King Fahd University of Petroleum & Minerals
College of Computer Science and Engineering
Information and Computer Science Department
ICS 201 – Introduction to Computing II
Spring Semester 2012-2013 (122)

SOLUTION to Major Exam 01

28th February 2013

Time: 120 minutes

Name: _____ StudentID: _____

This exam consists of four questions. All questions must be answered.

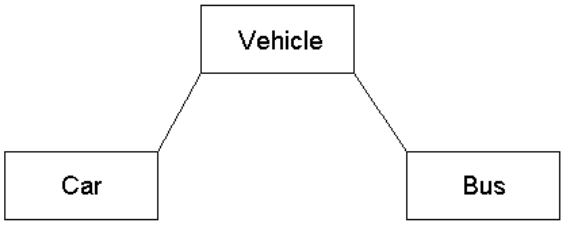
Question#	Max Marks	Marks Obtained
1	10 * 2 = 20	
2	30	
3	20	
4	30	
Total	100	

Q. 1 [2*10 = 20 marks] For each of the following statements, write your answer in the space provided:

Question	Ans
1. Which of the following is the correct way of defining an abstract method: A. **public abstract void myMethod(Object o) {}** B. **public abstract void myMethod(Object o);** C. **public void myMethod(Object o) {}** D. **public void myMethod(Object o);**	B
2. Which of the following can be a member of an interface: A. **public static final double MY_PI = 3.1416;** B. **private static final double MY_PI = 3.1416;** C. **protected static final double MY_PI = 3.1416;** D. **static double MY_PI = 22/7;**	A
3. Which of the following statements will return false (Assume **String x = "201", String y = "102", Object o = new Integer(201)**) A. **x instanceof String** B. **x instanceof Object** C. **x.getClass().equals(y.getClass())** D. **x.getClass().equals(o.getClass())**	D
4. Which of the following is not inherited in a derived class from a base class: A. public instance variables B. private instance variables C. public methods D. private methods	D
5. A call to **super()** in the constructor of the derived class must be A. the first statement in the constructor B. placed anywhere in the constructor C. the last statement in the constructor D. placed before a call to **this()** in the constructor	A
6. In Java every class is a descendant of the class A. super B. Java C. Object D. Class	C

7. When a method in a derived class has a different signature but the same name and return type from the method in the base class, that mechanism is called A. overloading B. encapsulation C. overriding D. shadowing	A
8. Given two classes **Letter** and **Alphabet**, where Alphabet is a derived class of Letter, which of the following is illegal statement: A. **Letter** x = new **Alphabet**(); B. **Alphabet** y = new **Letter**(); C. **Letter** x = new **Letter**(); D. **Alphabet** y = new **Alphabet**();	B
9. If a base class has a method **public Object myWork**(**Object o**), then it can be overridden by which of the following methods: A. **public Object myWork(String x)** B. **public String myWork(Object o)** C. **public String myWork(String x)** D. **public Object myWork()**	B
10. An Anonymous Inner Class has A. no constructors B. no instance variables C. no private methods D. no derived methods	A

Q. 2 [**30 marks**] Suppose we want to write an application that maintains an inventory of automobile vehicles of different types. Assume that we have the following inheritance hierarchy to implement:



The following are the descriptions of these classes:

(a) **Vehicle**: is an ***abstract class*** with the following details:
   a. has a private instance variable **weight** of type double
   b. an accessor method for **weight**
   c. an abstract method **computeMileage()** that is expected to compute and return the mileage of the vehicle as a double value
   d. a method **remainingDistance(double fuel)** that returns the remaining travel distance, as the product of the vehicle's mileage and the amount of fuel remaining (which is given as a parameter)

(b) **Car**: is a ***non-abstract class*** that extends the **Vehicle** class as follows:
   a. has a private instance variable **hybrid** of type boolean initialized to false
   b. a constructor to initialize the instance variables with values given as parameters
   c. the method **computeMileage** as follows:
      i) If the car is a **hybrid**, then the mileage = 20 * (1000/**weight**)
      ii) If the car is not a **hybrid**, then the mileage = 6.5 * (1000/**weight**)

(c) **Bus**: is a ***non-abstract class*** that extends the Vehicle class as follows:
   a. has a private instance variable **passengers** of type int
   b. a constructor to initialize the instance variables with values given as parameters
   c. the method **computeMileage** as:

      mileage = 10 * (2000/**weight**) - (**passengers** * 0.2)

(d) **Main**: a class with the main method to test the above classes as follows: Define an array of type Vehicle of size 3 having the following objects:
      i) a hybrid Car object, with weight 1600
      ii) a non-hybrid Car object, with weight 1400
      iii) a Bus object with weight 2400 and 10 passengers

For each object in the array, print the remaining distance when the remaining fuel amount = 20.

```
abstract class Vehicle {
    private double weight;

    // 1. Define an initializing constructor to initialize the
    // instance variable with a value given as a parameter
    public Vehicle(double weight) {
        this.weight = weight;
    }

    // 2. Define an accessor method for weight
    public double getWeight() {
        return weight;
    }

    // 3. Define an abstract method "computeMileage()" that is expected
    // to compute and return the mileage of the vehicle as a double value
    public abstract double computeMileage();

    // 4. Define a method "remainingDistance(double fuel)" that returns the
    // remaining travel distance, as the product of the vehicle's mileage
    // and the amount of fuel remaining (which is given as a parameter)
    public double remainingDistance(double fuel) {
        return this.computeMileage() * fuel;
    }
}

class Car extends Vehicle {
    private boolean hybrid = false; //whether it's a hybrid or not

    // 5. Define an initializing constructor to initialize the instance
    // variables with values given as parameters
    public Car(double weight, boolean hybrid){
        super(weight);
        this.hybrid = hybrid;
    }

    // 6. Define the method "computeMileage" as follows:
    // - If the car is a hybrid, then the mileage = 20 * (1000/weight)
    // - If the car is not a hybrid, then the mileage = 6.5 * (1000/weight)
    public double computeMileage() {
      if (hybrid)
            return 20.0 * (1000 / getWeight());
      else
            return 6.5 * (1000 / getWeight());
    }
}
```

```
class Bus extends Vehicle {
    private int passengers;  // number of passengers aboard

    // 7. Define an initializing constructor to initialize the instance
    // variables with values given as parameters
     public Bus(double weight, int passengers) {
        super(weight);
        this.passengers = passengers;
    }


    // 8. Define the method "computeMileage" as:
    // mileage = 10 * (2000/weight) - (passengers * 0.2)
    public double computeMileage() {
        return 10 * (2000 / getWeight()) - (passengers * 0.2);
    }
}

public class VehicleTest {
    public static void main(String[] args) {

     // 9. Define an array of type Vehicle of size 3 having the following
     // objects:
     //   - a hybrid Car object, with weight 1600
     //   - a non-hybrid Car object, with weight 1400
     //   - a Bus object with weight 2400 and 10 passengers
      Vehicle[] vehicle = new Vehicle[3];
      vehicle[0] = new Car(1600, true);
      vehicle[1] = new Car(1400, false);
      vehicle[2] = new Bus(2400, 10);


     // 10. For each object in the array, print out the remaining distance
     // when the remaining fuel amount is 20
      System.out.println("vehicle[0]: " + vehicle[0].remainingDistance(20));
      System.out.println("vehicle[0]: " + vehicle[1].remainingDistance(20));
      System.out.println("vehicle[0]: " + vehicle[2].remainingDistance(20));
    }
}
```

Q. 3 [**20 marks**] Consider the following interface:

```
interface NumberAsString {
        public int realPart();
        public int fractionalPart();
        public boolean isInteger();
        public NumberAsString roundedProduct(NumberAsString s2);
}
```

Design and implement a class **DoubleAsString** that implements the interface **NumberAsString**. The class **DoubleAsString** should have a string **value** as the instance variable. The method **realPart()** should return the real part of the double precision number represented by the string **value**. The method **fractionalPart()** should return the fractional part of the double precision number represented by the string **value**. The method **roundedProduct**(**NumberAsString s2**) should round off the two doubles (**this** and **s2**) and return their product. Include a **toString**() method also.

You may use the methods **Integer.parseInt(String val)** and **Double.parseDouble(String val)**.
Do **not** use any methods from the **Math** class.

For example the following main method code can be executed in the main class,

```
DoubleAsString s1 = new DoubleAsString("3.1416");
DoubleAsString s2 = new DoubleAsString("6.52");
DoubleAsString s3 = new DoubleAsString("7000.0");

System.out.println("For "+s1+", real = "+s1.realPart()+", frac =
"+s1.fractionalPart());
System.out.println("Is s3: "+s3+" an integer? "+s3.isInteger());
System.out.println("Rounded Product of "+s1+" and "+s2+" is "
+s1.roundedProduct(s2));
```

The output is as follows:

```
For 3.1416, real = 3, frac = 1416
Is s3: 7000.0 an integer? true
Rounded Product of 3.1416 and 6.52 is 21
```

```
class DoubleAsString implements NumberAsString {
        private String number;

        public DoubleAsString(String x) {
                this.number = x;
        }

        public int realPart() {
                return Integer.parseInt(number.substring(0, number.indexOf('.')));
        }

        public int fractionalPart() {
                return Integer.parseInt(number.substring(number.indexOf('.') + 1));
        }

        public boolean isInteger() {
                return (fractionalPart() == 0);
        }

        public NumberAsString roundedProduct(NumberAsString s2) {
                int val1, val2, firstDigit;
                firstDigit = Integer.parseInt((this.fractionalPart() + "").substring(0, 1));

                if(firstDigit < 5)
                        val1 = this.realPart();
                else
                        val1 = this.realPart() + 1;

                firstDigit = Integer.parseInt((s2.fractionalPart() + "").substring(0, 1));
                if(firstDigit < 5)
                        val2 = s2.realPart();
                else
                        val2 = s2.realPart() + 1;

                System.out.println(val1 + " "+ val2);
                return new DoubleAsString(val1 * val2 + "");
        }

        public String toString() {
                return number;
        }
}
```

Q. 4 [10+10+10 = 30 marks] What is the output of the following programs:

(a)

```
public class OuterOne {
   private int x;

   public class InnerOne {
      private int y;

      public InnerOne(int y) {
         this.y = y*y*y;
      }

      public InnerOne() {
           this(2);
           x = 6;
      }

      public void innerMethod() {
        System.out.println("Outer x is "+x);
        System.out.println("y is "+y);
      }
   }

   public OuterOne(int x) {
      this.x = x*x;
   }

   public void outerMethod() {
     System.out.println("x is " + x);
   }

   public void makeInner() {
     InnerOne anInner = new InnerOne();
     anInner.innerMethod();
   }
 public static void main(String args[]) {
   OuterOne o = new OuterOne(3);
   OuterOne.InnerOne i = o.new InnerOne(4);
   i.innerMethod();
   o.outerMethod();
   o.makeInner();
 }
}
```

```
Outer x is 9
y is 64
x is 9
Outer x is 6
y is 8
```

(b)

```
class Base {
  public Base ( ) {
    System.out.println("Base Constructor");
  }
  public void m1( ) {
    m2( );
    m3( );
  }
  public void m2( ) {
    System.out.println("Base m2");
  }
  public static void m3( ) {
    System.out.println("Base m3");
  }
}

class Child extends Base {
  public Child( ) {
    System.out.println("Child Constructor");
  }
  public void m2( ) {
     System.out.println("Child m2");
  }
  public static void m3( ) {
    System.out.println("Child m3");
  }
}

class Test {
  public static void main(String[] args) {
    Base b = new Base( );
    b.m1( );
    Child c = new Child( );
    c.m1( );
    b = c;
    b.m1( );
  }
}
```

```
Base Constructor
Base m2
Base m3
Base Constructor
Child Constructor
Child m2
Base m3
Child m2
Base m3
```

(c)

```
class Shoe {
      public Shoe() {
            this("This is a shoe");
            System.out.println("Base Class");
      }
      public Shoe(String s) {
            System.out.println(s);
}}

class TennisShoe extends Shoe {
      public TennisShoe(){
            this("This is a Tennis Shoe");
            System.out.println("Derived Class");
      }
      public TennisShoe(String s) {
            super("Exam 1");
            System.out.println(s);
}}

class WhiteTennisShoe extends TennisShoe {
      public WhiteTennisShoe(String s) {
            System.out.println(s);
}}

class Test {
      public static void main(String args[]) {
            new WhiteTennisShoe ("A white tennis shoe is created");
}}
```

```
Exam 1
This is a Tennis Shoe
Derived Class
A white tennis shoe is created
```