

1. Matching[10 pts]

Choose the **best** word for each statement.

1. _____ Provides runtime resolution to the most specific implementation possible.
2. _____ A named collection of method headers (without definitions) that can also declare constants.
3. _____ Holds the location of an object.
4. _____ LIFO data structure.
5. _____ The process in which a reference of a given class can refer to any instance of a subclass.
 - A. Constructor
 - B. Polymorphism
 - C. Abstract
 - D. Compiler
 - E. Recursion
 - F. Reference
 - G. Inheritance
 - H. Interface
 - I. Private method
 - J. Dynamic Binding
 - K. Constructor Overloading
 - L. Stack
 - M. Queue

2. Multidimensional Arrays[10 pts]

Write `public float[][] addMatrices(float[][] a, float[][] b)` which adds the values of the two matrices passed in (a and b) and returns the result in a new matrix. You may assume **the arrays passed in are rectangular and have the same dimensions..**

```
public float[][] addMatrices(float[][] a, float[][] b){
```

```
}
```

3. Interfaces[10 pts]

Declare a class **Money** that has the following properties:

- Has a private double called value
- Has an accessor and modifier for the private double.
- Implements the Comparable interface. Items should be compared by the value field to determine if it is less than, equal, or greater than the Object it is being compared to as specified by the interface.

4. Abstract Classes[10 pts]

Given the following class:

```
public abstract class Machine{
    private String name;
    public void setName(String name){ this.name = name; }
    public String getName(){ return name; }
    public abstract double calculatePower();
}
```

Declare an instantiable (concrete) subclass of Machine. You may make the subclass do anything as long as it compiles.

5. Recursion[10 pts]

Write the method `public int geomJava(int x)` which calculates the following geometric series. If the input is less than 1, the method returns 1. Otherwise, the method returns the sum of `geomJava(x/2)` plus `geomJava(x-4)`. You must use recursion to solve this problem. If you use any iteration, or do not use recursion, you will not receive credit.

```
public int geomJava(int x) {
```

```
}
```

6. Stacks and Queues [10 pts]

Assume you have the following **LinkedList** class. You may only assume that you have the following methods fully implemented and fields in the **LinkedList** class:

- private `LLNode head`;
- `public void addToFront(Object o){...}`
- `public void addToBack(Object o){...}`
- `public Object removeFromFront(){...}`
- `public Object removeFromBack(){...}`

Write the following methods for the **Stack** class. The methods should maintain the integrity of the **Stack** data structure.

```
public class Stack extends LinkedList {
```

5 (a) `public void push(Object o){`

```
    }
```

5 (b) `public Object pop(){`

```
    }  
} //end class Stack
```

7. Static vs. Instance [10 pts]

Given the following class, write the output when the main method is run.

```
public class TryMeA{
    private static int one;

    private int two;

    public static void setOne(int x){
        one = x;
    }
    public static int getOne(){
        return one;
    }
    public void setTwo(int two){
        this.two=two;
    }
    public int getTwo(){
        return two;
    }
    public int specialSum(){
        return one*2+two/10;
    }

    public TryMeA(int x){
        one = x/10;
        two = x;
    }
    public static void main(String[] args){
        TryMeA a = new TryMeA(4);
        TryMeA b = new TryMeA(6);
        TryMeA c = new TryMeA(20);
        System.out.println("a.getOne() is " + a.getOne());
        System.out.println("c.getOne() is " + c.getOne());
        a.setOne(4);
        b.setTwo(5);
        System.out.println("b.getOne() is " + b.getOne());
        System.out.println("a.getTwo() is " + a.getTwo());
        System.out.println("c.specialSum() is " + c.specialSum());
    }
}
```

8. Polymorphism [10 pts]

Given the following class hierarchy with listed methods:

- **public abstract class Emotion**
has: **public void express()**
- **public interface Crying**
has: **public void tears()**
- **public class Joy extends Emotion implements Crying**
has: **public void smile()**
has: **public void tears()**
- **public class PureJoy extends Joy**
has: **public void exult()**
- **public class Anger extends Emotion**
has: **public void yell()**

Determine which of the following statements will compile and run without any errors or exceptions. If there is no error, write **OK**; if there will be an error at compile time, write **Compile Error**; if there will be a problem during during execution, write **Runtime Error**. If you know there is an error but cannot determine whether it is at Run-time or compile-time, you may write error and explain why in 15 words or less. If you write both, you will not receive credit for that part.

- 2 (a) `Emotion e = new Emotion();`
- 2 (b) `Emotion my = new Joy();`
- 2 (c) `Emotion great = new PureJoy();`
`great.tears();`
- 2 (d) `Emotion well = new Joy();`
`((Anger)well).yell();`
- 2 (e) `PureJoy lastone = new Joy();`

9. Linked List[10 pts]

Given the following class `LLNode`:

```
public class LLNode{
    private Comparable data;
    private LLNode next=null;
    public void setData(Comparable data){ this.data = data; }
    public void setNext(LLNode next){ this.next = next; }
    public Comparable getData(){ return data; }
    public LLNode getNext(){ return next; }
    public LLNode(Comparable data){ this.data = data; }
}
```

Write the `removeAllOccurrences(Comparable c)` method for the `LinkedList` class, which will remove all occurrences of `c` from the Linked List. You may not assume anything else about the `LinkedList` class or the `LLNode` class. You may write any helper methods you see fit to solve the problem.

```
public class LinkedList{
    private LLNode head;

    public void removeAllOccurrences(Comparable c){
```

```
    }
}
```

10. **Tracing**[10 pts]

Given the following classes, write the output when the main method is run.

```
public class ParentClass{
    public String random(){ return "Cheese"; }
    public String getName(){ return "Me"; }
    public ParentClass(){
        this(null);
        System.out.println("In ParentClass()");
    }
    public ParentClass(String x){
        System.out.println("In ParentClass(String)");
    }
} //end ParentClass

public class ChildClass extends ParentClass{
    public String random(){ return "Mechanic"; }
    public ChildClass(){
        System.out.println("In ChildChildClass()");
    }
    public ChildClass(String x){
        this();
        System.out.println("In ChildChildClass{String}");
    }
    public static void main(String[] args){
        ParentClass a = new ParentClass();
        ChildClass b = new ChildClass();
        System.out.println(a.random());
        System.out.println(a.getName());
        System.out.println(b.getName());
        a = b;
        System.out.println( a.random() );
        System.out.println( ((ParentClass)a).random() );
    }
}
```
