1 Short Answer Problems

1. (10 points) Explain the difference between the `==` operator and the `equals` method.

The `==` operator is almost always used to compare values that are primitive types. When comparing objects, one should almost always use `equals`.

Some examples:

```java
int x=1, y=1;
StringBuilder s1 = new StringBuilder("abc");
StringBuilder s2 = new StringBuilder("abc");

System.out.println(x == y); // prints true
System.out.println(s1 == s2); // prints false
System.out.println(s1.equals(s2)); // prints true
```

2. (15 points) What is the output of the following code?

```java
public class Problem2 {
    public static void tryToChange(int x, int[] y) {
        x = 10;
        y[0] = 20;
    }

    public static void main(String[] args) {
        int x=1, y[] = new int[2];
        y[0] = 1; y[1] = 2;
        tryToChange(x,y);
        System.out.print(x);
        for (int i=0; i<2; i++)
            System.out.println(" "+y[i]);
        System.out.println();
    }
}
```

The output is 1 20 2. This is because Java passes parameters by value. A copy of the value of the variable `x` is passed to `tryToChange`, so when this method changes `x`, it is changing the copy of the value. The same is true for `y`; however, an object variable stores a “reference” to the object (i.e., a pointer). So when `y` is passed to `tryToChange`, it’s a copy of the object.
of the pointer that is passed. Since the pointer refers to the same object, y is changed by
the assignment statement in tryTochange.

3. (15 points) In the for loop below, assume that x is a Bag of integers.

```java
for (int i : x)
    System.out.println(i);
```

Fill in the blanks in the while loop below so that it behaves in the same way as the for loop.

```java
Iterator<Integer> iter = __x.iterator____;
while (iter.___hasNext___)
    System.out.println(__i.next___);
```

### 2 Coding problems

```java
public static <T> boolean bagsAreEqual(Bag<T> bag1, Bag<T> bag2) {
    Iterator<T> i1 = bag1.iterator(),
                i2 = bag2.iterator();
    while (i1.hasNext() && i2.hasNext())
        if (!i1.next().equals(i2.next()))
            return false;
    if (i1.hasNext() || i2.hasNext())
        return false;
    return true;
}
```

```java
public static <T> void reverse(Stack<T> stack) {
    Queue<T> q = new Queue<T>();
    while (!stack.isEmpty())
        q.enqueue(stack.pop());
    while (!q.isEmpty())
        stack.push(q.dequeue());
}
```

```java
public static <T> Set<T> union(Set<T> set1, Set<T> set2) {
    Set<T> answer = new HashSet<T>();
    for (T i : set1)
        answer.add(i);
    for (T i : set2)
        answer.add(i);
    return answer;
}
```