CS165
First Midterm
Practice Exam

Warning: You must fill in the blank in Canvas with exactly what Java would print. Here are a few hints:

1) Java does not print double quotes on strings, unless asked to, for example:

```
System.out.println("Hello "+ "World!");
// prints Hello World!
```

2) Java does not print single quotes on characters, unless asked to, for example:

```
System.out.println('A' + ";" + '$' + ";" + '8');
// prints A;$;8
```

3) Java prints boolean values as **true** or **false**:

```
System.out.println(5 >= 10);
// prints false

String[] array = { "Denver", "Boulder", "Fort Collins" };
```

4) Java overrides toString() for Arrays and Collections, as follows:

```
String[] array = { "Denver", "Boulder", "Fort Collins" };
System.out.println(Arrays.toString(array));
// prints [Denver, Boulder, Fort Collins]
```

Note the square brackets, commas, and space between entries.

For problems 1-4, circle the single answer that best answers the question. (3X4 = 12)

1. Select the correct description below:

a) In black-box testing, test inputs are derived from the implementation (code).

b) In white-box testing, test inputs are derived from the implementation (code).

c) Branch coverage requires that every decision evaluates to true or false, but not necessarily both.

d) Black-box testing requires statement coverage.

2. Select the best description of the utility of **inheritance** as implemented by the Java language:

a) Allows a programmer to extend classes (to specialize them) without code duplication.

b) Allows a programmer to override legacy code that works with new code that has defects!

c) Allows a programmer to write a subclass that overrides private methods in the super class.

d) Allows a programmer to write a concrete subclass without overriding abstract methods.

3. Select the best description of **polymorphism** as implemented by the Java language:

a) Prevents multiple objects in an inheritance hierarchy from being stored in a single collection.

b) Ensures that the methods associated with the class used to create the object (not the type of the reference variable) are called.

c) Provides dynamic binding which allows any method in any class to be loaded and executed.

d) Lets the programmer decide which method in the inheritance tree is called for a given object.

4. Select the best statement concerning the visibility modifiers:

a) The public modifier makes an instance variable and methods of a class visible to other public (but not private) methods in other classes.

b) The protected modifier makes an instance variable and methods of a class visible only to the subclasses of the class.

c) The private modifier makes an instance variable in an object inaccessible to objects of the same class.

d) The private modifier makes an instance variable in an class inaccessible to other classes.

JUnit assertions (4X2 = 8 points)

Give the declarations below, state which of the listed assertions pass:

```
int x = 10;
int y = 41;
double val = 4.0/3;
String word = "story".substring(2,2);
int[] nums = {0, 0, 0, 0, 0};
int[] copy = nums;
```

Assertion 1:
`assertFalse(y%x == 0);` **PASS**

Assertion 2:
`assertEquals(copy.length,0);` **FAIL**

Assertion 3:
`assertEquals(nums, copy);` **PASS**

Assertion 4:
`assertNotNull(word);` **PASS**

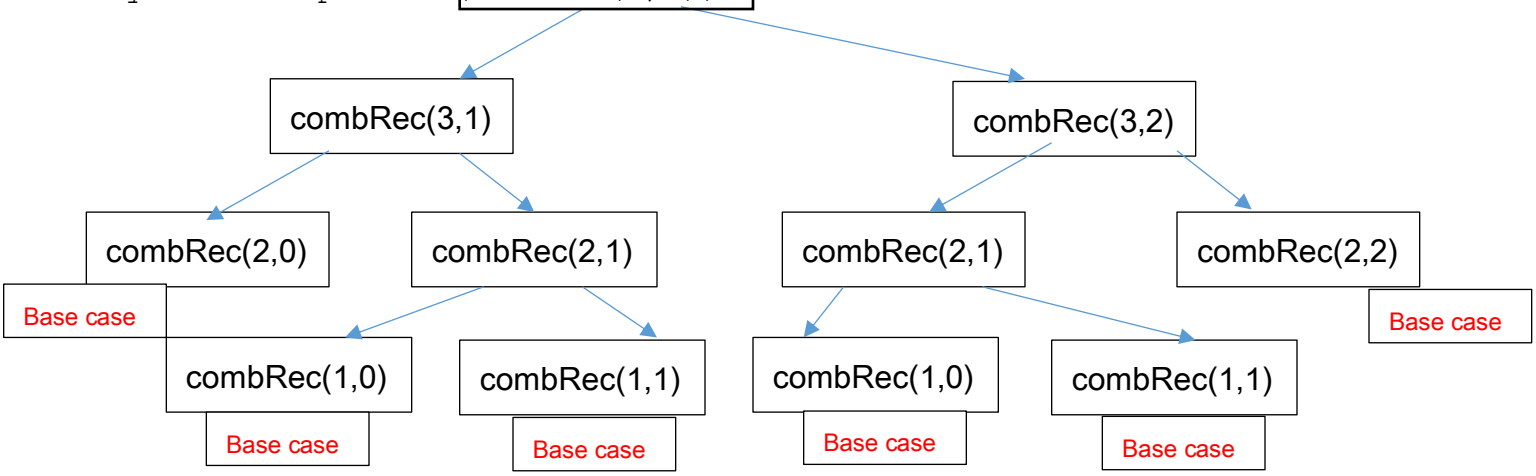
Recursion (6 points)

The code for the recursive method combRec is given below.

```
public long combRec(long n, long k) {
     if (n==k || k==0)
          return 1;
     else
          return combRec(n-1,k-1) +  combRec(n-1,k);
}
```

How many calls are made to to combRec (inclusive of the original call) when you execute: 11

```
System.out.println(combRec(4,2));
```



Inheritance (4X4 + 5 =21 points)

```
public class Book {
   protected int pages;
   protected String name;
   public Book(int pages, String name){
      this.pages = pages; this.name = name;
   }
   public String motto(){
      return this + " The greatest book!";
   }
   public String toString(){
      return "Book: " + name + "\npages: " + pages ;
   }
}
public class Dictionary extends Book {
   private int defs;
   public Dictionary (int pn, String dn, int defs){
      super(pn,dn);
      pages = 2*pages;
      this.defs = defs;
   }
   public int getDefs(){
      return defs;
   }
   public String motto(){
      return("This dictionary is a fine " + super.motto() );
   }
   public String toString(){
      return super.toString() + " defs: " + defs;
   }
 }
 public class Words {
   public static void main(String[] args) {
      Dictionary D = new Dictionary(200, "Webster", 10000);
      Book limbo = new Book(150, "Alice in Wonderland");
      System.out.println(limbo.motto());
      limbo = D;
      System.out.println(limbo.motto());

      // what is wrong with the following line and how can it be fixed?
      //int defs = limbo.getDefs();    //
   }
}
```

What are the 4 lines printed above?

```
Book: Alice in Wonderland
pages: 150 The greatest book!
This dictionary is a fine Book: Webster
pages: 400 defs: 10000 The greatest book!
```

What is wrong with the last line in the class and how would you fix it? limbo is of type Book and doesn't have the getDefs method. Cast limbo to Dictionary.

Inheritance (4X4 =16 points)

Show what the program shown below would print.
HINT: Consider the inheritance hierarchy and polymorphism

```
public class PolymorphismProgram {

   public static class Vehicle {
      public void printMe() {
         System.out.println("Vehicle");
      }
   }
   public static class Car extends Vehicle {
      public void printMe() {
         System.out.println("Car");
      }
   }
   public static class Chevy extends Car {
      public void printMe() {
         System.out.println("Chevy");
      }
   }
   public static class Ford extends Car {
      public void printFord() {
         System.out.println("Ford");
      }
   }
   public static void main(String[] args) {
      Vehicle a = new Vehicle();
      Vehicle b = new Car();
      Vehicle c =  new Chevy();
      Car d = new Ford();
      a.printMe();
      b.printMe();
      c.printMe();
      d.printMe();
   }
}
```

Vehicle
Car
Chevy
Car (since printMe is not overridden in Ford)

Constructor Chaining (15 points)

What does this program print?

```
class P {
	public P() {
		System.out.println("P");
	}
	public P(int x) {
		System.out.println("P" + x);
	}
}

class Q extends P {
	public Q(int x) {
		System.out.println("Q"+x);
	}
	public Q() {
		this(4);
		System.out.println("Q");
	}
}


public class ChainingPractice {

	public static void main(String[] args) {
		Q q1 = new Q(3);
		Q q2 = new Q();
	}
}
```

P
Q3
P
Q4
Q

Interfaces (4X4 =16 points)

```
import java.util.Arrays;
public class State implements Comparable<State> {
	private String name;
	private int year;

	public State(String name, int year) {
		this.name = name;
		this.year = year;
	}
	@Override
	public int compareTo(State o) {
		if(this.year > o.year)
			return 1;
		else if(this.year < o.year)
			return -1;
		else
			return 0;
	}
	@Override
	public boolean equals(Object o) {
		if(o instanceof State) {
			State other = (State)o;
			return (name.equals(other.name));
		}
		return false;
	}
	@Override
	public String toString() {
		return name + " " + year;
	}
	public static void main(String[] args) {
		State s1 = new State("Idaho", 1890);
		State s2 = new State("Maine", 1820);
		State s3 = new State("Texas", 1845);
		State s4 = new State("Utah", 1896);
		State[] list = new State[4];

		list[0] = s1;
		list[1] = s2;
		list[2] = s3;
		list[3] = s4;
		System.out.println(Arrays.toString(list));          // Line 1

		System.out.println(s1.compareTo(s2));  // Line 2
		System.out.println(s2.equals("Maine"));// Line 3
		System.out.println(s3.compareTo(s4));  // Line 4

		double total=0;
		for(State s: list) {
			total = total + s.year;
		}
		System.out.println((int)total/4); // Line 5
	}
}
```

What does this program print? (next page)

```
[Idaho 1890, Maine 1820, Texas 1845, Utah 1896]
1
false
-1
1862
```

Generics (3X2 =6 points)

Assume that the appropriate import statement for ArrayList exists.

```
// Fragment A
ArrayList<String> a1 = new ArrayList<>();
a1.add("Hi");
a1.add("Bye");
Collections.sort(a1);
System.out.println(a1);
```

TRUE: This fragment compiles without errors or warnings.

TRUE: This fragment executes without exceptions.

```
// Fragment B
ArrayList a2 = new ArrayList();
a2.add("Hello");
a2.add(25);
Collections.sort(a2);
System.out.println(a2);
```

FALSE: This fragment compiles without errors or warnings.

FALSE: This fragment executes without exceptions.

```
// Fragment C
ArrayList<Integer> a3 = new ArrayList<>();
a3.add(45);
a3.add("45");
Collections.sort(a3);
System.out.println(a3);
```

FALSE: This fragment compiles without errors or warnings.

FALSE: This fragment executes without exceptions. (doesn't execute because it doesn't compile)