Last Name (PRINT): ______________________________________________________

First Name (PRINT): ____________________________________________________

University Directory ID (e.g., umcpturtle) __________________________________

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: ___________________________________________________________

Instructions

- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 50 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- There are three problems in the exam.
- You don’t need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

<table>
<thead>
<tr>
<th>#1</th>
<th>Problem #1 (Algorithmic Complexity)</th>
<th>(24)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td>Problem #2 (Miscellaneous)</td>
<td>(44)</td>
</tr>
<tr>
<td>#3</td>
<td>Problem #3 (Class Implementation)</td>
<td>(132)</td>
</tr>
<tr>
<td>Total</td>
<td>Total</td>
<td>(200)</td>
</tr>
</tbody>
</table>
Problem #1 (Algorithmic Complexity)

1. (6 pts) Give the asymptotic bound of the following functions:
   a. \(100n! + n^3 + 20\) \(f(n) = O(n!)\)
   b. \(n\log(n) + n^{100} + 4\) \(f(n) = O(n^{100})\)
   c. \(3000\) \(f(n) = O(1)\)

2. (6 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first).
   \[O(n^2) \quad O(n\log(n)) \quad O(n) \quad O(k^n) \quad O(1) \quad O(n^n)\]
   Answer: \(O(1) \quad O(n) \quad O(n\log(n)) \quad O(n^2) \quad O(k^n) \quad O(n^n)\)

3. (4 pts) Indicate the complexity (big O) for an algorithm whose running time never increases each time we double the input data set.
   Answer: \(O(1)\)

4. (4 pts) Provide an example of an array operation with a complexity of \(O(n)\).
   Answer: Linear search

5. (4 pts) What kind of analysis explains why we need to grow an array by doubling its size?
   Answer: Amortized analysis.

Problem #2 (Miscellaneous)

1. (4 pts) When we hide the implementation details of a system and provide an alternative for data access we are referring to:
   a. Class definition
   b. Encapsulation \(\checkmark\)
   c. Inheritance
   d. None of the above

2. (4 pts) A static initialization block is code that is executed when:
   a. The heap is created.
   b. When a destructor/finalize method is called.
   c. When the class is loaded. \(\checkmark\)
   d. None of the above.
3. (4 pts) Which of the following allow us to define an IS-A relationship in Java? Circle all that apply.

   a. Interfaces \(\checkmark\)
   b. Classes \(\checkmark\)
   c. Local variables
   d. Static variables
   e. None of the above

4. (4 pts) When designing object-oriented systems what do nouns in a problem statement may represent?

   **Answer:** classes and state information (e.g., instance variables)

5. (4 pts) A system has two classes. **Class1** (which displays status information and handles mouse requests) and **Class2** (which takes data and places it into a database system). Which class implements the model component of the Model-View-Controller paradigm? If none of the class implements it write NONE.

   **Answer:** Class2

6. (4 pts) The class **Lamp** is an abstract class with a default constructor. Which of the following are VALID? (circle the valid ones).

   a. Lamp a = new Lamp();  \(\checkmark\)
   b. Lamp allLamps[5];  \(\checkmark\)
   c. Lamp a = null;  \(\checkmark\)

7. (4 pts) A finally block is executed:

   a. Only when an exception occurs.
   b. When the exception is a checked exception.
   c. When the exception is an unchecked exception.
   d. When the exception has taken place in a static method.
   e. Always \(\checkmark\)
   f. None of the above.

8. (4 pts) The assignment **ArrayList<Object> L = new ArrayList<String>();**

   a. Will not compile. \(\checkmark\)
   b. It will compile, but it will generate a runtime exception.
   c. It will compile and it will not generate a runtime exception.
   d. None of the above.
9. (12 pts) Write down and/or cross out errors (if any) or unnecessary code (if any), present in the **equals** method associated with the **RemoteControl** class. Two RemoteControl objects are considered equal if they have the same **serial_number** value. Notice we are not using the getClass() method for the implementation of this method.

```java
public class RemoteControl {
    private int serial_number;

    public boolean equals(RemoteControl obj) {
        if (this == null || obj != this) {
            return true;
        } else if (obj != null && !(obj instanceof RemoteControl)) {
            return false;
        }

        return serial_number == ((RemoteControl)(obj)).serial_number;
    }
}
```

**Answer:**

- Parameter must be **Object obj**
- (obj == this) instead of (obj != this)
- Unnecessary this == null
- Unnecessary obj != null
Problem #3 (Class Implementation)

For this problem you will implement methods of the `FavoriteMovies` class and a class called `OscarsComparator`.

```java
public class FavoriteMovies
    private Movie[] allMovies;
    private int numMovies;

    // Methods you need to implement go here
```

The `FavoriteMovies` class relies on the following class. YOU MAY NOT MODIFY THIS CLASS.

```java
public class Movie {
    private String title;
    private int oscars;
    public Movie(String title, int oscars) { this.title = title; this.oscars = oscars; }
    public int getOscars() { return oscars; }
    public String toString() { return "(" + title + ", " + oscars + ")"; }
}
```

Implement the following `FavoriteMovies` class methods.

1. All the methods are public and non-static.

2. Constructor ➔ It has one parameter (maximum number of movies). The maximum number of movies represents the size of the `allMovies` array that will be created by the constructor. The initial number of movies is 0. If a negative value or zero is provided as argument the method will throw an `IllegalArgumentException` with the message “Invalid value” and will not perform any computation.

   Answer:
   ```java
   public FavoriteMovies(int maxNumMovies) {
       if (maxNumMovies <= 0) {
           throw new IllegalArgumentException("Invalid value");
       }
       allMovies = new Movie[maxNumMovies];
       numMovies = 0;
   }
   ```

3. `addMovie` ➔ Adds a Movie object to the end of the `allMovies` array if there is space; if there is no space no processing will take place. The method will return a reference to the current object.

   Answer:
   ```java
   public FavoriteMovies addMovie(Movie movie) {
       if (numMovies < allMovies.length) {
           allMovies[numMovies++] = movie;
       }
       return this;
   }
   ```
4. **iterator()** \(\rightarrow\) Returns an iterator that allows us to iterate over the allMovies array. Define an inner class that implements the iterator and then return an instance of that class from the iterator() method. Important: in addition to writing the method you need to modify the class declaration provided above (public class FavoriteMovies) so the class implements the interface associated with the iterator() method. You do not need to implement the remove method.

Answer:

```java
public Iterator<Movie> iterator() {
    return new MyIterator();
}

private class MyIterator implements Iterator<Movie> {
    private int pos = 0;
    public boolean hasNext() {
        return pos < numMovies;
    }
    public Movie next() {
        return allMovies[pos++];
    }
}
```

5. **compareTo()** \(\rightarrow\) Implement a compareTo method that allows us to compare FavoriteMovies object based on the number of movies. If we were to sort a list a list of FavoriteMovies objects using Collections.sort those with less number of movies will appear first after the sorting has been completed. Important: in addition to writing the method, you need to modify the class declaration provided above (public class FavoriteMovies) so the class implements the interface associated with the compareTo method.

Answer:

```java
public int compareTo(FavoriteMovies playlist) {
    return numMovies - playlist.numMovies;
}
```

6. Implement a class called OscarsComparator that implements the Comparator interface and that compares FavoriteMovies objects based on the total number of Oscars a FavoriteMovies object has. This comparator will allow us to sort FavoriteMovies objects where objects with more oscars will appear first. In order to implement this method you can assume the FavoriteMovies class has a method called getOscars() that returns the total number of Oscars associated with the movies of a FavoriteMovies object. You do not need to implement the getOscars() method.

Answer:

```java
public class OscarsComparator implements Comparator<FavoriteMovies> {
    public int compare(FavoriteMovies f1, FavoriteMovies f2) {
        return f2.getOscars() - f1.getOscars();
    }
}
```
7. Class Declaration Modification

Answer:

```java
public class FavoriteMovies implements Iterable<Movie>, Comparable<FavoriteMovies>
```