

This lab has **five (5)** checkpoints. You have until lab time next week to complete checkpoints.

Learning Outcomes

Upon completing this lab, students should be able to

- **Build** a list of class *hierarchy* objects from a file.
- Add a match method (almost equals) to match a fruit to a *type/variety* pair.
- Recursively **delete** a node from a linked list.

Introduction

1. **Do not** type the code in and compile it. You are expected to trace the execution flow.

```
class Fruit { ... }
class Apple extends Fruit { ... }
class Pear extends Fruit { ... }
class Orange extends Fruit { ... }

public class Main {
    public static void main(String[] args) {
        ListOfListsOfFruit all = new ListOfListofFruit();

        ListOfFruit w = new ListOfFruit();
        w.add(new Apple());
        w.add(new Orange());
        w.add(new Apple());
        all.add(w);

        ListOfFruit x = new ListOfFruit();
        x.add(new Orange());
        x.add(new Apple());
        x.add(new Orange());
        all.add(x);

        ListOfFruit y = new ListOfFruit();
        y.add(new Pear());
        all.add(y);

        ListOfFruit z = new ListOfFruit();
        all.add(z);

        System.out.println(all.length());
        System.out.println(all.get(all.length() - 1).length());
        // draw structure of all
    }
}
```

Give the output *and* draw the structure that results. Don't forget that a *LinkedList* class is an *object* with (at least) a head field.

- ✓ Show your output/drawing to the lab instructor.

2. Assume `ListOfFruit.delete(int n)` deletes the element at location `n` in the list. What, if *anything* changes in all if, after your drawing line, the following code is executed:

```
x.delete(1);
all.get(0).delete(1);
```

- ✓ Explain your answer to the lab instructor.

Coding

Move the code in `src` into a new directory. Note that it is a complete Gradle project along with a `data` folder and everything. You will need to initialize `git` in the new directory and do the standard `git add` and `git commit` to be able to push to Gitea.

The code has the following structure (down in `app/src/main/java`, the root of Gradle's class path):

```
.
├──
├── fruit
│   ├── Apple.java
│   ├── Fruit.java
│   ├── Orange.java
│   └── Pear.java
├── main
│   └── UseLinkedList.java
└── rlinked
    └── ListOfFruit.java
```

3. `UseLinkedList` is the main program. You will need to write the `loadFromFile` method. The method opens the file named on the command-line and reads it *by line*.

Each line has two words on it: fruit variety. You must check the fruit value. If it is `Apple`, `Pear`, or `Orange`, create an object of that type and insert it into fruits. If it is anything else, print an error message and ignore the line.

To run the program with the short, mixed fruits data file, the following will run it in Gradle:

```
gradle run -q --console=plain --args=data/Mixed.fruits
```

It is assumed you are in the directory with `app` and `data`; adjust as necessary.

Note You will need three (3) different constructors, one for each known fruit. But you can put the result into a single variable of type `Fruit`. Set the variable to `null` and then run an `if` to construct new fruits. At the end of the `if`, the fruit variable is either the new fruit or still `null`.

Don't insert `null` into the fruit list.

- ✓ Show your working code to the lab instructor. Be prepared to use other data files and the `print` command.

4. The elements in `fruit` are the data objects in the list. You will need to write the `match` function for all `Fruit` objects:

```
boolean match(String fruit, String variety) ...
```

`match` returns `true` if the object is of the type named `fruit` and has a `variety` value as passed in the string `variety`. Do your comparisons without worrying about case. (Documentation is your friend.)

Your code will not yet run.

- ✓ Show your *compiling* code to the lab instuctor.

5. `ListOfFruit` contains two public *stub* functions: `boolean contains(String fruit, String variety)` and `boolean delete(String fruit, String variety)`. Both just return `false`.

Add private **recursive** functions to do the right thing in the list. Note that the private `delete` will have to return a node, not a boolean. Then the public version calls `contains` to see if there is work to do, returning `false` if `contains` returns `false` and deleting the value otherwise.

Delete the *first* (closest to the head) copy of the fruit if there is more than one. Use `match` (which is case-insensitive) to check for matches.

✓ Show your *working* code to the lab instructor.

6. [Not a separate checkpoint but necessary] Add and commit your full `gradle` project (if you `git add` just the root directory, `..`, `git` will add all subdirectories recursively).

Create a repository in your Gitea turn-in organization, associate it with your lab work, and push the repo. This will be done at the end of every lab.