Name: Account:

433 Quiz, January 29th

1. Basic Java

   Consider the following interface:

```
interface Number {
  // all operations update the value stored in the instance and
  // return the updated value.
  Number add(int other);
  Number multiply(int other);
  Number square();
  Number abs();
}
```

   Write the following classes that implement the Number interface. Make sure to include constructors for both classes. You don't have to check the types of arguments to the Number methods.

   - Class 1: myInteger - Internally, value is represented as an int. Number operations use traditional integer arithmentic.

```
public class myInteger implements Number{
    int val ;

    myInteger (int v) {
      val = v;
    }

    public Number add(int other) {
      val += other;
      return this;
    }

    public Number multiply(int other) {
      val *= other;
      return  this;
    }

    public Number square() {
      val *= val;
      return this;
    }

    public Number abs() {
      if (val < 0) val = -val;
      return this;
    }
}
```

- Class 2: myModInteger - Internally, both value and base are represented as ints. You can assume that the base is greater than 0. Operations are computed modulo that base. For example, if an instance of myModInteger with value 2 and base 6 is added to another instance with value 5 (and base 6), the result is 1 (i.e., ((2 + 5) mod 6)).

```
public class myModInteger implements Number{
    int val, base;

    myModInteger (int v, int b) {
      base = b;
      val = v % base;
    }

    public Number add(int other) {
      val = (val + other) % base;
      return this;
    }

    public Number multiply(int other) {
      val = (val * other) % base;
      return  this;
    }

    public Number square() {
      val = (val * val) % base;
      return this;
    }

    public Number abs() {
      if (val < 0) val = -val;
      return this;
    }
}
```

Name: Account:

2. Java synchronization

   Design a simple one-place buffer for use in a multithreaded Java applications. The buffer can hold one object; when the buffer is initially allocated, it is empty. If a thread tries to take from an empty buffer, the thread blocks until it can return something. If a thread tries to put into a full buffer, the thread blocks until the buffer has been emptied and the request can be fulfilled.

   Your class should implement the following interface:

```
package cmsc433;
interface OnePlaceBuffer {
  Object take() throws InterruptedException;
  void put(Object o) throws InterruptedException;
  }
```

   In this solution, we assume that take() should never return null, and that doing put(null) is a no-op.

```
public class Buf implements cmsc433.OnePlaceBuffer {
  private Object data;

  public synchronized Object take() throws InterruptedException {
    while (data == null) wait();
    Object tmp = data;
    data = null;
    notifyAll();
    return tmp;
  }

  public synchronized void put(Object o) throws InterruptedException {
    while (data != null) wait();
    data = o;
    notifyAll();
  }
}
```