CSE143X Section #15 Problems

1. Define a class Date for storing a calendar date with month, day and year components. Your class should include the following methods:

Date(month, day, year)	constructs a date with given month, day, year
getMonth()	returns the month component
getDay()	returns the day component
getYear()	returns the year component
toString()	returns the date in standard mm/dd/yyyy format

   You may assume that dates are constructed with appropriate values. Months will be between 1 and 12, days will be between 1 and 31 and years will be four-digit numbers. Notice that the toString method is required to produce a standard format with two digits for the month and two digits for the day. For example, January 7th, 2005, should generate the string "01/07/2005".

   In addition, your class should implement the Comparable<E> interface. Dates should be ordered chronologically.

2. Define a class USCurrency that can be used to store a currency amount in dollars and cents (both integers) where one dollar equals 100 cents. Your class should include the following methods:

USCurrency(dollars, cents)	constructs a currency object with given dollars and cents
dollars()	returns the dollars
cents()	returns the cents
toString()	returns a String in standard $d.cc notation (-$d.cc for negative amounts)
add(other)	returns the result of adding another currency amount to this one
subtract(other)	returns the result of subtracting another currency amount from this one

   A currency amount can be negative. The cents method should return values in the range of 0 to 99 for nonnegative currency amounts and should return values in the range of 0 to -99 for negative currency amounts.

   The add and subtract methods should return new USCurrency objects that represent the result of adding or subtracting the other currency amount.

   Note that the toString method should return the amount in a standard format ($d.cc) with two digits for cents and with negative values indicated with a single minus sign in front of the dollar sign (-$d.cc). For example, 4 dollars and 5 cents would be expressed as "$4.05" while -19 dollars and -43 cents would be expressed as "-$19.43".

   In addition, your class should implement the Comparable<E> interface. USCurrency objects should be compared in the obvious way, with smaller currency amounts considered "less" than larger currency amounts (e.g., -$13.45 < -$2.03 < $5.13 < $98.06).

3. Define a class ClockTime that stores information about time of day using a standard clock. Each ClockTime object keeps track of hours, minutes, and a String to indicate "am" or "pm". It has the following public methods:

ClockTime(hours, minutes, amPm)	constructs a ClockTime with given hours, minutes and amPm setting
getHours()	returns the hours
getMinutes()	returns the minutes
getAmPm()	returns the am/pm setting
toString()	returns a string version of the time

Assume that the values passed to your constructor are legal. In particular, hours will be between 1 and 12 inclusive, minutes will be between 0 and 59 inclusive, and the am/pm parameter will be either the String "am" or the String "pm". These values should be returned by the various "get" methods.

The toString method should return a String composed of the hours followed by a colon followed by the minutes (2 digits) followed by a space followed by the am/pm String. For example, given these declarations:

```
ClockTime time1 = new ClockTime(8, 31, "am");
ClockTime time2 = new ClockTime(12, 7, "pm");
```

time1.toString() should return "8:31 am" and time2.toString() should return "12:07 pm". You must exactly reproduce the format of these examples.

Your class should implement the Comparable<E> interface. The earliest time is 12:00 am and the latest time is 11:59 pm. Time increases as it would in a standard clock. Keep in mind that 12:59 am is followed by 1:00 am, that 11:59 am is followed by 12:00 pm, and that 12:59 pm is followed by 1:00 pm.

4. Define a class BookData that keeps track of information for a book and how it is rated by customers (real numbers between 0.0 and 5.0). The class has the following public methods:

BookData(title, author)	constructs a BookData object with the given title and author
review(rating)	records a review for the book with given rating
getTitle()	returns the title of the book
getRating()	returns the average of all ratings (0.0 if none)
toString()	returns a String with title, author, average rating, and number of ratings

Below is an example for a book that has been reviewed four times:

```
BookData book = new BookData("1984", "George Orwell");
book.review(4.7);
book.review(5);
book.review(4.9);
book.review(4.9);
```

After these calls, the call book.getRating() would return 4.875 (the average of the ratings). The toString method should return a string of the form:

```
<book-title>, by <author>, <rating> (<count> reviews)
```

The rating should be truncated to a single digit after the decimal point. For example, given the previous calls, book.toString() would produce:

```
"1984, by George Orwell, 4.8 (4 reviews)"
```

If a book has been reviewed just once, then toString should include the grammatically correct text "1 review" rather than "1 reviews".

The BookData class should implement the Comparable<E> interface. Books that have a higher average rating should be considered "less" than other books so that they appear at the beginning of a sorted list. You should use the complete value of the average rating rather than the truncated value displayed by toString. Books that have the same average rating should be ordered by the number of reviews, with books that have been reviewed more often considered "less" than books that have been reviewed less frequently.