1) (10 pts) Fill in the blanks to specify the missing keywords or definitions.

```java
public class SomeClass extends OtherClass implements SomeInterface { ... }
```

The keyword `super` is used to reference the parent/base class.

If a class, variable, or method is `public`, then it can be referenced in any other class.

A(n) **local** variable is defined in the body of a method.

A method that has the same name and signature as a method in its parent class is **overriding**.

A(n) **constructor** creates a new instance of an object.

Each instance of an object has its own copy of the **instance** variables.

A(n) **interface** contains method signatures/header, but never contains any method bodies.

To test if `instance1` is an instance of the class `SomeClass`, we write `instance1 instanceof SomeClass`.

2) (30 pts) Write the output of each piece of code. If the code gives an error, write any output produced before the error, and then write “ERROR”.

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) (5 points) What will be the output from the call to <code>mystery(3)</code>; public static void mystery(int num){ if(num&lt;=0){ System.out.println(num); } else{ //before recursive call System.out.print(num); mystery(num-1); } }</td>
<td>3210</td>
</tr>
</tbody>
</table>
| b) (5 points) | What will be the output from the call `mystery(4)`? | -1  
1  
2  
3  
4 |
| --- | --- | --- |
| public static void mystery(int num){  
  if(num<=0){  
    System.out.println(num-1);  
  }  
  else{  
    mystery(num-1);  
    // after recursive call  
    System.out.println(num);  
  }  
} |
| c) (5 points) | String a = "Bedford";  
String b = "Fordham";  
System.out.println(  
a.substring(0,4) + a.charAt(6)  
+ b.substring(4,7)); | Bedfdham |
| --- | --- | --- |
| d) (5 points) | int i = 4;  
double c = 2;  
while( i > 1 ) {  
  System.out.println(i/c);  
  System.out.println(i/3);  
  i--;  
} | 2.0  
1  
1.5  
1  
1.0  
0 |
### e) (5 points)

String[] a = {"carrot", "celery", "lettuce", "tomato"};
String[] b = a;
System.out.println(a[0]);
a[0] = "beet";
System.out.println(a[1]);
System.out.println(a[3]);
System.out.println(a[0]);
System.out.println(b[0]);
System.out.println(b[1]);

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>carrot</td>
<td>carrot</td>
</tr>
<tr>
<td>celery</td>
<td>celery</td>
</tr>
<tr>
<td>tomato</td>
<td>tomato</td>
</tr>
<tr>
<td>beets</td>
<td>beets</td>
</tr>
<tr>
<td>beets</td>
<td>beets</td>
</tr>
<tr>
<td>celery</td>
<td>celery</td>
</tr>
</tbody>
</table>

### f) (5 points)

boolean flag = true;
int n = -1;
int k = 4;
do {
    System.out.println("n = " + n + ", k = " + k);
    if (flag) {
        n++;
        flag = false;
    } else {
        k--;
    }
    if (k == 0) {
        flag = true;
    }
} while ((n < k) & & !flag);

<table>
<thead>
<tr>
<th>n</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
3) (30 pts) Use the classes below to complete this section.

a) (15 pts)
Do the following for the class Mammal:

I. (5 pts)
Write get and set methods for the mass variable. Be sure only positive values are set; negative or zero values should throw an Exception. Assume the other private variables have their getter/setter methods already defined.

II. (5 pts)
Define the compareTo method so that it compares the mass of the Mammal calling it with the mass of the otherMammal. The method should return -1 if the mass of the calling Mammal is less than that of otherMammal, 0 if the two are equal, and 1 otherwise. Comment your code.

III. (5 pts)
Override the default toString method so it returns a nicely-formatted String containing the names and values of all the Mammal's variables.

```java
public class Mammal implements Comparable<Mammal> {
    private boolean hasTail;
    private int numLegs;
    private double mass=0.0;

    public Mammal(){
        numLegs=2;
        hasTail=false;
        mass=10;
    }

    public void setHasTail(boolean inTail) {...}
    public void setNumLegs(int inLegs) {...}

    // write your get and set methods here

    public void setMass(double newMass) throws Exception {
        if( newMass > 0 ){
            mass = newMass;
        } else {
            throw new Exception("Mass cannot be negative.");
        }
    }

    public double getMass() {
        return mass;
    }
```
public int compareTo(Mammal otherMammal) {
    if (mass < otherMammal.mass) {
        return -1;
    }
    if (mass == otherMammal.mass) {
        return 0;
    }
    return 1;
}

public String toString() {
    String retString = "This mammal has " +
                       numLegs + " legs. 
" +
                       "Its mass is " + mass + "kg. 
" +
                       "It ";
    if (hasTail) {
        retString += "has ";
    } else {
        retString += "doesn't have ";
    }
    retString += "a tail."

    return retString;
}
b) (15 pts) Do the following for the subclass Dog:

I. (5 pts) Define the default constructor so it sets hasTail to true, numLegs to 4, and mass to 10.0, as in the parent class.

II. (5 pts) Create a setter for mass that throws an Exception if the input parameter is less than 2 or greater than 350.

III. (5 pts) Write an equals method for Dog. It should return true if the breed and name of the two Dog instances are the same, and false otherwise.

```java
public class Dog extends Mammal {
    private String breed;
    private String name;
    public Dog() {
        super();
        super.setHasTail(true);
        super.setNumLegs(4);
    }
    // question 3bI
    public Dog() { 
        super();
        setHasTail(true);
        setNumLegs(4);
        /** without super or this it will default to the nearest method... if none exists in the child class it will use the on inherited from the parent */
    }
    public void setMass(double newMass) throws Exception {
        if (newMass >= 2 && newMass <= 350) {
            super.setMass(newMass);
        } else {
            throw new Exception("Mass cannot be less than 2 
or greater than 350");
        }
    }
    public boolean equals(Object o) {
        // assume o is Dog object
        return breed.equals(o.breed) && name.equals(o.name);
    }
}
// question 3bIII
@Override
public boolean equals(Object otherObj){
    if((otherObj!=null)&&(otherObj instanceof Dog)){
        Dog otherDog = (Dog)otherObj;//cast otherObject to dog
        return ((this.breed.equalsIgnoreCase(otherDog.breed)) &&
        (this.name.equalsIgnoreCase(otherDog.name)));
    }
    else{
        return false; //either otherObject was null or not a Dog
    }
}
```
4) (10 points) Write a static method divide that takes in three int arrays, inArr, firstHalf, and secondHalf, as parameters. It should copy the first half of the array inArr into the array firstHalf and copy the second half of the array inArr into the array secondHalf.

Assume that the arrays firstHalf and secondHalf have already been created, and are the correct size. If the array inArr has an odd size, then the array secondHalf will have the extra entry.

Here is the method signature:

```java
public static void divide(int[] inArr, int[]firstHalf, int[] secondHalf){
    int middle = inArr.length / 2;
    for(int i = 0; i < middle; i++ ) {
        firstHalf[i] = inArr[i];
    }
    for(int i = middle; i < inArr.length; i++ ) {
        secondHalf[i-middle] = inArr[i];
    }
}

// alternate answer
public static void divide(int[] inArr, int[]firstHalf, int[]secondHalf ){
    for(int i=0; i<firstHalf.length; i++){
        firstHalf[i]=inArr[i];
    }
    for(int i=0; i<secondHalf.length; i++){
        secondHalf[i]=inArr[i+firstHalf.length];
    }
}
```
5) (15 points) Write a **recursive** static method `longString` that takes two parameters, `String s` and `int num`. It should return the string formed by `s` repeated `num` times with a single space in between repetitions.

For example, if `s` is "kayak", and `num` is 5, then `longString` should return "kayak kayak kayak kayak kayak".

```java
public static String longString(String s, int num) {
    if (num == 0) {
        return "";
    }
    if (num == 1) { // this to eliminate space at end
        return s;
    }
    return s + " " + longString(s, num - 1);
}
```
6) (20 points) Write a method called `addStars` that takes in two `String` objects `infileName` and `outfileName`. The method should open and read from the file `infileName`. It should then write the following to the file `outfileName`: the first line of the file `infileName`, followed by a line with 5 `*`s, then the second line of the file `infileName`, followed by a line with 5 `*`s, etc.

For example, if the contents of `infileName` is:

Twinkle twinkle little star,
How I wonder what you are,
Up above the world so high,
Like a diamond in the sky.

The contents of `outfileName` would be:

Twinkle twinkle little star,
*****
How I wonder what you are,
*****
Up above the world so high,
*****
Like a diamond in the sky.
*****

```java
public static void addStars(String infileName, String outfileName) {
    Scanner inFile = null;
    PrintWriter outFile = null;
    try {
        inFile = new Scanner(new File(infileName));
        outFile = new PrintWriter(outfileName);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    String line;
    while( inFile.hasNextLine() ) {
        line = inFile.nextLine();
        outFile.println(line);
        outFile.println("*****");
    }
    inFile.close();
    outFile.close();
}
```
(This page is intentionally left blank.)