Your answers may be slightly different! That's OK!

**Question 1**

**(a)**
```
/** see Final for the spec */
public class SalesTransaction extends Transaction {
   /** Constructor: a transaction with the given items.*/
   public SalesTransaction(Item[] items) {
      super(items);
   }

   /** = the total value of this transaction */
   public double total() {
      double subtotal= 0;
      for (int i= 0; i < numItems(); i= i+1)
         subtotal= subtotal + getItem(i).getPrice();
      return subtotal;
   }
}
```

**(b)**
```
/** An instance is an item that can be rented. */
public class RentalItem extends Item {
   /* The cost of renting this item for one day */
   private double dailyRate;

   public RentalItem(String name, double price,
                     double rate) {
      super(name, price);
      dailyRate= rate;
   }

   public double getDailyRate() {
      return dailyRate;
   }
}
```

**Question 2 (a)**
```
public class IRE extends Exception {
   public IRE() {}

   public IRE(String reason) {
      super(reason);
   }
}
```

**(b)**
```
/** An instance represents … */
public class RentalTransaction extends Transaction {

   // This transaction can contain only RentalItems.
   int rentalDate; // The date the rental was given out

   int dueDate; // the date the items are due back

   public RentalTransaction(Item[] items, int date,
            int days) throws IRE {
      super(items);
      rentalDate= date;
      dueDate= date + days;
      for (int i= 0; i < numItems(); i++)
         if (!(getItem(i) instanceof RentalItem))
            throw new IRE(
               "Item '" + getItem(i).getName() +
               "' cannot be rented.");
   }

   public double total() {
      double subtotal= 0;
      int ndays= dueDate – rentalDate;
      for (int i= 0; i < numItems(); i= i+1)
         subtotal= subtotal + ((RentalItem) getItem(i))
               .getDailyRate() * ndays;
      return subtotal;
   }
}
```

**Question 3**
```
int n= s.length()–1;
// pre: s[0..n] is sorted
int i= –1; int t= n+1;
// inv: s[0..i] <= c < s[t..n]
while (i+1 < t) {
   int e= (i+t)/2;
   if (s.charAt(e) <= c) i= e;
   else t= e;
 }
 // post: s[0..i] <= c < s[i+1..n]
```

**Question 4**
```
/** = rotate each row one position to the left, as
       explained on prelim.
   *   Precondition: b is not null, b is possibly ragged,
   *                 each row contains at least one value */
public static void rotate(int[][] b) {
      // invariant: rows 0..r–1 of b have been rotated
      for (int r= 0; r < b.length; r= r+1) {
         // Rotate row r one position to the left;
            int first= b[r][0];
            // inv: b[r][1..c–1] moved to b[r][0..c–2]
            for (int c= 1; c < b[r].length; c= c+1) {
                  b[r][c-1]= b[r][c];
            }
            // post: b[r][1..] has been moved to b[r][0..]
            b[r][b[r].length–1]= first;
      }
      // rows 0..b.length–1 of b has been rotated
}
```

**Question 5.**
```
/** = "This Elephant's ancestry is consistent." */
public boolean isConsistent() {
   if (mom != null) {
      if (mom.isMale() || !isYounger(this, mom) ||
                          !mom.isConsistent())
         return false;
   }
   if (dad != null) {
      if (!dad.isMale() || !isYounger(this, dad) ||
                           !dad.isConsistent())
         return false;
   }
   return true;
}
```

```
/** Throw an IAE if e's ancestry is not consistent.
      Precondition: e is not null. */
public static void verify(Elephant e) {
   if (e.mom != null) {
      if (e.mom.isMale() || !isYounger(e, e.mom))
            throw new IAE();
      verify(e.mom);
      }
  if (e.dad != null) {
      if (!e.dad.isMale() || !isYounger(e, e.dad))
            throw new IAE();
      verify(e.dad);
   }
}
```

**Question 6.**

**(a)** If condition1 is true, execute statement1; otherwise, if condition2 is true, execute statement2.

Another way:

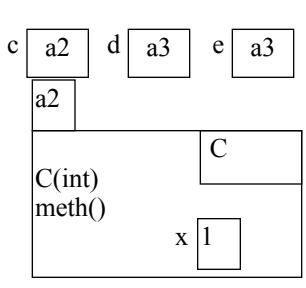
If condition1 is true, execute statement1. If condition1 is false and condition2 is true, execute statement2. But if condition1 and condition2 are both false, don't do anything.

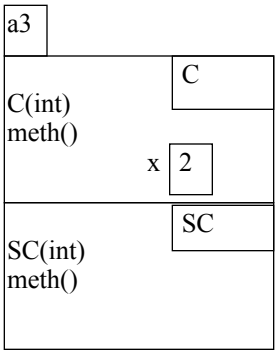
**(b)** b= **new int**[][]{{2, 3, 4}, {3}, {2, 1}}; OR

```
b= new int[3];
b[0]= new int[] {2, 3, 4};
b[1]= new int[] {3};
b[2]= new int[] {2, 1};
```

**(c)**
```
/** = "c is a digit." */
  public static boolean isDigit(char c) {
     try {
          Integer s= Integer.parseInt("" + c);
          return true;
     } catch (NumberFormatException e) {
          return false;
     }
  }
```

**Question 6. (a)**




**(b)** Object d contains two methods called meth(). There is no way to call the one in partition C (unless we rewrite a method in partition SC). To call the one in partition SC, use the call d.meth() or e.meth().

**(c)** The value of the expression is a5: