0. Name (2 pts):

CS 2334: Programming Structures and Abstractions

Midterm Exam

Wednesday, October 6, 2010

General instructions:

- This examination booklet has 7 pages.
- Do not forget to write your name at the top of this 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 one sentence.
- 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	Object Hierarchies	40	
2	Abstract Classes and Interfaces	25	
3	Generic Programming and Generics	15	
4	Abstract Data Types	20	
Total			

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

Signature: ______________________

Date: ______________________

1. **Object Hierarchies** (40 pts)

Consider the following definition of four classes:

```
public class A
{
    private String name;

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

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

public class B extends A
{
    protected String name;

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

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

public class C extends A
{
    private B b;

    public C(String name){
        super(name);
        b = new B("subB");
    };

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

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

      objects[0] = new A("foo");
      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?

(c) (10 pts) For each line of code below, indicate one of the following:

- the compiler will reject the line (**REJECT**),
- the compiler will not reject the line but there will be a run-time error (**RUN-TIME**), or
- the compiler will not reject the line and there will be no run-time error (**OK**).

```
A a1 = new A("foo");

C c = new C("baz");

A a2 = new C("bar");

B b2 = a1;

B b3 = c;

C c3 = (C) a1;
```

2. **Abstract Classes and Interfaces** (25 pts)

(a) (10 pts) Briefly explain the distinction between abstract methods defined by an abstract class and the methods defined by an interface.

(b) (5 pts) True or False and briefly explain: a method defined by an abstract class must be implemented by a child of that class.

(c) (10 pts) Give two reasons why *Comparable* is an interface and not an abstract class.

3. **Generic Programming and Generics** (15 pts)

Consider the following method prototypes and variable definitions:

```
public static <T> boolean find1(GenericQueue<T> q, T key);

GenericQueue<Number> a1 = new GenericQueue<Number>(5);
GenericQueue<Integer> a2 = new GenericQueue<Integer>(10);

Number v1 = new Integer(42);
Integer v2 = new Integer(24);
```

(a) (10 pts) Indicate whether the Java compiler will **REJECT** or **ACCEPT** each of the following lines. Briefly explain why or why not.

```
find1(a1, v2);



find1(a2, v1);



find1(a2, v2);

.
```

(b) (5 pts) True or False: The type parameters for generic classes are **only** checked at compile time.

4. **Abstract Data Types** (20 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.

```
public class GenericDeque<T> extends GenericQueue<T>
{
        // GenericDeque constructor

        public GenericDeque(int size) {
          // FILL IN IMPLEMENTATION




        };

        // Remove obj from the end of the queue
        //
        // Return = null if the queue is empty, or
        //        = the object at the end of the queue
        //
        // Post: If queue has an object in it, then:
        //      1. The number of objects in the queue is decreased by one
        //      2. The element at the back of the queue is removed
        //

        public T removeBack() {
          // Is the queue empty?
          if(isEmpty())  return null;

          // FILL IN IMPLEMENTATION








    }
}
```

Note: the above implementation has been changed to reflect the changes made on the board during the exam.