

COMP204-08B Test 1

11 September 2008

First name: _____ Last name: _____

ID Number: _____

Instructions

1. Write your name and ID number into the spaces provided above.
2. There are 4 questions of equal value.
3. Time allowed is 50 minutes.
4. This test is open book; but NO computers, pdas, cell phones, or similar, are allowed.
5. Write your answers in the spaces provided. Do not use your own paper! There is a blank page at the end of the test that you can use. You can use the backs as well.

Question 1: Java Trivia

Circle the correct answer. All statements are about the Java language and its libraries.

- The Collections class provides utility methods like `sort()` for other collection classes. YES
- Class `RandomAccessFile` is a subclass of class `InputStream`. NO
- Private data fields can only be accessed by the defining class and all subclasses. NO
- Generic types reduce the number of casts needed in your code. YES
- Streams can represent files on disk, but also byte arrays in memory. YES
- If `"a.equals(b)"` returns true, then `"b.equals(a)"` must return true as well. YES
- If `"a.equals(b)"` and `"b.equals(c)"` return, then `"a.equals(c)"` must true as well. YES
- Java supports multiple inheritance. NO
- Class `Object` provides utility methods (e.g. `hashCode()`) for all other classes. YES
- Any object in Java can be serialized. NO
- The Collection classes can be used with primitive types (e.g. `List<int>`). NO
- If class B extends A, then `List<A> people = new ArrayList();` is valid Java. NO
- `int hashCode() {return 0;}` is inefficient, but ok regarding constraints on hashing. YES
- Different objects must have different hash codes. NO
- The `clone()` method produces a deep copy of an object. NO
- Using interfaces to declare types make your code more flexible. YES
- Java has automatic garbage collection, and therefore no memory leaks. NO
- `Equals` and `==` perform identical tests in Java. NO
- Reflection can be used to generate multi-level arrays at runtime. YES
- In Java wrapper classes exist for all primitive types. YES

Question 2: Constructor mess

What output will the main method of class TestAB below produce?

```
class A {
    private String _name;
    public String getName() { return _name; }
    { System.out.println("code block in A");
      _name = "defaultA";
    }
    public A(String name) {
        _name = name;
        System.out.println("Constructor A yields " + this);
    }
    public String toString() { return "<A " + _name + ">"; }
}

class B extends A {
    private int _count;
    { System.out.println("code block in B");
      _count = 42;
    }
    public B(String name, int count) {
        super(name);
        _count = count;
        System.out.println("Constructor B yields " + this);
    }
    public String toString() { return "<B " + getName() + "_" + _count + ">"; }
}

public class TestAB {
    public static void main(String[] args) {
        A a = new A("someA");
        B b = new B("and_a_B", 13);
        System.out.println("done");
    }
}
```

OUTPUT:

```
code block in A
Constructor A yields <A  someA>
code block in A
Constructor A yields <B  and_a_B42>
code block in B
Constructor B yields <B  and_a_B13>
done
```

Question 3: Inheritance

Assume an Interface LineItem which describes a single item to make up an invoice:

```
public interface LineItem {
    double getPrice();
}
```

Also assume a Class Product which implements LineItem:

```
public class Product implements LineItem {
    private String description;
    private double price;
    public double getPrice() { return price; }
    public String toString() { return description; }
    public Product(String description, double price) {
        this.description = description;
        this.price = price;
    }
}
```

Your task is to implement a subclass DiscountedProduct. Its Constructor will take one more parameter, the discount (as a percentage, i.e. ranging from 0.0 to 100.0, non-inclusive). You will need to implement this constructor, and an appropriate getPrice method to return the discounted price, and a toString method that indicates the discount, e.g. "LOTR Complete Collection, Discount 33.0%" if the description was "LOTR Complete Collection", and the discount was 33.0.

```
public class DiscountedProduct extends Product{
    private double discount;

    public double getPrice() {
        return super.getPrice() * (1.0 - discount/100);
    }

    public String toString() {
        return super.toString() + ", Discount " + discount + "%";
    }

    public DiscountedProduct(String description, double price, double discount) {
        super(description,price);
        this.discount = discount;
    }
}
```

Question 4: Composition

Now use composition to implement a class for discounted products. Again provide an appropriate constructor, and a `getPrice()`, and a `toString()` method (which all follow the same specification as were given for the inheritance variant above). You must also decide which data fields this class will need. Note that the constructor has only two arguments, one which is already a `Product`, plus the discount for that product.

```
public class DiscountedItem implements LineItem {

    // add data fields as needed:
    private Product product;
    private double discount;

    public DiscountedItem(Product product, double discount) {
        this.product = product;
        this.discount = discount;
    }

    public double getPrice() {
        return product.getPrice() * (1.0 - discount/100);
    }

    public String toString() {
        return product.toString() + ", Discount " + discount + "%";
    }

    // any other methods, if you think any are needed:
    // no other methods needed.
}
```

Extra space for answering questions