Problem I: Warm Up (40 Points)

Warm up with some critical concepts in Java Object Oriented and Multi-threading Programming: Choose the correct answer(s). Explain and justify.

2pts  a. Which of the following statements is correct for a method that overrides the following method:

```
public void add(int a) {
}
```

- The overriding method must return void
- The overriding method must return int
- The overriding method can return any type

2pts  b. Which of the following statements is correct for a method that overloads the following method:

```
public void add(int a) {
}
```

- The overloading method must NOT return void
- The overloading method can return any type
- The overloading method must take int a as a parameter
- The overloading method cannot take int as its ONLY parameter

Handout continues on the next page(s)
c. Given the following classes defined in separate files, what is the effect of compiling and running class Test?

```java
class Vehicle {
    public void drive() {
        System.out.println("Vehicle: drive");
    }
}

class Car extends Vehicle {
    public void drive() {
        System.out.println("Car: drive");
    }
}

public class Test {
    public static void main(String args[]) {
        Vehicle v;
        Car c;
        v = new Vehicle();
        c = new Car();
        v.drive();
        c.drive();
        v = c;
        v.drive();
    }
}
```

- Generates compile error at `v = c`
- Generates runtime error at `v = c`
- Prints: `Vehicle: drive`  
  - Car: drive
  - Car: drive
- Prints: `Vehicle: drive`  
  - Car: drive
  - Vehicle: drive

*Handout continues on the next page(s)*
d. What is wrong with the following code?

```java
class MyException extends Exception {}

public class Q1d {
    public class MyExcetion {}

    public void foo() {
        try {
            bar();
        } finally {
            baz();
        }
    }

    public void bar() throws MyException {
        throw new MyException();
    }

    public void baz() throws RuntimeException {
        throw new RuntimeException();
    }
}
```

- Since the method `foo()` does not catch the exception generated by the method `baz()`, it must declare the `RuntimeException` in a `throws` clause
- A `try` block cannot be followed by both a `catch` and a `finally` block
- An empty `catch` block is not allowed
- A `catch` block cannot follow a `finally` block
- A `finally` block must always follow one or more `catch` blocks

```
public class Main extends MinBase {
    public static void main(String argv[]) {
        int[] arr = new int[5];

        for (i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

- Prints a sequence of five zeros
- Generates runtime error since array `arr` has not been initialized
- Generates other error(s)
f. Which statement(s) are true about the following code?

```java
class A {
    public A() {}  
    public A(int i) { this(); }
}
class B extends A {
    public boolean B(String msg) { return false; }
}
class C extends B {
    private C() { super(); }
    public C(String msg) { this(); }
    public C(int i) {}
}
```

- The code will fail to compile
- The constructor in A that takes an int as an argument will never be called as a result of constructing an object of class B or C
- Class C defines three constructors
- Objects of class B cannot be constructed
- At most one of the constructors of each class is called as a result of constructing an object of class C
g. What are possible outputs of the given function?

```java
public void divide(int a, int b) {
    try {
        int c = a / b;
    } catch (Exception e) {
        System.out.print("Exception");
    }
    finally {
        System.out.println("Finally");
    }
}
```

- Finally
- Exception
- Exception Finally
- No Output

3pts

h. Consider the `Comparable<T>` java interface, which implements `int compareTo(T o)` method. Which type constraints, when inserted at (1), will allow the class to compile?

```java
class Interval<__________> {
    // (1) TYPE CONSTRAINT HERE
    private N lower, upper;
    public void update(N value) {
        if (lower == null || value.compareTo(lower) < 0)
            lower = value;
        if (upper == null || value.compareTo(upper) > 0)
            upper = value;
    }
}
```

- N extends Object
- N extends Comparable<N>
- N extends Object & Comparable<N>
- N extends Number
- N extends Number & Comparable<N>
- N extends Comparable<N> & Number
- N extends Integer
- N extends Integer & Comparable<N>
i. Given the following interface declaration, which declaration is valid?

```java
interface I {
    void setValue (int val);
    int getValue();
}
```

- Option 1:
  ```java
class A extends I {
    int value;
    void setValue(int val) { value = val; }
    int getValue() { return value; }
}
```

- Option 2:
  ```java
interface B extends I {
    void increment();
}
```

- Option 3:
  ```java
abstract class C implements I {
    int getValue() { return 0; }
    abstract void increment();
}
```

- Option 4:
  ```java
interface D implements I {
    void increment();
}
```

- Option 5:
  ```java
class E implements I {
    int value;
    public void setValue(int val) { value = val; }
}
```
j. Which parameter declarations can be inserted at (1) so that the program compiles without warning?

```java
interface Wagger {}
class Pet implements Wagger {}
class Dog extends Pet {}
class Cat extends Pet {}
public class Q1k {
    public static void main (String[] args) {
        List<Pet> p = new ArrayList<Pet>();
        List<Dog> d = new ArrayList<Dog>();
        List<Cat> c = new ArrayList<Cat());
        examine(p);
        examine(d);
        examine(c);
    }

    static void examine(List<? extends Pet> pets) { // (1)
        System.out.print("Your pets need urgent attention.");
    }
}
```

- List<? extends Pet>
- List<? super Pet>
- List<? extends Wagger>
- List<? super Wagger>
- List<?>
- All of the above
k. Given the following code, which statements are true about the program?

```java
public class Person {
    protected transient String name;
    Person() { this.name = "NoName"; }
    Person(String name) { this.name = name; }
}

public class Student extends Person {
    protected long studNum;
    Student() { }
    Student(String name, long studNum) {
        super(name);
        this.studNum = studNum;
    }
}

import java.io.Serializable;
public class GraduateStudent extends Student implements Serializable {
    private int year;
    GraduateStudent(String name, long studNum, int year) {
        super(name, studNum);
        this.year = year;
    }
    public String toString() {
        return "(" + name + "," + studNum + "," + year + ")";
    }
}

import java.io.*;
public class Q11 {
    public static void main(String args[])
    throws IOException, ClassNotFoundException {
        FileOutputStream outputFile = new
        FileOutputStream("storage.dat");
        ObjectOutputStream outputStream = new
        ObjectOutputStream(outputFile);
        GraduateStudent stud1 = new GraduateStudent("Aesop", 100, 1);
        System.out.print(stud1);
        outputStream.writeObject(stud1);
        outputStream.flush();
        outputStream.close();

        FileInputStream inputFile = new
        FileInputStream("storage.dat");
        ObjectInputStream inputStream = new
        ObjectInputStream(inputStream);
        GraduateStudent stud2 = (GraduateStudent)
        inputStream.readObject();
        System.out.println(stud2);
        inputStream.close();
    }
}
```

- Fails to compile
- Compiles, but throws a runtime exception
- Prints (Aesop, 100, 1)
- Prints (null, 0, 1)
1. Which statements are true about the classes SubA, SubB, and SubC?

<table>
<thead>
<tr>
<th>Class</th>
<th>SubB will not compile</th>
<th>SubC will not compile</th>
<th>SubB will compile</th>
<th>SubC will compile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Class SubB will not compile
Class SubC will not compile
Class SubB will compile
Class SubC will compile
Class SubB overloads the methods in class SupA
Class SubC overloads the methods in class SupA
Class SubB overrides the methods in class SupA
Class SubC overrides the methods in class SupA

2pts  m. Which interface is used to define a class that can execute within its own thread?

- Run
- Runnable
- Thread
- Threadable
- Executable

2pts  n. Which method is used to schedule a thread for execution?

- Init()
- Start()
- Run()
o. Which method(s) may cause a thread to stop execution?

- Sleep()
- Stop()
- Yield()
- Wait()
- Notify()

p. Given the following code, which of the following statements are true?

```java
public class Agg {
    public static void main(String argv[]) {
        Agg a = new Agg();
        a.go();
    }
    
    public void go() {
        DSRoss ds1 = new DSRoss("one");
        ds1.start();
    }
}

class DSRoss extends Thread {
    private String sTname = "";
    DSRoss(String s) {
        sTname = s;
    }
    
    public void run() {
        notwait();
        System.out.println("finished");
    }
    
    public void notwait() {
        while (true) {
            try {
                System.out.println("waiting");
                wait();
            } catch (InterruptedException ie) {}
            System.out.println(sTname);
            notifyAll();
        }
    }
}
```

- Fails to compile
- Prints "waiting"
- Prints "waiting" "finished"
- Compiles, but throws a runtime exception
Problem II: Practice with Inheritance and Polymorphism (15 Points)

Let’s use polymorphism to implement a very famous story line from our childhood - the Tortoise and Hare race. In this race, the Tortoise moves slowly and steadily with the motto of "slowly and steady wins the race." On the other hand, the Hare spurs to the finish line with the philosophy of "jump, let the muscles pump." Hence our two racers differ in the way they move.

To represent a contender, we define a base class `Racer` and two derived classes `Tortoise` and `Hare`. The Racer class has three instance variables: String `ID` that identifies the type of racer ('Tortoise' and 'Hare'), int `x` which denotes the x-axis position, and int `y` which denote the y-axis position. Inspect and study the activity’s archive, `HareTortoise.zip`, and perform the following:

5pts. Create a collection of racers: before a race begins, the user enters, via a pre-programmed dialog box, the desired number of racers of each type. Your job is to create, instantiate, and add Tortoise and/or Hare objects to an ArrayList leveraging the concept of polymorphism. Complete the appropriate method to do so.

5pts. Run the race: run the race by moving and re-drawing all the racers towards the finish line. Complete the appropriate method to do so.

5pts. Rank the racers: at the end of the race, display the final positions of all the racers sorted by their proximity to the finish line (from the farthest to the closest). For this purpose, use the sort method of the Arrays class. Complete the appropriate method to do so.

Problem III: Practice with Socket Programming (45 Points)

In this exercise, you will code a group chat room. In this chat room, clients connect to a remote server with a designated IP and port number. The server accepts connections from an arbitrary number of clients. Any message sent from one client is broadcast to all other clients. You must leverage the concepts of socket programming and multi-threading to achieve this task. You may or may not create a GUI for your chat room.