b[p..q] and store in j so that:
     Postcondition: b[p..j-1] <= x = b[j] <= b[p+1..q]
     and return the value j. */
public static int partition(int[] b, int p, int q) {
     int j= p;   int t= q;
     // invariant: b[p..j-1] <= x = b[j] <= b[t+1..q]
     while (j < t) {
        if (b[j+1] <= b[j]) {
           int k= b[j+1]; b[j+1]= b[j]; b[j]= k;
           j= j+1;
        } else {
           int k= b[j+1]; b[j+1]= b[t]; b[t]= k;
           t= t-1;
        }
     }
     return j;
  }
```

Question 8. (a) A JFrame uses a BorderLayout manager. Five components can be added to it, on the north, east, west, and south (spells NEWS!) and on the center.

(b) Make a class abstract so that it cannot be instantiated. To make it abstract, put keyword **abstract** before **class**.

(c) Make a method abstract so that it has to be overridden in any subclass (that is not itself abstract). To make it abstract, put **abstract** after **public** (or **private**) and replace the body with a semicolon.