**0. Name (2 pts):**

---

**CS 2334: Programming Structures and Abstractions**

**Midterm Exam**

Monday, October 5, 2009

*General instructions*:

- This examination booklet has 7 pages.
- Do not forget to write your name at the top of the page and to sign your name below.
- The exam is open book and notes, but closed electronic device.
- The exam is worth a total of 100 points (and 10% of your final grade).
- Explain your answers clearly and concisely. Do not write long essays (even if there is a lot of open space on the page). A question worth 5 points is only worth an answer that is at most 2 sentences.
- You have 50 minutes to complete the exam. Be a smart test taker: if you get stuck on one problem go on to the next. Don't waste your time giving details that the question does not request. Points will be taken off for answers containing excessive or extraneous information.
- Show your work. Partial credit is possible, but only if you show intermediate steps.

| Problem | Topic | Max | Grade |
|---|---|---|---|
| 0 | Name | 2 | |
| 1 | Inheritance | 30 | |
| 2 | Abstract Classes and Interfaces | 30 | |
| 3 | Generic Programming and Generics | 25 | |
| 4 | Abstract Data Types | 15 | |
| Total | | | |

---

On my honor, I affirm that I have neither given nor received inappropriate aid in the completion of this exam.

**Signature:** _____

**Date:** _____

1. **Inheritance** (30 pts)

Consider the following definition of four classes:

```
public class A
{
    protected String name;

    public A(String name){
        this.name = name;
    }

    public String toString(){
        return("A: " + name);
    };
}

public class B extends A
{
    public B(String name) {
        super("SUPER-B");
        this.name = name;
    }

    public String toString() {
        return("B: " + name);
    };
}

public class C extends B
{
    private String name;

    public C(String name){
        super("SUPER-C");
        this.name = name;
    };

    public String toString() {
        return("C: " + super.name);
    };
};

public class driver
{
    public static void main(String args[]) {
        A[] objects = new A[4];

        objects[0] = new A("foo");
        objects[1] = new B("bar");
        objects[2] = new C("baz");

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

(a) (15 pts) Draw the corresponding UML diagram. Include all variables, methods and relevant relations.

(b) (15 pts) What output does executing the driver class produce?

2. **Abstract Classes and Interfaces** (30 pts)

(a) (10 pts) Two classes, AA and BB share a common set of properties:

```
Property1 var1;
Property2 var2;
```

where **Property?** are other classes. However, there is one method defined for each class:

```
methodAA();     // For class AA
methodBB();     // For class BB
```

Draw the UML diagram that expresses the appropriate relationships between these classes. Include all relevant classes (including any that you need to invent), properties and methods.

(b) (10 pts) Two classes, XX and YY share a common set of method signatures:

```
void method1(Foo a);
double method2();
```

where **Foo** is a class. The method implementations and property sets are different for the two classes. Draw the UML diagram that expresses the appropriate relationships between these classes. Include all relevant classes (including any that you need to invent), properties and methods.

(c) (10 pts) Briefly describe the conditions under which you would implement an abstract class in place of an interface.

3. **Generic Programming and Generics** (25 pts)

Assume the following initialization of two instances of our GenericQueue that we developed in class:

```
GenericQueue<Number> queue1 = new GenericQueue<Number>(10);
GenericQueue<Integer> queue2 = new GenericQueue<Integer>(10);
```

(a) (15 pts) Indicate whether the Java compiler will accept each of the following lines. Briefly explain why or why not.

```
queue1.add(new Integer(5));


queue2.add(4.7);


queue1.add("foo");


queue2.add(3);


queue1.add(queue2.remove());

.
```

(b) (10 pts) True or False and explain: generic programming requires the use of **generics**.

4. **Abstract Data Types** (15 pts)

The **GenericQueue** that we implemented in class captures the notion of a "line" of objects: new objects are inserted at the end of the line and objects are removed from the beginning of the line. A *deque* stands for a "double ended queue" in which new objects can be added to either the end **or** beginning of the line. Furthermore, removed objects can come from either the end or beginning of the line.

As a reminder, here are the properties of GenericQueue (note that they are now *protected*):

```
public class GenericQueue<T>
{
    protected T list[];
    protected int front;      // Next object to return
    protected int back;       // Next slot to insert a new object

    :
```

Fill in the requested method implementation below.
Hints: the value of $-1\%N$ is $-1$. **GenericQueue.getNum()** returns the number of objects currently in the queue.

```
public class GenericDeque<T> extends GenericQueue<T>
{
    public GenericDeque(int size) {
        super(size);
    };

    // Add obj to the front of the queue
    //
    // Return = true if successful addition
    //        = false if the queue is full
    //
    // Post: If queue is not already full:
    //       1. Size is increased by one
    //       2. obj is in the array element indicated by the new front

    public boolean addFront(T obj) {
        // FILL IN IMPLEMENTATION HERE









    }
}
```