The following problems are questions from previous finals. Some of the programming language features may now be different (e.g. we are now using ArrayLists instead of Vectors). Additionally some topics may have been covered in prior years that were not covered this year (e.g. graphics).

1. **Short Answer.** Provide brief (1-3 sentence) answers to the following:

   a) What is the purpose of a stub?

   b) What advantage(s) are gained by using the debugger of an IDE (e.g. NetBeans) vs. using no debugger (e.g. println statements)?

   c) What role does the stack play in declaring local variables?

   d) If s1 and s2 are both String variables, why doesn’t s1==s2 compare if the contents of the two strings are identical?

   e) What is the difference between a reference parameter and a value parameter that is passed to a method?

   f) Under what circumstances might we want to declare a class as abstract?

   g) What is the difference between method overloading and method overriding?
2. Bad Design. The following code will compile and run but it suffers from some design flaws. Identify the flaws, how you would fix them, and why they are considered poor programming techniques. Do not be concerned that the program does not really compute anything very useful.

```java
public class Money {
    public double amount = 0;
    public static int i;
    public static int dollars;
    public static int cents;

    public void setAmount(int dollars, int cents) {
        amount = dollars + ((double) cents / 100);
    }

    public int getDollars() {
        return ((int) amount);
    }

    public int getCents() {
        return (int) (amount*100 - getDollars()*100);
    }

    public static void main(String[] args) {
        // Array of 10 money objects
        Money[] moneyArray = new Money[10];
        for (i=0; i<10; i++) {
            moneyArray[i] = new Money(); // Create each one
            moneyArray[i].amount = i + (double) i/100;
        }
        cents = 0;
        for (i=0; i<10; i++) {
            dollars = (int) moneyArray[i].amount;
            cents += (int) (moneyArray[i].amount*100 - dollars*100);
        }
        System.out.println(cents);
    }
}
```
3. Bugs. Find the bugs in the following snippets of code. You don’t have to fix them, but must identify the bugs.

a)
```
char[] a = new char[3];
```

b)
```
class Foo
{
    int ProcessIt(int x, int y)
    {
        return (x*y);
    }

    float ProcessIt(int x, int y)
    {
        float f;
        f = (float) x * 3.14 * (float) y;
        return(f);
    }
}
```

c)
```
public class Foo {
    public int num;

    public static void main(String[] args)
    {
        Foo a,b;
        a.num = 10;
        b.num = 20;
        a.num = b.num;
    }
}
```
4. **Graphics and Loops.** Draw the output of the following program:

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class WindowDestroyer extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}

class MyWindow extends JFrame
{
    public MyWindow()
    {
        this.setSize(300, 300);
        this.addWindowListener(new WindowDestroyer());
        this.setTitle("Fun Stuff");
        this.setVisible(true);
    }

    public void paint(Graphics g)
    {
        for (int i=0; i<255; i++) {
            g.setColor(new Color(i,0,0));
            g.drawLine(25,i+25,i+25,i+25);
        }
    }
}

class DrawSomething
{
    public static void main(String[] args)
    {
        MyWindow w = new MyWindow();
    }
}
```
5. Using 1-D Arrays.

Write a method named `ReverseArray` that takes as input an array of integers and reverses the order of the values.

Here is a sample context for which `ReverseArray` could be invoked:

```java
public static void main(String[] args) {
    int[] arr = {1, 22, 3, 444, 5};
    ReverseArray(arr);
    for (int i=0; i<arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}
```

This program should output:

```
5  444  3  22  1
```

Your program should work on any array, not just the array in the example above.
6. **Pointer Data Structures.** The code below constructs a singly linked list of four elements:

```java
class Node {
    public int data;
    public Node next;
    public Node(int i) {
        data = i;
        next = null;
    }
}

class LinkedList {
    public static void main(String[] args) {
        Node head = null, newData = null, tail = null;
        int i;
        for (i = 0; i < 4; i++) {
            newData = new Node(100 + i);
            if (head == null) {
                head = newData;
                tail = newData;
            } else {
                tail.next = newData;
                tail = newData;
            }
        }
        // Code for you to write would start here, see next page
    }
}
```

Graphically, the list that is created looks like this:

```
data: 100
next: ———

head

data: 101
next: ———

data: 102
next: ———

data: 103
next: NULL

tail
```

(problem continued on the next page)
Write code that reverses the linked list. The easiest way to do this is to change the “next” pointers around, although it is also possible to swap values among nodes. Your code should work on an arbitrary number of nodes, not just for this case of four nodes. For example, if the original list were reversed, it would result in the following graphical picture:
7. **(11 pts) Vectors and File I/O.** You have given monetary donations to a number of individuals. They are stored in the text file “donations.txt” where the name is stored first followed by the amount of money donated. An “*” for the name signals the end of the file. For example, the file might contain:

```
Nephew George
400.00
Salvation Army
200.00
Kenrick Mock
150000.00
*
```

Write a method that reads this file into a Vector of the following class:

```java
public class Donation
{
    public String sName;
    public double dAmount;
    public Donation(String s, double d) {
        sName = s; dAmount = d;
    }
    public boolean equals(Object otherD) {
        Donation dn = (Donation) otherD;
        return ((sName.equals(dn.sName)) &&
                (dAmount == dn.dAmount));
    }
}
```

Your method should ensure that no duplicate values are inserted into the vector in the event of duplicate entries in the text file. A duplicate occurs if the name and value is identical. Finally, your method should output the sum of the donated money.
8. **Recursion.** Fill in the blanks to complete the following recursive function, which returns true if the String passed in is a palindrome (the same forward as it is backward), and false otherwise. The function is passed the String, a value indicating the leftmost index of the string we’re interested in, and a value indicating the rightmost index of the string we’re interested in.

If the leftmost index equals the rightmost index, this would indicate a string consisting of one character (the character at s.charAt(leftIndex)). If the leftmost index is greater than the rightmost index, this would indicate a string consisting of zero characters. Both a string of one or zero characters is considered to be a palindrome.

```java
public static boolean Palindrome(String s, int leftIndex, int rightIndex) {
    // Base case or termination condition; return true for string 1 or 0 chars long
    if (__________________________) {
        return true;
    }

    // Check ends of the string
    if (s.charAt(leftIndex)!=s.charAt(rightIndex)) {
        return false;
    }
    else {
        return ________________(s, leftIndex +______, ________________);
    }
}
```

Usage samples:
```
b = Palindrome(“wow”,0,2);  // Returns true
b = Palindrome(“goober”,0,5);  // Returns false
b = Palindrome(“z”, 0, 0);  // Returns true
b = Palindrome(“abba”,0,3);  // Returns true
```
9. **Inheritance.** Give the output of the program below.

```java
class Stuff {
    private int x;
    public Stuff() {
        x = 5;
    }
    public Stuff(int i) {
        x = i;
    }
    public int getVal() {
        return x;
    }
}

class MoreStuff extends Stuff {
    private int y;
    public MoreStuff() {
        y = 2;
    }
    public MoreStuff(int i, int j) {
        super(i);
        y = j;
    }
    public void setY(int j) {
        y = j;
    }
    public int getVal() {
        return (super.getVal() * y);
    }
}

class Inherit {
    public static void main(String[] args) {
        Stuff s1 = new Stuff();
        Stuff s2 = new Stuff(4);
        MoreStuff s3 = new MoreStuff();
        MoreStuff s4 = new MoreStuff(6, 7);

        PrintStuff(s1);
        PrintStuff(s2);
        PrintStuff(s3);
        PrintStuff(s4);
        s1 = s3;
        s3.setY(3);
        PrintStuff(s3);
        PrintStuff(s1);
    }
    public static void PrintStuff(Stuff s) {
        System.out.println(s.getVal());
    }
}
```