University of Maryland College Park
Dept of Computer Science
CMSC132 Fall 2017
Midterm I Key

Last Name (PRINT): ________________________________________________________________

First Name (PRINT): ______________________________________________________________

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

Grader Use Only

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Problem #1 (Miscellaneous)</td>
<td>38</td>
</tr>
<tr>
<td>#2</td>
<td>Problem #2 (Class Implementation)</td>
<td>162</td>
</tr>
<tr>
<td>#3</td>
<td>Honors</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>Total</strong></td>
<td><strong>200/212</strong></td>
</tr>
</tbody>
</table>
Problem #1 (Miscellaneous)

1. (3 pts) Does the following class rely on encapsulation? If it does, write YES; otherwise NO and fix it. Notice the getName method returns a string and not a StringBuffer.

```java
public class Name {
    StringBuffer name;
    public String getName() {
        return name.toString();
    }
}
```

Answer: No, as name is not private.

2. (3 pts) What will take place when the following code fragment is executed?

```java
Object lc = null;
boolean val = lc instanceof Object;
```

a. An exception will be generated.
b. false will be assigned to val.
c. true will be assigned to val.
d. None of the above.

Answer: b.

3. (7 pts) A class called Hero is defined as follows:

```java
public class Hero {
    private int power;
    public int increasePower(double delta) {
        return power += (int) delta;
    }
}
```

Another class called SuperHero extends Hero and can define one of the methods below. Indicate what will happen if one of the methods below were to be added to the SuperHero class. Use RIDE to indicate the method will override increasePower, LOAD to indicate it will overload increasePower, and ERROR if it will generate a compilation error.

a. `public int increasePower(double delta) { return 4 + (int)delta; }` RIDE
b. `protected int increasePower(double delta) { return 4 + (int)delta; }` ERROR
c. `int increasePower(double delta) { return 4 + (int)delta; }` ERROR
d. `public double increasePower(double delta) { return 4.0 + delta; }` ERROR
e. `public int increasePower() { return power += 100; }` ERROR
f. `public double increasePower() { return power += 100.00; }` ERROR
g. `public increasePower() { return power += 100.00; }` ERROR

4. (3 pts) Would the Hero class have a default constructor if the following method where added to the class? Yes/No.

```java
public Hero(int powerIn) { power = powerIn; }
```

Answer: No
5. (3 pts) Assume the **Hero** and **SuperHero** classes above define a method with the following signature:

```java
public void origin()
```

If Java were using dynamic binding, which method (the one in **Hero** or in **SuperHero**) will be used when the following code is executed?

```java
Hero h = new SuperHero();
h.origin();
```

Answer: The one in **SuperHero**

6. (3 pts) For the code below, either write what value for x will printout or, if it will not compile, write **will not compile**.

```java
public class Exam1 {
    int x = 5;
    
    public Exam1() { x = 6; }
    { x = 7; }
    public static void main(String[] args) {
        System.out.println(new Exam1().x);
    }
}
```

Answer: It compiles and the output is 6

7. (3 pts) You are given two classes A and B without no information about them except they work together in order to solve a problem. Which of the following alternatives would you use? Briefly explain. Shorter, brief, explanations will receive more credit.

```java
// Alternative One
public class B extends A {
}

// Alternative Two
public class B {
    A a = new A();
}
```

Answer: Alternative Two as we prefer composition over inheritance.

8. (3 pts) A method called **sequence** throws a checked exception. How would you update the method so we don’t have to add a try/catch block?

```java
public void sequence(){
    // code that throws checked exception
}
```

Answer: By adding a `throws Exception` after sequence(). They will get full credit as long as they mention `throws` needs to be added after sequence()

9. (3 pts) The signature of the clone method in the Object class is **protected Object clone()**. If you are overriding this method in a class called Hero, which of following is the preferred signature for the method in the Hero class?

a. public Object clone()
b. public Hero clone()
c. protected Object clone()
d. protected Hero clone()

Answer: b.
10. (7 pts) Define a class called `StringComp` that implements the Comparator interface and compares String objects. The class will allow us to sort strings based on the string’s lengths (shorter strings will appear first after sorting an ArrayList of strings).

Answer:

```java
public class StringComp implements Comparator<String> {
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

**Problem #2 (Class Implementation)**

For this problem you will complete implementation of two classes, `Engine` and `GasEngine`, whose partial definition is provided below. An Engine has a make, mileage, and suppliers that manufacture the engine. There is one supplier designated as the worst supplier for that engine. A `GasEngine` keeps track of the octane required by the engine. All methods of both classes are non-static unless specified otherwise. On the next page you will see a sample driver that illustrates the functionality of the classes you need to implement. Feel free to ignore it if you know what to implement. Also feel free to ignore the ElectricEngine class you see (you do not have to implement it). **Students in the honor section: there is another question for you at the end of the exam.**

```java
public abstract class Engine implements Iterable<String>, Comparable<Engine> {
    protected String make;  
    private int mileage, numSuppliers;
    private String worstSupplier, suppliers[];  
    private static int MAX_SUPPLIERS = 10;
}
```

```java
public class GasEngine extends Engine {
    private int octane;
}
```

**Engine Class Methods**

1. (18 pts) Define a constructor that takes as parameters make, mileage, and worstSupplier as parameters initializing the corresponding instance variables. The method will create a String array of length MAX_SUPPLIERS initializing the number of suppliers to 0.

Answer:

```java
public Engine(String make, int mileage, String worstSupplier) {
    this.make = make;  
    this.mileage = mileage;
    this.worstSupplier = worstSupplier;
    suppliers = new String[MAX_SUPPLIERS];
    numSuppliers = 0;
}
```

2. (4 pts) Define a method named `getMileage` that returns the mileage.

Answer:

```java
public int getMileage() { return mileage; }
```
3. (18 pts) Define a method named `addSupplier` that adds a supplier to the suppliers array. If the argument is `null`, the exception `IllegalStateException` will be thrown with the message “invalid supplier” and no further processing will take place. The method returns a reference to the current object. You can assume we will not add more than `MAX_SUPPLIERS`.

Answer:

```java
public Engine addSupplier(String supplier) {
    if (supplier == null) {
        throw new IllegalArgumentException("invalid supplier");
    }
    suppliers[numSuppliers++] = supplier;
    return this;
}
```

4. (10 pts) Provide any method(s) necessary for the implementation of the `Comparable<Engine>` interface. Engines will be compared based on mileage and those with a higher mileage will appear first if we were to sort a list of engines.

Answer:

```java
public int compareTo(Engine engine) {
    return engine.mileage - mileage;
}
```

5. (46 pts) Using anonymous inner classes, implement any method(s) necessary for the implementation of the `Iterable<String>` interface. The iterator will allow us to iterate over the suppliers associated with an engine. The iterator will return the next supplier that is different than `worstSupplier` (the iterator will skip the `worstSupplier`). You can assume only one `worstSupplier` has been added to the list.

Answer:

```java
public Iterator<String> iterator() {
    return new Iterator<String>() {
        private int curr = 0;
        public boolean hasNext() {
            while (curr < numSuppliers) {
                if (!suppliers[curr].equals(worstSupplier)) {
                    return true;
                }
                curr++;
            }
            return false;
        }
        public String next() {
            return suppliers[curr++];
        }
    };
}
```

6. (4 pts) Define an abstract method named `start()` that has void as return type.

Answer:

```java
public abstract void start();
```
GasEngine Methods

1. (12 pts) Define a constructor that takes as parameters the make, mileage, worstSupplier, and octane as parameters initializing the corresponding instance variables.

   Answer:
   ```java
   public GasEngine(String make, int mileage, String worstSupplier, int octane) {
       super(make, mileage, worstSupplier);
       this.octane = octane;
   }
   ```

2. (10 pts) Define a constructor that takes as parameters the make and mileage as parameters, initializing the corresponding instance variables. The worstSupplier value will be “DSUPPLIER” and octane will be set to 0. You must rely on the previous constructor to implement this one.

   Answer:
   ```java
   public GasEngine(String make, int mileage) {
       this(make, mileage, "DSUPPLIER", 0);
   }
   ```

3. (4 pts) Define a method named getOctane that returns the octane.

   Answer:
   ```java
   public int getOctane() { return octane; }
   ```

4. (4 pts) The start method will just have the following code: System.out.println("Starting gas engine: " + make);

   Answer:
   ```java
   public void start() {
       System.out.println("Starting gas engine: " + make);
   }
   ```
Utilities Class Static Method

(32 pts) An Utilities class defines the method `getSortedByMileage` that has the signature below. The method will return an ArrayList with GasEngines that have at least an octane value that corresponds to `minimum_octane`. The ArrayList returned must be sorted by mileage. Feel free to use Collections.sort to sort your data. **Notice that the engines array parameter can have engines other than GasEngines.**

Answer:

```java
public static ArrayList<GasEngine> getSortedByMileage(ArrayList<Engine> engines,
                        int minimum_octane) {
    ArrayList<GasEngine> answer = new ArrayList<>();
    for (Engine engine : engines) {
        if (engine instanceof GasEngine) {
            GasEngine gasEngine = (GasEngine) engine;
            if (gasEngine.getOctane() >= minimum_octane) {
                answer.add(gasEngine);
            }
        }
    }
    Collections.sort(answer);
    return answer;
}
```

HONORS

1. Students in the Honor’s section must answer this question.
2. Only students in the Honor’s section will receive credit for answering this question.

Implement the clone method for the Engine class previously described. The returned object should such that modifications to this object will not affect the original one. You will lose credit if you use deep copy when it is not necessary.

Answer:

```java
public Engine clone() {
    Engine obj = null;
    try {
        obj = (Engine) super.clone();
        String [] newArray = new String[MAX_SUPPLIERS];
        for (int i = 0; i < numSuppliers; i++) {
            newArray[i] = suppliers[i];
        }
        obj.suppliers = newArray;
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
    return obj;
}
```