King Fahd University of Petroleum & Minerals
College of Computer Science and Engineering
Information and Computer Science Department
ICS 201 – Introduction to Computing II
Summer Semester 2011-2012 (113)

SOLUTION to Major Exam 01

26th June 2012

Time: 100 minutes

Name: _____ StudentID: _____

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

Question#	Max Marks	Marks Obtained
1	1.5 * 18 = 27	
2	30	
3	20	
4	23	
Total	100	

Q. 1 [1.5*18 = 27 marks] For each of the following statements, indicate whether TRUE or FALSE

Question	Ans
1. A derived class inherits all the public methods, all the public and private instance variables and all the public and private static variables from the base class.	***TRUE***
2. A derived class inherits private methods from the base class.	***FALSE***
3. A final method of a base class cannot be over-ridden in a derived class.	***TRUE***
4. If a base class has a method **public Object clone()**, then it can be overridden by a method **public String clone()** in a derived class.	***TRUE***
5. A **final** class cannot be extended (cannot have derived classes).	***TRUE***
6. The access permission of an overridden method can be changed from private in the base class to public in the derived class.	***TRUE***
7. A call to **super()** in the constructor of the derived class must be the last statement.	***FALSE***
8. A constructor definition can contain an invocation of **this(…)** and **super(…)**, with the invocation of **this(…)** before the invocation of **super(…)**.	***FALSE***
9. Given two classes **Letter** and **Alphabet**, where **Alphabet** is a derived class of **Letter**, the following is a legal statement: **Letter x = new Alphabet()**;	***TRUE***
10. Given two classes **Letter** and **Alphabet**, where **Alphabet** is a derived class of **Letter**, the following is a legal code: **Alphabet y = new Alphabet()**; **Letter x = (Letter) y**;	***TRUE***
11. A **protected** method has a wider access as compared to a default (or package) access.	***TRUE***
12. Within the definition of a method in a derived class, the following code is illegal: **return super.super.toString();**	***TRUE***
13. In Java every class is a descendant of the class Object.	***TRUE***
14. The method **public boolean equals (Employee e)** overrides the method **public boolean equals(Object o)**	***FALSE***
15. What is the output of the following code: **return (new String("abc").getClass() == new String("abc").substring(2).getClass());**	***TRUE***
16. If a class contains at least one abstract method, then it must be declared abstract.	***TRUE***
17. All methods defined in an interface are public, static and abstract.	***FALSE***
18. A non-static inner class can be initialized without creating an object of the outer class.	***FALSE***

Q. 2 [**30 marks**] You have a car company that *sells cars* as well as *rents them* out to customers. In order to keep track of each car sold or rented out, the following abstract class must be implemented.

```
abstract class Car
{
    private String name;

    public abstract double total_cost();
    public String toString() { return name; }
}
```

Design and implement suitable classes for modeling car sale and car rental.

(a) **[10 marks]** ***CarSale***: Each car sold has a base price. If a car is sold on down-payment, the `total_cost()` method should return its `base_price`. If a car is sold on **monthly installments** payable every year, the `total_cost()` should add 10% to the base_price of the car per year. Your `toString()` method should print the name of the car, the mode of payment (down-payment or installments), the number of installments (use 1 for down-payment), the amount per installment and the total_cost. (Note that there are 12 monthly installments in a year. To calculate the price per installment, divide the total_cost by the number of installments. For down-payment, the number of years is zero). The instance variables for this class are **base_price** and **years**.

(b) **[10 marks]** ***CarRental***: Each car rented out has a base_rent. If a car is rented for one day only, the `total_cost()` method should return the base_rent only. If a car is rented out for several **days**, your class should calculate the `total_cost()` by multiplying the number of days by the base_rent. Your `toString()` method should print the name of the car, the number of days rented, base_rent and the total_cost. The instance variables for this class are **base_rent** and **days**.

(c) **[10 marks]** Test your program by making a test class. Use an array of cars having the following objects:

car[0]: Toyota, base_price: 55,000, payment-mode: installments, number of years = 2,
car[1]: Mazda, down-payment, price: 60,000.
car[2]: Nissan, car-rental, base_rent = 100/day, number of days = 30.

Use the `toString()` method to print the `total_cost` of each car, and the combined cost of all cars..

```
class CarSale extends Car {
	private double base_price;
	private int years;

	public CarSale(String name, double bp, int y) {
		super(name);
		base_price = bp;
		years = y;
	}

	public double total_cost() {
		if(years == 0)
			return base_price;
		else
			return base_price + years * 0.1 * base_price;
	}

	public String toString() {
		String mode;
		int installments;
		if(years == 0) {
			mode = "Down_Payment";
			installments = 1;
		}
		else {
			mode = "Installments";
			installments = 12*years;
		}

		return super.toString() + ", " + mode + ", # of installments = "+installments+
			", Amount/Installment = "+total_cost()/installments+ ", Total Cost = "+total_cost();
	}
}

class CarRental extends Car {
	private double base_rent;
	private int days;

	public CarRental(String name, double br, int d) {
		super(name);
		base_rent = br;
		days = d;
	}
```

```
	public double total_cost() {
		return base_rent * days;
	}

	public String toString() {
		return super.toString() + ", # of days rented = "+days+
			", Base Rent = "+base_rent+ ", Total Cost = "+total_cost();
	}
}

public class CarTest {
	public static void main(String[] args) {
		Car[] c = new Car[3];
		double total_cost = 0;
		c[0] = new CarSale("Toyota", 55000, 2);
		c[1] = new CarSale("Mazda", 60000, 0);
		c[2] = new CarRental("Nissan", 100, 30);

		for(int i = 0; i < c.length; i++) {
			System.out.println(c[i]);
			total_cost += c[i].total_cost();
		}

		System.out.println("Total Cost for all Cars = "+total_cost);
	}
}
```

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

```
interface SpecialNumber {
      double realValue();
      SpecialNumber simplify();
}
```

Design and implement a class Fraction that implements the interface **SpecialNumber**. Each fraction should have a **numerator** (integer) and a **denominator** (integer). The method **realValue()** should return the decimal value of the fraction as a double. The method **simplify()** should return a fraction in its simplest form by removing common factors from the numerator and the denominator. Include a **toString**() method also.

For example the following code can be executed in the main class,
SpecialNumber s = new Fraction(2, 4);
System.out.println(s + " = " s.simplify() + " = " + s.realValue()); ***//Output is 2/4 = 1/2 = 0.5***

```
class Fraction implements SpecialNumber {
        int num, den;

        public Fraction(int n, int d) {
                num = n; den = d;
        }

        public double realValue() {
                return 1.0*num/den;
        }

        public SpecialNumber simplify() {
                int factor = Math.min(num, den); //Assuming both are positive
                for(; factor > 1; factor--) {
                        if((num % factor == 0) && (den % factor == 0)) {
                                num /= factor; den /= factor;
                                break; //No need for it too!
                        }
                }
                return this; //Alternatively return new Fraction(num, den);
        }

        public String toString() {
                return num + "/" + den;
        }
}
```

Q. 4 [11+6+6 = 23 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(1);
         x = 4;
    }

    public void innerMethod() {
      System.out.println("enclosing 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(5);
  OuterOne.InnerOne i = o.new InnerOne(3);
  i.innerMethod();
  o.outerMethod();
  o.makeInner();
 }
}
```

```
enclosing x is 25
y is 27
x is 25
enclosing x is 4
y is 1
```

(b) class Test { **private static int i=0;** **public Test ( ) {** **i++;** **}** **public static void main (String[] args) {** **Test t=new Test();** **t.i=t.i+1;** **System.out.print(t.i);** **System.out.println(new Test( ).i);** **}** **}**	**23**
(c) class Parent { **Parent ( ) { }** **Parent (int x, int y) {** **System.out.println("Created Parent");** **}** **}** **class Child extends Parent {** **Child( ) { }** **public Child (int x, int y) { }** **public Child (int x, int y, int z) {** **this(x, y);** **System.out.println("Created Child");** **}** **public static void main (String[] args) {** **Parent c= new Child(1,2,3);** **}** **}**	**Created Child**