Intermediate Programming Exam 2

1. (5 point) What is your name?

2. (15 points) Add a copy constructor to the Lib object below that makes a deep copy. (Remember that the Java String class is immutable. You may assume that the Book class has a suitable copy constructor.)

```
class Lib
{
   ArrayList<String> titles;
   ArrayList<Book> books;

   Lib()
   {
      titles = new ArrayList<String>();
      books = new ArrayList<Book>();
   }

   void addBook(String title, Book b)
   {
       titles.add(title);
       books.add(b);
   }









}
```

3. (10 points) Below is code for a node in a linked list. Please complete the copy constructor in this class to use recursion to make a deep copy of the list of nodes that begins with node "n".

```
class Node {
   int val;
   Node next;

   Node(int v) { val = v; }

   Node(Node n)
   {









   }
}
```

4. (5 points) What is polymorphism?
   - A way of casting objects to different types.
   - It is when the compiler forces you to implement abstract methods declared in the super class.
   - A way of calling the super constructor.
   - It is when a call behaves differently, depending on the object it operates upon.
   - Using a catch block to suppress exceptions.
   - A debugging technique involving the printing of all variable types.
   - A fancy name for inheritance.
   - Using an iterator to visit all elements in a collection.

5. (5 points) What would be the output of this code?

```
class Stone extends Object
{
   Stone() {
      System.out.print("t");
   }

   String crack(String h) {
      System.out.print("a" + h);
      return "z";
   }
}

class Granite extends Stone {
   Granite(String g) {
      if(g.equals("o"))
         System.out.print("r");
   }

   void fall(String f) {
      System.out.print("b");
      crack(f);
   }
}

class Main {
   public static void main(String[] args)
   {
      Granite q = new Granite("a");
      q.fall("c");
      System.out.print("k");
   }
}
```

6. (5 points) ConcurrentModificationException is thrown when
   - You modify a variable that already references an object.
   - Generic types are used improperly.
   - An iterator detects that the contents of its collection have been changed.
   - You fail to use the "-g" flag to build your code.
   - Code is modified while you are debugging.
   - You modify an object to which multiple variables refer.

7. (5 points) Suppose a class named Alpha extends a class named Beta. Which of these lines will cause a compile error?
   - Beta a = new Alpha();
   - Alpha b = new Beta();

8. (10 points) If you break at a breakpoint on the "break here" line,

```
class Mambo
{
    int a = 0;
    int b = 1;

    public static void main(String[] args)
    {
           int c = 2;
           Mambo x = new Mambo();
           Mambo y = new Mambo();
           Mambo z = new Mambo();
           System.out.println("break here");
    }
}
```

How many Mambo refs will be on the stack?

How many integers will be on the stack?

How many Mambo objects will be on the heap?

How many integers will be on the heap?

9. (5 points) Consider this code:

```
class Cheese {
   Mold d;
   Milk w = null;

   Cheese() {
         Mold d = new Mold();
   }

   void eat(int chews) {
      if(chews < 0)
         throw new Exception("!");
      if(this.w == null)
          System.out.pring("Nooo!");
      int i;
      d.grow(chews + i);
   }
}
```

A NullReferenceException is thrown in the "eat" method. Which of the following will fix the problem? (Circle one)
   - Declare eat to be "static".
   - Change "`Mold d = new Mold();`" to "`d = new Mold();`"
   - Change "`Milk w = null;`" to "`Milk w = new Milk();`".
   - Change "`int i;`" to "`int i = 4;`".
   - Pass a value for chews greater than zero.
   - change the "==" to "!=".

10. (5 points) Circle the true pair of statements:
    - Java is a static typed language. Javascript is a dynamic typed language.
    - Java is dynamic typed language. Javascript is static typed language.

11. (5 points) What will be the output of this Java function if you call f(8)?

```
static void f(int n)
{
        System.out.print(n);
        if(n > 4)
                f(1);
        System.out.print(n);
        if(n > 3)
                f(n / 2);
}
```

12. (5 points) In the space to the right, write a class named Tom that inherits from the Cat class (below). The Tom class should not be abstract.

```
abstract class Cat
{
    int lives;

    Cat(int n)
    {
        lives = n;
    }

    abstract String name();

    boolean setLives(int n)
    {
        lives = n;
    }
}
```

13. (5 points) Give the Tom class a constructor that takes zero parameters, and initially gives it 9 lives. (Note that the Cat class constructor requires one parameter.)

14. (5 points) Add a method to your Tom class that decrements the number of lives. If the number of lives reaches 0, return "false" to indicate that the Tom instance has died. If the Tom instance already has 0 lives when the method is called, throw a RuntimeException to indicate that Tom is already completely dead.

15. (10 points) Use overriding to ensure that the setter cannot be used to give more lives to Tom objects with zero lives. (But when a Tom object has more than zero lives, the setter should still allow the user to adjust its number of lives.)