Final, CS1110, Fall 2011
Name (last name ALL CAPS): **NetID**:

Grades for the final will be posted on the CMS as soon as they are completed. Course grades will be posted next week. You can look at your final when you return in January, not before.

HAVE A GOOD WINTER BREAK!

Submit all regrade requests by 8PM TONIGHT. Use the CMS where possible. Regrades for prelims will not be considered.

You have 2.5 hours to complete the questions in this exam, which are numbered 0..7. Please glance through the whole exam before starting. The exam is worth 100 points.

Question 0 (2 points). Print your name and Cornell **net id** at the top of each page. Please make them legible.

The first few questions concern an inventory and cash register system for a ski and snowboard rental shop. (We have plenty of time to work on this system while we wait for it to snow.)

Classes `Transaction` and `Item` appear to the right. They will be referred to later.

An instance of class `Transaction` represents an order that is paid for at the cash register; among other types, it could be a sale of some items from the shop or a rental transaction lending out some equipment for some number of days.

When answering these questions, read carefully, look at the pertinent class invariants and specifications of methods, and deal with one issue at a time.

Question 0. _____	(out of 02)
Question 1. _____	(out of 15)
Question 2. _____	(out of 15)
Question 3. _____	(out of 13)
Question 4. _____	(out of 13)
Question 5. _____	(out of 14)
Question 6. _____	(out of 14)
Question 7. _____	(out of 14)
Total _____	(out of 100)

```
/** An instance is an item in the store. */
public abstract class Item {
    private String name;  // Description of item
    private double price;  // Selling price
    /** Constructor: An item with given
        name/price.  */
    public Item(String name, int price) { … }
    /* = the name of this item */
    public String getName() { … }
    /* = the price of this item */
    public double getPrice() { … }
}
```

```
/** An instance represents a transaction that is
paid for at the cash register. */
public abstract class Transaction {
    private Item[] items; // items in the transaction.
    /** Constr.: a transaction with the given
        items. */
    public Transaction(Item items[]) { ... }
    /** = no. of items in this transaction */
    public int numItems() {
        return items.length;
    }
    /** = item i, 0 <= i < numItems() */
    public Item getItem(int i) {
        return items[i];
    }
    /** = the total value of this transaction */
    public abstract int total();
}
```

Some operations of Vector v and String s

s.endsWith(s1) = "s ends with String s1"

v.size() = number of elements of v
v.get(i) = element i of vector v
v.add(ob); Add object ob to v
v.clear(); Remove all elements from v
v.contains(ob) = v contains Object ob
v.indexOf(ob) = index of first occurrence of ob in v (-1 if not in)
v.lastIndexOf(ob) = index of last occurrence of ob in v (-1 if not in)

Question 1 (15 points) Abstract classes and subclasses.

(a) **Abstract classes, overriding.** Below (use the back of the previous page if necessary), write a subclass `SalesTransaction` of class `Transaction`. Give it a constructor that takes an array of `Items`, and appropriately override any methods required by class `Transaction`.

```
/** An instance is a transaction in which items are sold. */
public class SalesTransaction extends Transaction {




}
```

(b) **Subclasses.** Below (use the back of the previous page if necessary), write a subclass `RentalItem` of class `Item` (see page 1). While an `Item` represents anything in the store that can be sold, a `RentalItem` represents an item that can also be rented (for instance, a pair of skis that someone might rent to go skiing for the weekend). `RentalItem` should contain the following; make them private or public according to standard practice. You do not have to put in method specifications because they are given here, but do write the class invariant.

- `RentalItem(String name, double price, double dailyRate)`: A constructor for a rental item with the given name and price as well as the given per-day rental rate.
- `double getDailyRate()`: = the cost per day to rent it

```
/** An instance is an item that can be rented. */
public class RentalItem extends Item {




}
```

Question 2 (15 points) Casting, exceptions.

(a) On the back of the previous page, write a class `InvalidRentalException`, whose objects can be thrown. It represents the condition that someone has tried to put an item that cannot be rented into a rental transaction. You determine which class it should extend. You may use the abbreviation "`IRE`".

(b) **Casting, exceptions.** Below (use the back of the previous page if necessary), write a subclass `RentalTransaction` of class `Transaction` (see page 1). *In addition to any components required from class* `Transaction`, `RentalTransaction` should contain the following; make them private or public according to standard practice. You do not have to put in a class invariant or method specifications because they are already given.

- **`int`** `rentalDate` (a field): the day on which this rental was handed out.
- **`int`** `dueDate` (a field): the day on which this rental is due to be returned.
- Constructor `RentalTransaction(Item[] items, int date, int days)`: A constructor for a new transaction containing the given items, to be rented out on `date` for a period of `days` days. Throw an `IRE` (which contains a detail message describing the error) if `items` contains an item that is not a rental item.

Assume that dates are represented by integers giving the number of days since the store opened. Assume that the customer pays the rental fee, which is the daily rate times the number of days, for each item as part of this transaction.

public class RentalTransaction **extends** Transaction {

// Constraints on fields: this transaction can contain only RentalItems.

}

Question 3 (13 points). Algorithms. Assume s and c are declared and contain some values:

```
String s;
char   c;
```

The characters of s are guaranteed to be in their natural ordering (based on character representations). For example, s might contain:

"113AAAZZbbbbbbccd"

Write a binary search, as done in class and on handouts, to search s for the character that is in variable c. You must: (1) draw the precondition (in the space provided below), (2) draw the postcondition, (3) develop an invariant from the pre-and post-condition, and (4) write the loop with initialization using the four loopy questions. No points may be awarded for a loop that "works" but does not use the invariant. Thus, in grading the question, we rely on how well each of the 4 loopy questions is dealt with.

precondition:

postcondition:

invariant:

while ( ) {

}

Question 4 (13 points) Two-dimensional arrays and loops. Someone messed up in writing a method to create certain arrays for us. For example (and this is only an example), they produced the array:

3 1 2		1 2 3
2 1 7 8 5	instead of	1 7 8 5 2
5	the array	5
6 8		8 6

Thus, they put the last value of each row at the beginning instead of the end.

We ask you to write a procedure that fixes this by rotating each row one position to the left —each element is moved one position earlier, and the first element is placed in the last position. The method is declared below; write its body.

Do not use recursion. It will help you to use our methodology for writing loops that process ranges of integers and to write the repetend of each loop as an English statement saying *what* has to be done, before implementing that English statement. You don't have to if you don't want to.

```
/** Rotate each row one position to the left, as explained above.
    Precondition: b is not null, b is possibly ragged, and
                  each row contains at least one value */
public static void rotate(int[][] b) {




}
```

Question 5 (14 points). Recursion.

Consider class Elephant —the only fields and methods necessary for this question are given to the right. Complete the bodies of the methods declared below, each of which checks the ancestry of an Elephant for two kinds of inconsistencies (see the second box to the right).

The first one, a non-static function, returns **true** to indicate a consistent ancestry tree and **false** to indicate an inconsistent ancestry tree The second one, a static procedure, returns normally if the tree is consistent but throws an exception if it is inconsistent —the thrown exception may have an empty message.

These are two separate, independent questions. Do not solve the second one by calling the first or vice versa.

```
/** an instance maintains info
    about an Elephant */
public class Elephant {
  … other stuff …
  Elephant mom;    // mother of this Ele.
                   // (null if unknown)
  Elephant dad;    // father of this Ele.
                   // (null if unknown)
  /** = "this Elephant is male." */
  boolean isMale();
  /** = "e is not null and
         f is not null and
         e is younger than f. */
  static boolean isYounger
     (Elephant e, Elephant f);
}
```

Let e be an Elephant. It's *ancestry tree* consists of e and all its known parents, grandparents, etc.

e's ancestry tree *is consistent* if:

1. All known `mom`'s are female,
2. All known `dad`'s are male,
3. Each elephant in the tree is younger than its known parents.

```
/** = "This Elephant's ancestry tree is consistent." */
public boolean isConsistent() {

}

class IAE extends RuntimeException { // (We shorten InconsistentAncestryException to IAE)
  public IAE() { … }
  public IAE(String reason) { … }
}

/** Throw an IAE if e's ancestry tree is not consistent.
    Precondition: e is not null. */
public static void verify(Elephant e) throws IAE{

}
```

Question 6 (14 points) Miscellaneous.

(a) Explain how to execute the statement:

```
if ( condition1 ) statement1
else if ( condition2 ) statement2
```

(b) Below is a declaration of a variable `b`. Under the declaration, write a sequence of one or more statements that stores in `b` a ragged array that looks as shown to the right.

```
int[][] b;
```

2	3	4
3		
2	1	

(c) We have forgotten how to check whether a character is a digit, and we are in a hurry. We do remember that calling `Integer.parseInt(s)` for a string `s` throws a `NumberFormatException` if `s` is not a valid decimal number. So we (meaning you) write the function below. Fill in the body of the function. You will need a try-statement. You may not call `Character.isDigit` or compare the character to ten different character literals. Note that to use `Integer.parseInt`, you have to create a `String` that consists only of the character `c`. We expect that you know how to do that.

```
/** = "c is a digit." */
public static boolean isDigit(char c) {




}
```

Question 7 (14 points) new-expressions, assignments, and debugging. Consider the two classes at the bottom of this page. When drawing an object, don't draw the partition for class `Object`.

(a) Below is a sequence of three statements. First, draw the variables declared in the statements. Second, execute the sequence, drawing any objects that are created. Do not draw frames for calls. Do not attempt to show us the state of affairs after each statement by drawing the variables again and again. Just execute the sequence of statements, changing the values of variables as execution requires.

```
C c= new C(1);
SC d= new SC(3);
C e= d;
```

The declarations of `c`, `d`, and `e` do NOT appear in classes `C` and `SC`; they are in some other class.

(b) How many methods named `meth` does the object whose name is in d contain? If there is more than one, which ones can be called from a method in some other class? Write statements to call each `meth` that can be called, using variables `c`, `d`, or `e`, below:

(c) Evaluate the new-expression

new SC(5)

As you do it, draw any objects that are created and also draw the frames for any method calls that are executed. Don't erase the frames; just cross them out. Write the value of the expression here: _____.

```
public class C {
   public int x;

   public C (int n) {
     x= (n is even ? 2 : n);
   }

   public int meth() {
     return x / 2;
   }
}
```

```
public class SC extends C {

   public SC(int n) {
     super(2*n);
   }

   public int meth() {
     return x + 3;
   }
}
```