German University in Cairo
Faculty of Media Engineering and Technology
Prof. Dr. Slim Abdennadher

June 13, 2010

Introduction to computer programming Spring term 2010

Final Exam

Bar Code

Instructions: Read carefully before proceeding.

1) Duration of the exam: 3 hours (180 minutes).
2) (Non-programmable) Calculators are allowed.
3) No books or other aids are permitted for this test.
4) This exam booklet contains 15 pages (5 Exercises) including this one. Three extra sheets of scratch paper are attached and have to be kept attached. The exam consists of 6 exercises. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
5) Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem or on the three extra sheets and make an arrow indicating that. **Scratch sheets will not be graded unless an arrow on the problem page indicates that the solution extends to the scratch sheets**.
6) When you are told that time is up, stop working on the test.

Good Luck!

Don't write anything below ;-)

Exercise	1	2	3	4	Bonus	$\sum$
Possible Marks	8	18	24	14	8	64
Final Marks						

Exercise 1 (8 Marks)

Consider the following program.

```
public class MyInteger {
  private int x;
  public MyInteger(int myX) {
     x = myX;
  }
  public double add(int y, int z) {
    System.out.println("add(int, int): " + x + " + " + y + " + " + z);
    return x + y + z;
  }
  public double add(int y, double z) {
    System.out.println("add(int, double): " + x + " + " + y + " + " + z);
    return x + y + z;
  }
  public double add(double y, double z) {
    System.out.println("add(double, double): " + x + " + " + y + " + " + z);
    return x + y + z;
  }
  public static void main(String[] args) {
    MyInteger n1 = new MyInteger(1);
    MyInteger n2 = new MyInteger(2);
    double r1, r2, r3, r4;
    r1 = n1.add(3, 4);
    r2 = n2.add(3, 4);
    r3 = n1.add(5, 6.0);
    r4 = n2.add(7.0, 8.0);
    System.out.println("n1.add(3, 4) == " + r1);
    System.out.println("n2.add(3, 4) == " + r2);
    System.out.println("n3.add(5, 6.0) == " + r3);
    System.out.println("n4.add(7.0, 8.0) == " + r4);
  }
}
```

The program compiles without error and terminates normally when it is executed; what does it display?

Solution:

```
add(int, int): 1 + 3 + 4
add(int, int): 2 + 3 + 4
add(int, double): 1 + 5 + 6.0
add(double, double): 2 + 7.0 + 8.0
n1.add(3, 4) == 8.0
n2.add(3, 4) == 9.0
n3.add(5, 6.0) == 12.0
n4.add(7.0, 8.0) == 17.0
```

Exercise 2 (2+2+3+2+2+3+3+4=18 Marks)

The String class is one of the most useful classes among those provided by the Java Standard Class Library, but how does it actually work?

The purpose of the exercise is to implement a similar class.

Write a class called `MiniString`; like `String` objects, objects that belong to the `MiniString` class represent ordered sequences of characters. Also like String objects, valid character indices in a `MiniString` range from 0 (inclusive) to the number of characters in the `MiniString` (exclusive); that is, the i^{th} character of a `MiniString` has index $i - 1$.

`MiniString` objects MUST keep track of the character sequences they represent using an array (of `char`); you **ARE NOT ALLOWED** to use regular Strings within the `MiniString` class under ANY circumstances.

Your `MiniString` should provide the following public methods:

- A constructor that takes no parameters, and initializes the newly-created `MiniString` so that it represents an empty character sequence, that is, one that contains no characters.
- A second constructor, which takes as a parameter an array of `char`, and initializes the newly created `MiniString` so that it represents the sequence of characters currently contained in the array passed as parameter. The contents of the character array are copied, so that any subsequent modification of the character array does not affect the newly-created `MiniString`
- A method called `length()`, which takes no parameters and returns a value of type `int` representing the length of this `MiniString`, that is, the number of characters it contains.
- A method called `charAt()`, which takes as a parameter a value of type `int`, and returns a value of type `char` representing the character in the `MiniString` at the position given by the parameter.
- A method called `concat()`, which takes as a parameter a `MiniString`, and returns a new `MiniString` representing the concatenation of the `MiniString` this method is called on and the `MiniString` parameter, in that order. For example, if the `MiniString` this method is called on represents the character sequence `"CSEN"`, and the parameter `MiniString` represents the character sequence `"202"`, then the method should return a new `MiniString` representing the character sequence `"CSEN202"`. If the `MiniString` passed as parameter is `null`, then the method MUST return a reference to the `MiniString` on which this method is called.
- A method called `equals()`, which takes as a parameter a `MiniString`, and returns a value of type `boolean`. This method returns `true` if the `MiniString` this method is called on represents the same character sequence as the `MiniString` parameter (that is, both contain the same number of characters in the same order), `false` otherwise. If the `MiniString` passed as parameter is `null`, then the method MUST return `false`.
- A `main` method that creates at least two `MiniString` objects and calls all methods described above.

Solution:

```
public class MiniString {
    char[] list;

    public MiniString() {
    // Creates an empty MiniString.
list = new char[0];
    }

    public MiniString(char[] c) {
    // Creates a MiniString with the same chars as in c
list = new char[c.length];
for(int i = 0; i < c.length; i++)
    list[i] = c[i];
    }
    public int length() {
    //returns the length of the MiniString
return list.length;
    }

    public char charAt(int i) {
    //returns the char at index i within the MiniString
return list[i];
    }

    public MiniString concat(MiniString m) {
    //creates a new String as a result of concatenating
    // this MiniString  and MiniString m and
    // returns a reference to this if m is null
if (m==null)
    return this;
else {
    char[] r = new char[m.list.length + this.list.length];
    for (int i =0; i<list.length; i++)
r[i] = list[i];
    int k = 0;
    for (int j = list.length; j<r.length; j++)
{
    r[j] = m.list[k];
    k++;
}
    return new MiniString(r);
}
    }

public boolean equals(MiniString m) {
//compares two miniStrings and return false if they are not equal
// and true otherwise
if (m == null)
return false;
if (m.list.length != this.list.length)
    return false;
for (int i = 0;i<list.length; i++ )
if (m.list[i]!= this.list[i])
return false;

return true;
}

public static void main (String args []) {
MiniString m = new MiniString();
char [] a = {'S','l','i','m'};
MiniString s = new MiniString(a);
char [] b = {'C','S','E','N','2','0','2'};
MiniString l = new MiniString(b);
System.out.println("m.length() is " + m.length());
System.out.println("the first char in a is "+ s.charAt(0));
MiniString newM = m.concat(l);
System.out.println("newM.length() is "+ newM.length());
System.out.println(l.equals(null));
}
```

```
        }
```

Exercise 3 (2+2+2+2+2+2+3+3+2+2=24 Marks)

in this exercise, we will consider card games that are played with a standard deck of playing cards.

a) First, we would like to implement a `Card` class.

- A `Card` object is defined by its value (`int`) and its suit (`int`). There are four suits:
  - 0 for spades
  - 1 for hearts
  - 2 for diamonds
  - 3 for clubs

  Cards 2 through 10 have their numerical values for their codes.
  - Ace has a value of 1
  - Jack has a value of 11
  - Queen has a value of 12
  - King has a value of 13
- Implement a constructor that takes two parameters the value and the suit of a card.
- Implement a method that return a String representing the card's suit. If the card's suit is invalid, `"??"` is returned.
- Implement a method that returns a String representing the card's value. If the card's value is invalid, `"??"` is returned.
- Implement a `toString` method that returns a String representation of this card, such as `"10 of Hearts"` or `"Queen of Spades"`.

Solution:

```
public class Card {

    public final static int SPADES = 0,       // Codes for the 4 suits.
                            HEARTS = 1,
                            DIAMONDS = 2,
                            CLUBS = 3;

    public final static int ACE = 1,          // Codes for the non-numeric cards.
                            JACK = 11,        //   Cards 2 through 10 have their
                            QUEEN = 12,       //   numerical values for their codes.
                            KING = 13;

    private final int suit;   // The suit of this card, one of the constants
                              //    SPADES, HEARTS, DIAMONDS, CLUBS.

    private final int value;  // The value of this card, from 1 to 11.

    public Card(int theValue, int theSuit) {
            // Construct a card with the specified value and suit.
            // Value must be between 1 and 13.  Suit must be between
            // 0 and 3.  If the parameters are outside these ranges,
            // the constructed card object will be invalid.
        value = theValue;
        suit = theSuit;
    }

    public int getSuit() {
            // Return the int that codes for this card's suit.
        return suit;
    }
    public int getValue() {
            // Return the int that codes for this card's value.
        return value;
    }

    public String getSuitAsString() {
            // Return a String representing the card's suit.
            // (If the card's suit is invalid, "??" is returned.)
        switch ( suit ) {
           case SPADES:   return "Spades";
           case HEARTS:   return "Hearts";
           case DIAMONDS: return "Diamonds";
           case CLUBS:    return "Clubs";
           default:       return "??";
        }
    }

    public String getValueAsString() {
            // Return a String representing the card's value.
            // If the card's value is invalid, "??" is returned.
        switch ( value ) {
           case 1:   return "Ace";
           case 2:   return "2";
           case 3:   return "3";
           case 4:   return "4";
           case 5:   return "5";
           case 6:   return "6";
           case 7:   return "7";
           case 8:   return "8";
           case 9:   return "9";
           case 10:  return "10";
           case 11:  return "Jack";
           case 12:  return "Queen";
           case 13:  return "King";
           default:  return "??";
        }
    }

    public String toString() {
           // Return a String representation of this card, such as
           // "10 of Hearts" or "Queen of Spades".
        return getValueAsString() + " of " + getSuitAsString();
    }


} // end class Card
```

b) A deck of cards contains 52 cards.

- Design a class `Deck` that consists of 52 cards with additional information about how many cards have been dealt from the deck
- Implement a constructor that create an unshuffled deck of cards. There exist 13 cards from each suit.
- Implement a method that puts all the used cards back into the deck, and shuffle it into a random order **Note:** You can use the `Math.random` method.
- As cards are dealt from the deck, the number of cards left decreases. Implement a method that returns the number of cards that are still left in the deck.

- Implement a method that deals one card from the deck and returns it. **Note:** Handle first the case if there are no cards to be dealt.

Solution:

```
public class Deck {

    private Card[] deck;   // An array of 52 Cards, representing the deck.
    private int cardsUsed; // How many cards have been dealt from the deck.

    public Deck() {
           // Create an unshuffled deck of cards.
       deck = new Card[52];
       int cardCt = 0; // How many cards have been created so far.
       for ( int suit = 0; suit <= 3; suit++ ) {
          for ( int value = 1; value <= 13; value++ ) {
             deck[cardCt] = new Card(value,suit);
             cardCt++;
          }
       }
       cardsUsed = 0;
    }

    public void shuffle() {
          // Put all the used cards back into the deck, and shuffle it into
          // a random order.
        for ( int i = 51; i > 0; i-- ) {
            int rand = (int)(Math.random()*(i+1));
            Card temp = deck[i];
            deck[i] = deck[rand];
            deck[rand] = temp;
        }
        cardsUsed = 0;
    }

    public int cardsLeft() {
          // As cards are dealt from the deck, the number of cards left
          // decreases.  This function returns the number of cards that
          // are still left in the deck.
        return 52 - cardsUsed;
    }

    public Card dealCard() {
          // Deals one card from the deck and returns it.
        if (cardsUsed == 52)
           shuffle();
        cardsUsed++;
        return deck[cardsUsed - 1];
    }

} // end class Deck
```

Exercise 4 (8+6=14 Marks)

a) Implement a Java method in the `Shape` class that takes an integer `n` as input and returns a two dimensional array of characters which is populated with the rows of the following shapes:

For example, for an input `n=6`:

```
******
.*....
..*...
...*..
....*.
******
```

For an input `n = 10`:

```
**********
.*........
..*.......
...*......
....*.....
.....*....
......*...
.......*..
........*.
**********
```

b) Write a `main` method using the command-line argument to call the method above.

For example

```
PROMPT> java Shape 6
******
.*....
..*...
...*..
....*.
******
PROMPT> java Shape -4
Sorry, you entered a negative number.
```

Solution:

```
public static char[][] populate(int n) {
    char[][] pic = new char[n][n];
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++) {
            if ( i == j  ||  i == 0  ||  i == n-1 )
               pic[i][j] = '*';
            else
               pic[i][j] = '.';
         }
    return pic;
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    if (n<0)
 System.out.println("Sorry, you entered a negative number.");
```

```
        else {
          char[][] pic = populate(n);

          for (int i = 0; i < n; i++) {
             for (int j = 0; j < n; j++)
                System.out.print(pic[i][j]);
             System.out.println();
          }
        }
    }
```

Exercise 5 (8 Marks)

Bonus Exercise

The look-and-say sequence is the sequence of integers beginning as follows:

```
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...
```

To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit. For example:

- 1 is read off as "one one" or `11`.
- 11 is read off as "two ones" or `21`.
- 21 is read off as "one two, then one one" or `1211`.
- 1211 is read off as "one one, then one two, then two ones or `111221`.
- 111221 is read off as "three ones, then two twos, then one one" or `312211`.

Write a Java method `lookAndsay` that takes an integer `n` as parameter and prints the first $\mathtt{n}^{th}$ terms of this sequence. **Hint**: use a String to store each term, starting with the String `"1"`.

Solution:

```
public static void lookAndsay(int n) {
    String s = "1";
    for (int j = 1; j < n; j++){
        System.out.print(s + ", ");
        String temp= "";
        char c = s.charAt(0);
        int count = 0;
        for(int i = 0; i < s.length() ; i++) {
            if(c == s.charAt(i))
                count++;
            else {
                temp = temp + count + c;
                c = s.charAt(i);
                count =1;
            }
        }
        temp = temp + count + c;
        s = temp;
    }
    System.out.print(s+ ".");
}
```

Extra Sheet

Extra Sheet

Extra Sheet