Containers

a) (1 Points). Suppose that we have the `Container` abstract class below, with the abstract method `pour` and the method `drain`. Implement the method `drain` so that all the liquid is drained from the container, i.e. `amountFilled` is set to 0. Return `true` if any liquid was drained, and `false` otherwise. In other words, return true if and only if there is liquid in the container prior to the function being called. You may add a maximum of **5 lines of code**. Note that the staff solution uses 3. You may *only* add code to the `drain` method. (Summer 2021 MT1)

```
1  public abstract class Container {
2      /* Keeps track of the total amount of liquid in the container */
3      public int amountFilled;
4
5      public boolean drain() {
6
7
8
9
10
11     }  // You may use at most 5 lines of code, i.e. this bracket should be on line 11 or earlier.
12
13     abstract int pour(int amount);
14 }
```

Solution: Here is a video walkthrough of the solution.

```
1  public abstract class Container {
2      /* Keeps track of the total amount of liquid in the container */
3      public int amountFilled;
4
5      public boolean drain() {
6          boolean answer = amountFilled > 0;
7          amountFilled = 0;
8          return answer;
9      }
10
11     abstract int pour(int amount);
12 }
```

b) (1.5 Points). Finish implementing the `WaterBottle` class so that it **is a** `Container`. You should *only* add code to the blanks, i.e. **fill in the `pour` method and the class signature**.

As stated in the `Container` class, the `pour` method should pour `amount` into the container and return the amount of the excess liquid, or 0 if there is no excess. For instance, suppose we have a `WaterBottle w` with `capacity` **`10`** and `amountFilled` **`5`**. Then, if we execute `w.pour(7)`, `amountFilled` should be set to **`10`** and **`2`** should be returned. Your solution *must* fit within the blanks provided. You may not need all

the lines.

```
class WaterBottle _______________ Container {
    private static final int DEFAULT_CAPACITY = 16;

    /* The capacity of the container, i.e. the maximum amount of liquid the water bottle can hold */
    private int capacity;

    WaterBottle() {
        this(DEFAULT_CAPACITY);
    }
    WaterBottle(int capacity) {
        this.capacity = capacity;
        this.amountFilled = 0;
    }

    @Override
    public int pour(int amount) {
        ______________________________________;
        if (_________________________________) {
            ___________________________________;
            ___________________________________;
            ___________________________________;
        }
        ______________________________________;
    }
}
```

Solution: Here is a video walkthrough of the solution.

```
class WaterBottle extends Container {
    private static final int DEFAULT_CAPACITY = 16;

    /* The capacity of the container, i.e. the maximum amount of liquid the water bottle can hold */
    private int capacity;

    WaterBottle() {
        this(DEFAULT_CAPACITY);
    }
    WaterBottle(int capacity) {
        this.capacity = capacity;
        this.amountFilled = 0;
    }

    @Override
    public int pour(int amount) {
        filled += amount;
        if (filled > capacity) {
```

```
            int excesss = filled - capacity;
            filled = capacity;
            return excesss;
        }
        return 0;
    }
}
```

c) (4 Points). Finally, suppose we have the `ContainerList` class, with the `drainFirst` method as implemented below. Unfortunately, the `drainFirst` method *sometimes* errors!

In order to fix it, you may add code to the **`ContainerList` constructor and the `UnknownContainer`** class! You may only **use** 5 lines of code in the `ContainerList` constructor and **add** 4 lines of code to the `UnknownContainer` class! If you decide to keep or modify the given line in the `ContainerList` constructor, it counts as one of the 5 lines.

Note that, after making your changes, the `drainFirst` should **never error and retain the functionality in the docstring**. You **may not modify the `drainFirst` method!** You may use classes from the previous part assuming they are implemented correctly.

Hint: Make sure that, with your fix, the `drainFirst` method won't error, even if the `drainFirst` method is called many times.

```
class UnknownContainer ____________________ {
    // TODO
}
} // You may add at most 4 lines of code to the class above
// i.e. the closing bracket should be on line 6 or earlier

class ContainerList {
    private Container[] containers;

    ContainerList(Container[] conts) {
        this.containers = conts; // you may delete, modify, or keep this line
        // YOUR CODE HERE
    } // You may use at most 5 lines of code in the Constructor
```

```
    // i.e. the closing bracket should be on line 18 or earlier

    /* Drains the water from the first nonempty container */
    void drainFirst() {
        int index = 0;
        while (!containers[index].drain()) {
            index += 1;
        }
    }
}
```

Solution: Here is a video walkthrough of the solution.

```
class UnknownContainer extends WaterBottle {
    @Override
    public boolean drain() {
        return true;
    }
}

class ContainerList {
    private Container[] containers;

    ContainerList(Container[] conts) {
        containers = new Container[conts.length + 1];
        for (int i = 0; i < conts.length; i += 1) {
            containers[i] = conts[i];
        }
//        System.arraycopy(conts, 0, containers, 0, conts.length); <- can replace for loop with this
        containers[conts.length] = new UnknownContainer();
    }

    /* Drains the first nonempty container */
    void drainFirst() {
        int index = 0;
        while (!containers[index].drain()) {
            index += 1;
        }
    }
}
```

Explanation: `drainFirst` cannot handle a case with all empty containers–it keeps incrementing `index` until it's out of bounds. The solution is to add a `Container` which can always be drained, the `UnknownContainer`. Thus, we write an `UnknownContainer.drain` which always returns `true`.
However, we can't just override `Container`, since this will require you to implement both `drain` and `pour` (which requires more than 4 lines). Instead, we have to extend `WaterBottle`.

Then, in `ContainerList`, we add an extra `UnknownContainer` to the end of the `containers` list. However, an array's size cannot be changed, so we have to copy `conts`, then add the last `UnknownContainer`.