Introduction to Programming (CS112): Sample

Name: 

Netid: 

- Write your answers directly on the examination paper, including any work that you wish to be considered for partial credit. Use the back side of each sheet if you require more space to answer a question.
- Each question is marked with the number of points assigned to that problem. The total number of points is 50.
- The duration of this exam is three hours.
- The examination is open-notes and open-book.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Topic</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Program Evaluation</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Recursive Program</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Type Conversion</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Variable Scope</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Inheritance</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Simple Objects</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Permutation Checker</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Arrays and Recursion</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Stack Abstraction</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1  Program Evaluation [5 points]

Consider the following method:

```java
public void looper(int num) {
    while (num > 1) {
        System.out.println(num);
        if (num % 2 == 0)
            num = num / 2;
        else
            num = 3*num + 1;
    }
}
```

What is the output from the following method invocations:

- looper(8):
  
  8
  4
  2

- looper(3):
  
  3
  10
  5
  16
  8
  4
  2
Consider the following method:

```java
public static String foo(String s) {
    if (s.length() == 1)
        return s;
    else
        return foo(s.substring(1)) + s.charAt(0);
}
```

- What does `foo("abcd")` evaluate to?
  - `dcba`

- Briefly describe the intended purpose of this method.
  - Reverses the string input.
Problem 3  Type Conversion [5 points]

Given the following class definitions identify whether there are any errors in Statements 1 through 4. Explain briefly why a certain statement is erroneous. Also indicate whether the errors are identifiable by the compiler or do they successfully pass compile-time checks and manifest as run-time errors.

class A {
    public void f() { }
    private void g() { }
}

class B extends A {
    public void h() { }
}

public class Check {
    public static void main(String[] args) {
        A o1 = new A();  // Statement 1
        A o2 = new B();  // Statement 2
        B o3 = (B)o2;    // Statement 3
        B o4 = (B)o1;    // Statement 4
    }
}

Answer: Statement 4 is incorrect. o1 was created as an A object and therefore cannot be converted to a B object. It results in a run-time error.

Are there any errors in the following alternate definition of the Check class? If a certain statement is incorrect, indicate whether the error is identifiable at compile-time or run-time.

public class Check {
    public static void main(String[] args) {
        A o1 = (A)new B();  // Statement 5
        o1.h();            // Statement 6
        B o2 = (B)o1;      // Statement 7
        o2.g();           // Statement 8
    }
}

Answer: Statement 6 is incorrect since an h method is not defined for A objects. This error will be identifiable at compile-time. Statement 8 also results in a compile-time error since g is defined as a private method.
Problem 4  

Variable Scope [3 points]

What is the result of executing the following program?

```java
public class Arcane {
    static int x;
    int y;
    public Arcane(int x) {
        int y = x + 1;
        this.y = y;
    }
    public void increment() {
        y += x;
    }
    public void print() {
        System.out.println(y);
    }
    public static void main(String[] args) {
        x = 5;
        int x = 2;
        Arcane t = new Arcane(x);
        t.increment();
        t.print();
    }
}
```

The program prints out 8.
Problem 5  

Inheritance [4 points]

The **Rectangle** class models a simple rectangle object with position and size attributes. The position and size attributes are declared as private variables inside the class definition:

```java
public class Rectangle {
    private int xcoord, ycoord, xsize, ysize;
    public Rectangle(int x, int y, int xs, int ys)
    {
        xcoord = x;
        ycoord = y;
        xsize = xs;
        ysize = ys;
    }
    public int area()
    {
        return xsize*ysize;
    }
}
```

The following instantiation command illustrates how this class could be used. The command creates a rectangle of size 5 units along the x dimension and 6 units along the y dimension with its top left corner positioned at coordinate (10, 20).

```java
Rectangle rect = new Rectangle(10, 20, 5, 6);
```

Given this **Rectangle** class, your task is to develop a **Square** class that extends the **Rectangle** class. Your solution should provide just a constructor for Square objects and **must not declare any new instance variables**. The signature of the constructor should be:

```java
public Square(int x, int y, int size);
```

A square object of size 5 units along each dimension and positioned at (10, 20) could then be created using: `new Square(10, 20, 5)`.

```java
public class Square extends Rectangle {
    public Square(int x, int y, int size)
    {
        super(x, y, size, size);
    }
}
```
Problem 6  

Simple Objects [5 points]

Enhance the Rectangle class definition from the previous problem to include a method called contains that takes a Point object, checks whether the point is contained inside the rectangle, and returns a boolean value of true/false. Here is the code for the Point class:

```java
public class Point {
    public int xcoord, ycoord;
    public Point(int x, int y) {
        xcoord = x;
        ycoord = y;
    }
}
```

and here is how a main method would check for containment:

```java
public static void main(String[] args) {
    Rectangle rt = new Rectangle(10, 20, 5, 6);
    Point pt1 = new Point(12, 22);
    Point pt2 = new Point(12, 18);
    System.out.println(rt.contains(pt1)); // prints true
    System.out.println(rt.contains(pt2)); // prints false
}
```

Fill in the code for the contains method such that it is consistent with the above description.

```java
public class Rectangle {
    private int xcoord, ycoord, xsize, ysize;
    public Rectangle(int x, int y, int xs, int ys) {
        xcoord = x;
        ycoord = y;
        xsize = xs;
        ysize = ys;
    }
    public boolean contains(Point p) {
        return (xcoord <= p.xcoord && p.xcoord <= xcoord + xsize &&
                ycoord <= p.ycoord && p.ycoord <= ycoord + ysize);
    }
}
```
Problem 7  
Permutation Checker [9 points]

Write a method called permChecker that checks whether its string arguments are permutations. The method has the following signature:

```java
public static boolean permChecker(String s1, String s2);
```

and behaves such that:

- `permChecker("abcd", "dcab")` evaluates to true
- `permChecker("abcd", "dcbb")` evaluates to false
- `permChecker("abcd", "abc")` evaluates to false
- `permChecker("aabc", "bcaa")` evaluates to true

[Note: To solve this problem, you do not have to generate all permutations of the input strings. This problem is simpler than the Anagram exercise.]

Here is one way of solving this problem:

```java
public static boolean permChecker(String s1, String s2) {
    if (s1.length() != s2.length())
        return false;

    int i, j;
    for (i=0; i<s1.length(); i++) {
        for (j=0; j<s2.length(); j++)
            if (s2.charAt(j) == s1.charAt(i))
                s2 = s2.substring(0, j) + s2.substring(j+1);
            else if (j == s2.length() - 1)
                return false;
    }
    return true;
}
```
Problem 8  
*Arrays and Recursion [6 points]*

Write a recursive method with the following signature:

```
public static int findMax(int[] group, int pos);
```

that computes the maximum value stored in the array `group` at any of the positions starting at `pos` through the end of the array. The following code snippet illustrates the method’s behaviour under different settings.

```
int[] elems = { 1, 9, 3, 8, 7 };  
System.out.println( findMax(elems, 0) ); // prints 9  
System.out.println( findMax(elems, 1) ); // prints 9  
System.out.println( findMax(elems, 2) ); // prints 8
```

Your solution needs to be recursive.

```
public static int findMax(int[] group, int index)  
{
    if (index == group.length - 1)
        return group[index];  
    else {
        int m = findMax(group, index + 1);  
        return (m > group[index] ? m : group[index]);
    }
}
```
Problem 9  Stack Abstraction [7 points]

In this problem, your task is to develop a class that implements a Stack of integers. You can assume that there exists a Java class that supports the abstraction of a List of integers. The methods provided by the List class are:

```java
public class List {
    public List(); // Constructor
    public void addToFront(int elem); // Adds elem to the front of the list
    public int removeFromFront(); // Returns value from the front of the list
    public void addToEnd(int elem); // Adds elem to the end of the list
    public int removeFromEnd(); // Returns value from the end of the list
    public boolean isEmpty(); // Checks whether list is empty
}
```

Use of this List class is illustrated by the following code:

```java
List l = new List();
l.addToFront(1);
l.addToFront(2);
l.addToFront(3);
System.out.println(l.removeFromFront()); // prints 3
System.out.println(l.removeFromEnd()); // prints 1
System.out.println(l.removeFromEnd()); // prints 2
System.out.println(l.isEmpty()); // prints true
l.addToFront(4);
System.out.println(l.removeFromEnd()); // prints 4
```

Without knowing the details of how this List class is implemented, develop the code for a Stack class that supports the following operations. Recall that a Stack implements a LIFO(Last-In-First-Out) ordering for storage and retrieval of elements.

```java
public class Stack {
    List data = new List();
    public void push(int elem) {
        data.addToFront(elem);
    }
    public int pop() {
        return data.removeFromFront();
    }
    public int peek() {
        int tmp = data.removeFromFront();
        data.addToFront(tmp);
        return tmp;
    }
    public boolean isEmpty() {
        return data.isEmpty();
    }
}
```