I have not received nor given any unauthorized assistance in completing this exam.

Signature: ________________________

Name: ___________________________

PID: _____________________________

Please be sure to put your PID at the top of each page.
This page left intentionally blank. Do not write on it.
Part I: Multiple choice (40 points total)

Directions:
For each multiple choice question below, indicate all correct answers among the choices provided. More than one answer may be correct and at least one answer will be correct. Each question is worth 5 points. Selecting all of the choices or selecting none of the choices will result in no points awarded.

1. Which of these statements about the relationship between a subclass and its parent class is true?
   a) A parent class constructor is called after the subclass constructor finishes executing.
   b) A subclass has direct access to protected fields defined in a parent class only if it is defined in the same package.
   c) A subclass can change the access modifier of a parent class field by redeclaring the field.
   * d) A subclass never has direct access to private fields declared in the parent class.
   * e) Two subclasses with a common parent class have direct access to each other's public fields.

2. Which of these statements about inheritance is true?
   a) A subclass must declare any interfaces implemented by its parent class.
   * b) A subclass can inherit from only one parent class.
   * c) All classes inherit from Object.
   d) A subclass can be legally cast to any sibling subclass (i.e., a subclass that also inherits from the same parent class).
   e) A subclass does not inherit private fields from its parent class.

3. Which of these statements about overriding is true?
   a) An overriding method can be declared as non-virtual.
   * b) An overriding method defined in a subclass must have the same method signature as declared in the parent class.
   c) A subclass can only override methods declared in its immediate parent class.
   * d) Overriding is related to the principle of polymorphism.
   * e) A method can be simultaneously overridden and overloaded.

4. Which of these statements about Composition and Aggregation are true?
   * a) Providing encapsulated objects as parameters to a constructor is a characteristic of aggregation.
   * b) Providing getters and setters for its encapsulated objects is a characteristic of aggregation.
   * c) An object can exhibit characteristics of both composition and aggregation.
   d) Encapsulated objects of a composition will also be compositions.
   e) Composition and aggregation are design techniques that rely on polymorphism.
5. Which of these statements about exceptions are true?
* a) An exception that is a direct subclass of Exception is subject to the “catch-or-specify” policy.
* b) The finally block of a try-catch-finally structure is executed if a method returns within a try block without causing an exception.
c) If more than one catch block has an is-a relationship with a thrown exception, all of them will be executed.
* d) A catch block handling an exception may throw a different exception while executing.
* e) A try-catch-finally structure can be nested within a try block.

6. Which of these statements about the Factory design pattern are true?
* a) A class employing the Factory design pattern will usually not have any public constructors.
* b) A class employing the Factory design pattern will usually have at least one public static method.
c) Dynamic subclass binding is not a use case for the Factory design pattern.
d) The Factory design pattern relies on inheritance.
* e) A class using the Factory design pattern prevents direct instantiation of new instances.

7. Which of these statements about Observer/Observable are true?
a) An observable class usually provides a public static method for registering observers.
* b) Java user interface components are observable.
c) The Observer/Observable design pattern requires an observer class to be immutable.
d) All observer classes of a particular observable class will extend a common parent class.
* e) An observable object notifies any registered observers when its state changes.

8. Which of these statements about the Decorator design pattern are true?
* a) The Decorator design pattern uses inheritance to add additional state or functionality to an existing interface.
* b) The Decorator design pattern relies on delegation.
c) A class that implements a decorated interface must be a subclass of a class that implements the original interface.
* d) The constructor of a class using the Decorator design pattern requires an instance of the original interface to be provided as a parameter.
e) Once a decorated instance is created, the original instance cannot be retrieved.
Part II: Is-A Relationships (10 points total)

Suppose the following interfaces and classes are declared (the actual definitions are unimportant)

```
public interface I1
public interface I2
public interface I3 extends I4
public interface I4 extends I1

public class A extends B implements I4
public class B
public class C extends B implements I3
public class D extends E implements I3
public class E extends F
public class F implements I2
public class G extends E implements I1, I2
```

Suppose the following variables are defined:

```
A obj_a = new A();
B obj_b = new B();
C obj_c = new C();
D obj_d = new D();
E obj_e = new E();
F obj_f = new F();
G obj_g = new G();
```

For each of the following statements, draw a line through any of the statements that are not legal (i.e., would either not compile or cause a runtime exception).

```
I1 ref_i1 = (I1) obj_d;
I2 ref_i2 = (I2) obj_d;
I3 ref_i3 = (I3) obj_a;
I4 ref_i4 = (I4) obj_a;
A ref_a = (A) obj_d;
B ref_b = (B) obj_c;
C ref_c = (C) obj_d;
D ref_d = (D) obj_g;
E ref_e = (E) obj_d;
G ref_g = (G) obj_e;
```
Part III: Evaluating Code (3 points for each part a-e, 15 points total)

Given the following class definitions:

class Doh {
    public int m(String s, int i) {
        if (i < 20) {
            return m(s, i+1);
        } else {
            return m(s);
        }
    }

    public int m(String s) {
        return 100;
    }
}

class Ray extends Doh {
    public int m (String s) {
        return m(s, s.length());
    }

    public int m (int i, String s) {
        return m(s, i);
    }

    public int m (String s, int i) {
        if (i < 10) {
            return super.m(s, i);
        } else {
            return i - s.length();
        }
    }
}

class Mee extends Ray {
    public int m (String s, int i) {
        return s.length() + i;
    }

    public int m (int i1, int i2) {
        return super.m("Hello", i1+i2);
    }
}
Part III continued...

Suppose the following variables are defined:

\[
\begin{align*}
\text{Doh } & \quad d = \text{new Doh}(); \\
\text{Ray } & \quad r = \text{new Ray}(); \\
\text{Mee } & \quad m = \text{new Mee}();
\end{align*}
\]

What is the value of the following expressions:

a) \(m.m(1, 2)\) 
9

b) \(r.m(\text{"Turing"})\) 
4

c) \(m.m(4, \text{"UNC"})\) 
7

d) \(((\text{Doh}) \ r).m(\text{"Brooks"})\) 
4

e) \(d.m(\text{"Brooks"})\) 
100
Part IV: Exceptions (3 points for each part a-e, 15 points total)

Given the following code defining four exception classes and two functions:

```java
class ExceptionA extends RuntimeException {}
class ExceptionB extends ExceptionA {}
class ExceptionC extends ExceptionA {}
class ExceptionD extends ExceptionB {}

class Bar {
    public static int bar(int x) {
        switch (x) {
            case 1:
                throw new ExceptionA();
            case 2:
                throw new ExceptionB();
            case 3:
                throw new ExceptionC();
            case 4:
                throw new ExceptionD();
            default:
                return x*x;
        }
    }

class Foo {
    public static int foo(int y) {
        int x = 0;
        try {
            try {
                x += bar(y);
            } catch (ExceptionD ex_d) {
                x = -3;
            } catch (ExceptionC ex_c) {
                x += bar(y-1);
            } catch (ExceptionA ex_a) {
                try {
                    x += bar(y+3);
                } catch (ExceptionB ex_b) {
                    x += 3;
                }
                finally {
                    x += 3;
                }
            } catch (RuntimeException e) {
                x += 5;
            }
            finally {
                x += 3;
            }
        }
        return x;
    }
```

Page 8 of 14
Part IV continued...

What is the value of the following expressions:

a) \( \text{foo(1)} \)

9

b) \( \text{foo(2)} \)

31

c) \( \text{foo(3)} \)

8

d) \( \text{foo(4)} \)

0

e) \( \text{foo(5)} \)

28
Part V: Inheritance (20 points)

Refactor the following classes DumbPhone, Smartphone, and SatellitePhone employing inheritance as appropriate given the following guidelines:

- DumbPhone and Smartphone should be subclasses of a common parent class called CellPhone
- CellPhone and SatellitePhone should be subclasses of a common abstract parent class called Phone

Write your code on pages 13 and 14.

class DumbPhone {
    private String number;
    private String model;
    private Network net;

    public enum Network {GSM, CDMA, LTE};

    public DumbPhone(String number, String model, Network net) {
        this.number = number;
        this.model = model;
        this.net = net;
    }

    public String getNumber() {
        return number;
    }

    public int getMemorySize() {
        return 1;
    }

    public String getModel() {
        return model;
    }

    public Network getNetwork() {
        return net;
    }

    public String getName() {
        return getMemorySize() + "GB " +
        getModel() +
        "(" + getNetwork().toString() + ")";
    }
}
class SmartPhone {
    private String number;
    private String model;
    private int memory_size;
    private Network net;
    private String processor;

    public enum Network {GSM, CDMA, LTE};

    public SmartPhone(String number, String model,
            int memory_size, String processor,
            Network net) {
        this.number = number;
        this.model = model;
        this.memory_size = memory_size;
        this.processor = processor;
        this.net = net;
    }

    public String getNumber() {
        return number;
    }

    public String getModel() {
        return model;
    }

    public int getMemorySize() {
        return memory_size;
    }

    public Network getNetwork() {
        return net;
    }

    public String getName() {
        return getMemorySize() + "GB " +
                getModel() +
                "(" + getNetwork().toString() + ")";
    }

    public String getProcessor() {
        return processor;
    }
}
class SatellitePhone {
    private String number;
    private String model;
    private int memory_size;

    public SatellitePhone(String number, String model, int memory_size) {
        this.number = number;
        this.model = model;
        this.memory_size = memory_size;
    }

    public String getNumber() {
        return number;
    }

    public String getModel() {
        return model;
    }

    public int getMemorySize() {
        return memory_size;
    }

    public String getRadioBand() {
        return "L";
    }

    public String getName() {
        return "Satellite Phone";
    }
}

Page 12 of 14
Model Answer:

abstract class Phone {
    private String number;
    private String model;

    public Phone(String number, String model) {
        this.number = number;
        this.model = model;
        this.memory_size = memory_size;
    }

    public String getNumber { return number; }  
    public String getModel { return model; }  

    abstract public int getMemorySize();
    abstract public String getName();
}

class SatellitePhone extends Phone {
    private int memory_size;

    public SatellitePhone(String number, String model, 
                           int memory_size) {
        super(number, model);
        this.memory_size = memory_size;
    }

    public int getMemorySize() { return memory_size;
    }

    public String getRadioBand() { return "L";
    }

    public String getName() { return "Satellite Phone";
    }
}
class CellPhone extends Phone {
    private Network net;

    public enum Network {GSM, CDMA, LTE};

    public CellPhone(String number, String model, 
            Network net) {
        super(number, model);
        this.net = net;
    }

    public Network getNetwork() { return net; }

    public String getName() {
        return getMemorySize() + "GB " + 
                getModel() + "(" + getNetwork().toString() + ")";
    }
}

class DumbPhone extends CellPhone {

    public DumbPhone(String number, String model, 
            Network net) {
        super(number, model, net);
    }

    public int getMemorySize() { return 1; }
}

class SmartPhone extends CellPhone {

    private int memory_size;
    private String processor;

    public SmartPhone(String number, String model, 
            int memory_size, String processor, 
            Network net) {
        super(number, model, net);
        this.memory_size = memory_size;
        this.processor = processor;
    }

    public int getMemorySize() { return memory_size; }

    public String getProcessor() { return processor; }
}