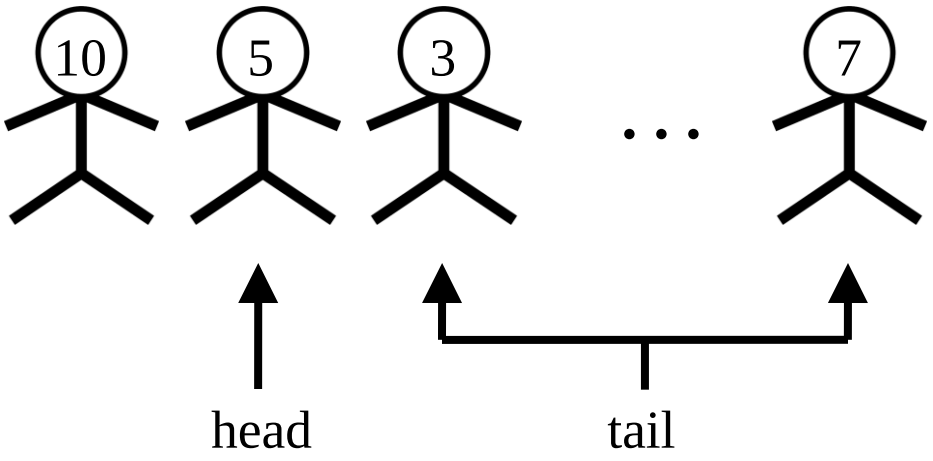
Discussion Section 2 (Thomas Feng, 09/04/2008)

```
/** Scheme-like pairs that can be used to form a list of
 *  integers. */
public class IntList {
    public int head;
    public IntList tail;

    /** A List with head and tail. */
    public IntList (int head, IntList tail) {
        this.head = head; this.tail = tail;
    }

    /** A List with null tail, and head = 0. */
    public IntList () { this (0, null); }
    // NOTE: public IntList () { }  would also work.

    /** A new IntList containing the ints in ARGS. */
    public static IntList list (Integer ... args) {
        IntList result, p;

        if (args.length > 0)
            result = new IntList (args[0], null);
        else
            return null;

        int k;
        for (k = 1, p = result; k < args.length; k += 1, p = p.tail)
            p.tail = new IntList (args[k], null);
        return result;
    }

    /** True iff X is an IntList containing the same sequence of ints
     *  as THIS. */
    public boolean equals (Object x) {
        if (! (x instanceof IntList))
            return false;
        IntList L = (IntList) x;
        IntList p;
        for (p = this; p != null && L != null; p = p.tail, L = L.tail){
            if (p.head != L.head)
                return false;
        }
        if (p != null || L != null)
            return false;
        return true;
    }

}
```



Facts:

IntList is a *class*. An *instance* of IntList is an *object*. We sometimes use the class name to refer to an instance, say, an IntList. Any object is an instance of the Object class.

IntList has two *fields*. The *type* of `head` is int, which is a *primitive data type* of Java. The type of `tail` is IntList itself. By defining these two fields, we declare that any instance of IntList has two data cells – one to hold an integer, and the other to hold a reference pointing to another instance of IntList.
Quiz: Can you draw the picture for the case in which the instance pointed to happens to be this instance itself?)

IntList has two *constructors*, which are designed to initialize its instances. One of the constructors is called when we create an instance with the `new` keyword.
Quiz: Draw the picture for the following independent uses:
1. new IntList()
2. new IntList(2, new IntList(1, new IntList(new IntList().head, null)))

IntList has two *methods*. `list()` is a *static* method, so it can be called with no specific IntList instance (using the syntax `IntList.list(…)`). It takes *variable arguments*. Those arguments are presented as an *array* (an object of a *composite data type*) inside the method.

`equals()` is not a static (*non-static*) method because it does not make sense to tell whether an IntList is equal to the other if no specific instances are given (`IntList.equal(…)` does not work).
Quiz: What are the results?
1. IntList.list(1, 2, 3).tail.head
2. new IntList() == new IntList()
3. new IntList().equals(new IntList())
4. IntList.list(1).equals(new IntList(1, new IntList()))

Several important keywords: `null` – a reference to nothing. It is a special value that has no type, and cannot be *dereferenced*; `this` – a reference to the current instance that can be used in a non-static method; `instanceof` – an *operator* to test whether an instance *belongs to* a class. (Note: `this` is also used in a constructor to invoke another constructor.)
Quiz: What happens?
1. null.head
2. new IntList().tail.head
3. using this in list()
4. new IntList() instanceof Object
5. null instanceof IntList

Questions about the assignment?

```
class Progs {

    /* 1a. */
    /** The sum of all integers, k, such that 1 <= k < N and
     *  N is evenly divisible by k. */
    static int factorSum (int N) {
        /* *Replace the following with the answer* */
        return 0;
    }

    /* 1b. */
    /** Print the set of all sociable pairs whose members are all
     *  between 1 and N>=0 (inclusive) on the standard output (one pair
     *  per line, smallest member of each pair first, with no
     *  repetitions). */
    static void printSociablePairs (int N) {
        /* *Fill in here* */
    }

    /* 2a. */
    /** A list consisting of the elements of A followed by the
     *  the elements of B.  May modify items of A.
     *  Don't use 'new'. */
    static IntList dcatenate(IntList A, IntList B) {
        /* *Replace the following with the answer* */
        return null;
    }

    /* 2b. */
    /** The sublist consisting of LEN items from list L,
     *  beginning with item #START (where the first item is #0).
     *  Does not modify the original list elements.
     *  It is an error if the desired items don't exist. */
    static IntList sublist (IntList L, int start, int len) {
        /* *Replace the following with the answer* */
        return null;
    }

    /* 2c. */
    /** The sublist consisting of LEN items from list L,
     *  beginning with item #START (where the first item is #0).
     *  May modify the original list elements. Don't use 'new'.
     *  It is an error if the desired items don't exist. */
    static IntList dsublist (IntList L, int start, int len) {
        /* *Replace the following with the answer* */
        return null;
    }

}
```