Note: These solutions are straight from the Discussion 5 Recording and have some commented annotations.

1. IntList Iterator

```java
import java.util.Iterator;

/**
 * Modify IntList so that it can be used in a foreach loop.
 * For an example, refer to the test file.
 */
public class IntList implements Iterable<Integer> {
    int first;
    IntList rest;

    public IntList(int f, IntList r) {
        first = f;
        rest = r;
    }

    // Iterable:
    // iterator()

    // Iterator:
    // boolean hasNext()
    // Integer next()

    // SLL, DLL, IntList, LinkedListDeque
    public class IntListIterator implements Iterator<Integer> {
        IntList curr;

        public IntListIterator(IntList x) {
            curr = x;
        }


        public boolean hasNext() {
            return curr != null;
        }

        // IntList a = new IntList(3, null);
        // IntList b = new IntList(2, a);
        // IntList c = new IntList(1, b);
        // c = [1, 2, 3]
```

```
        // Iterator<Integer> it = new IntListIterator(c);
        // it.next() --> 1
        // it.next() --> 2
        // it.next() --> 3
        // curr --> null
        // it.next() --> Error

        public Integer next() {
            // Retrieve first element THEN move to the next
            int k = curr.first;
            curr = curr.rest;
            return k;
        }
    }

    public Iterator<Integer> iterator() {
        return new IntListIterator(this);
    }
}
```

2. LinkedListDeque Greater Than Iterator

Java

```
import static com.google.common.truth.Truth.assertThat;
import org.junit.jupiter.api.Test;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;

public class TestGreaterThanIterator {
    public static <T extends Comparable<T>> LinkedListDeque61B<T> of(T... nums)
{
        LinkedListDeque61B<T> d = new LinkedListDeque61B<>();
        for (T num : nums) {
            d.addLast(num);
        }
        return d;
    }

    // LLD = [42]
    // d.greaterThanIterator(10)
    // --> [42]
    // d.greaterThanIterator(50)
    // --> []

    @Test
    public void dequeOfIntegers() {
        LinkedListDeque61B<Integer> d = of(42, 2, 14, 7);

        // below, write a test that:
        // 1. Creates a greaterThanIterator that will iterate only over items
greater than 10.
        // 2. Uses the resulting iterator to create a list of items called
actual that are greater than 10.
        // 3. Creates a list of items called expected that is the list of items
that are expected (i.e. 42, then 14)
        // 4. Compares these two lists using assertThat.
        Iterator<Integer> it = d.greaterThanIterator(10);
        ArrayList<Integer> actual = new ArrayList<>();


        // it.hasNext() --> returns true or false depending on if there are any
elements left (greater than 10)
        // it.next() --> return elements greater than 10

        // for loop --> strictly counts (int i = 0; i < x; i++)
```

```
        // while loop --> we can check for a boolean condition
        while (it.hasNext()) {
            int x = it.next();
            actual.add(x);
        }

        List<Integer> expected = List.of(42, 14);
        assertThat(actual).isEqualTo(expected);
    }
}

---
public class LinkedListDeque61B<T extends Comparable<T>> {
   ...
   /** Complete this class! */
    public class GreaterThanIterator implements Iterator<T> {
        Node curr;
        T threshold;

        public GreaterThanIterator(T x) {
            // your code here!
            threshold = x;
            curr = sentinel.next;

            // We need to set curr to be at the first element greater than x
            advance();
        }

        /** Advance to the next element greater than x */
        private void advance() {
            // We should only be moving forward if our current element is <= x
            // item < threshold --> < 0
            // item = threshold --> = 0
            // item > threshold --> > 0
            while (curr != sentinel && curr.item.compareTo(threshold) <= 0) {
                curr = curr.next;
            }

            // We've reached sentinel (because the tail points to the
sentinel!)
        }

        @Override
        public T next() {
```

```
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            // your code here!
            // LinkedListDeque61B<Integer> d = of(42, 2, 14, 7);
            // Iterator<Integer> it = d.greaterThanIterator(10);
            // it.next() --> 42

            // Retrieve the current element (that must be greater than x)
            T k = curr.item;
            // We need to move to the next node in order to avoid comparing the
same item over and over and over...
            curr = curr.next;
            // Then move to the next element that's greater than x
            advance();
            // We need to set curr to be at the first element greater than x
            return k;
        }

        @Override
        public boolean hasNext() {
            // your code here!
            return curr != sentinel;
        }
    }

    /**
     * Returns an iterator that yields only the items that are greater than the
     * given item.
     *
     * For example, if the deque is [42, 2, 14, 7], greaterThanIterator(10)
should
     * yield 42, 14.
     */
    public Iterator<T> greaterThanIterator(T x) {
        // your code here!
        return new GreaterThanIterator(x);
    }
}
```

3. Students Comparator (Extra)

Java

```
import java.util.Comparator;

public class Student implements Comparable<Student> {
    // Step 2: Modify the line below, and add any other necessary code.
    public static final Comparator<Student> NAME_COMPARATOR = new
NameComparator();
    private String name;
    private int year;

    public Student(String name, int year) {
        this.name = name;
        this.year = year;
    }

    private static class NameComparator implements Comparator<Student> {
        public int compare(Student s1, Student s2) {
            return s1.name.compareTo(s2.name);
        }
    }

    @Override
    public int compareTo(Student other) {
        // Step 1: your code here!
        // If this < other: < 0
        // If this = other: = 0
        // If this > other: > 0

        // If this.year > other.year, we should return > 0
        return this.year - other.year;

        // If we have a list of students and we called .sort() in ascending
order
        // [2, 7, 6, 3, 5, 4, 1] -->
        // [1, 2, 3, 4, 5, 6, 7]
    }
}

// Comparable --> comparing this instance to an other instance --> defines the
expected or natural way to compare this object to other objects of its type
// --> compareTo(T other)
// Comparator --> comparing instance one to instance two --> defining a
alternative or different way to compare two objects (custom ways to compare two
objects)
```

```
// --> compare(T o1, T o2)
```