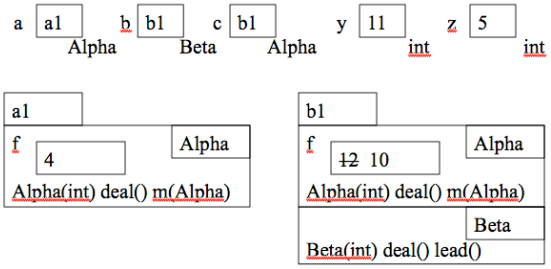


Statement "z= c.lead()" will not compile.

2. /** pre: A diagram that says
b[p..q] is ?
post: A diagram that says
b[p..h-1] < 0
b[h..k-1] = 0
b[k..q] > 0
*/
```
public static void DNF(int[] b, int p, int q) {
    int h= p;
    int k= p;
    int s= q;
    // inv: A diagram that says
         b[p..h-1] < 0
         b[h..k-1] = 0
         b[k..s] is ?
         b[s+1..q] > 0
    while (k <= s) {
        if (b[k] < 0) {
            Swap b[h], b[k]; h= h+1; k= k+1;
        } else if (b[k] > 0) {
            Swap b[s], b[k]; s= s-1;
        } else k= k+1;
    }
}
```

3. /** = as on the exam */
```
public String path() {
    if (in == null) {
        return name + (items == null ? "" : "/");
    }
    // This is not the top-level folder.
    return in.path() + name + (items == null ? "" : "/");
}
```

```
/** = as on the exam   */
public String directory() {
    String res= "\n" + path();
    if (items == null)
        return res;      // Base case: item is a file

    // This is a dir. Append to res all items in directory
    // inv: paths for items 0..k-1 have been appended
    for (int k= 0; k != items.size(); k= k + 1) {
        res=  res + items.get(k).directory();
    }

    return res;
}
```

4.
```
import acm.graphics.*; // it's OK if you forgot this
/** An instance is a Brick */
public class Brick extends GRect {
    /** Constr: a new brick with width w and height h*/
    public Brick(double w, double h) {
        super(w, h);
    }
}
```

(1)
```
// add new brick of width w and height h ...
add(new Brick(w, h));
```

(2)
```
// If g is a brick, remove it from the game
if (g instanceof Brick) {
    remove g;
}
```

5.
```
/** = the level of nesting of parentheses in s.
  Throw an IllegalArgumentException with message
    "unbalanced parens" if  unbalanced parens. */
public static int level(String s) {
    // Store in n the level of nesting of parentheses in s,
    // but throw the exception if unbalanced
    int op= 0;
    int n= 0;
    // inv: s[0..k-1] is balanced except that it has op
    //      '(' with no matching ')'.
    //      n = level of nesting of parentheses in s[0..k-1].
    for (int k= 0; k != s.length(); k= k+1) {
        if (s.charAt(k) == '(') {
            op= op + 1;
            n= Math.max(n, op);
        } else if (s.charAt(k) == ')') {
            if (op == 0)
                throw new IllegalArgumentException
                                    ("Unbalanced parens");
            op= op - 1;
        }
    }

    //post: s[0..s.length()-1] is balanced except that it
    //      has op '(' with no matching ')'.
    //      n = level of nesting of parentheses in s.
    if (op > 0)
        throw new IllegalArgumentException
                                    ("Unbalanced parens");
    return n;
}
```

6. (a) Put field `name` in HDItem, since every file and directory has a name. Put field `items` in class Directory, since only directories and not files have items. Put field `in` in HDItem, since all files and

directories (except for the top one) are in another directory.

(b) Field `items` should be non-static, since each directory can have different contents from other directory instances.

(c)

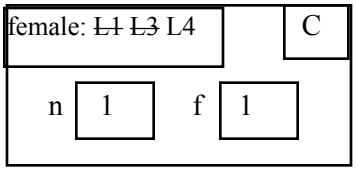
```
/** = "obj is an HDItem with same name as this one"*/
public boolean equals(Object obj) {
      if (!(obj instanceof HDItem))
         return false;
      return ((HDItem)obj).name.equals(name);
}
```

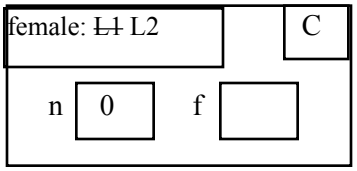
(d) We would declare path in HDItem, and make it abstract, for two reasons: (1) all subclasses would have to declare it. (2) A call v. path() would be legal no matter what the apparent type of v (as long as it is one of the 4, of course). In HDItem, there is no way to know what the complete path is, so it should be made abstract.

(e) The bodies of path:
(1) in Directory: **return** in.path() + name + "/";
(2) in TopDirectory: **return** name + "/";
in File: **return** in.path() + name;

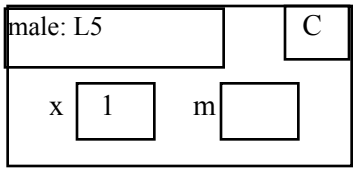
(3)

7.





(Cross this frame out)



8 (a).

```
try {
   x= x/y;
} catch (ArithmeticException e) {
    System.out.println("Whoops, you divided by 0");
}
```

8 (b). 1. Declare the appropriate method in some class C. 2. Indicate syntactically that class C contains the method declaration (usually through the use of an "implements" clause, which we haven't learned too much about. 3. Register an object of class C as a listener for the JFrame.

8 (c).