OO Class hierarchy/association

Tuesday, December 13, 2022 11:29 AM

for shared code (avoid re-typing the same AddFirst implementation in both SizedEmpty and SizedLink)

```
public abstract class SizedImmList implements IList {
    public IList addFirst(String newVal) {
        return new SizedLink(newVal, this);
    }
}

public class SizedEmpty extends SizedImmList {
    @Override
    public int size() {
        return 0;
    }
}

public class SizedLink extends SizedImmList {
    private int length;
    private String first;
    private IList rest;

    public SizedLink(String first, IList rest) {
        this.first = first;
        this.rest = rest;
        this.length = 1 + rest.size();
    }

    @Override
    public int size() {
        return this.length;
    }
}
```

Interfaces guarantee behavior ("any IList must have addFirst and size methods")

because SizedEmpty extends SizedImmList and SizedImmList implements IList, this implictly means that SizedEmpty implements IList

IList is being used to group code (essentially saying "this can be SizedLink or Empty")

How to tell when to use interface/parent class/abstract class
Abstract class: only for shared code
(cannot instantiate abstract classes)
Interface: A and B are not "subclassfications" of each ("A" is not "B" and "B" is not "A"), but they share behavior/need to be grouped together
Parent classes: If "A" is "B" but with additional behavior, then we would want A to extend B (B would be parent class)

Example: "B" is a university course
"A" is a university course with a lab component
We want A to be able to leverage all of the behavior of B, plus add new behavior

public class CourseWithLab extends Course

Need to be careful about what variables are defined as when dealing with interfaces and inheritance:

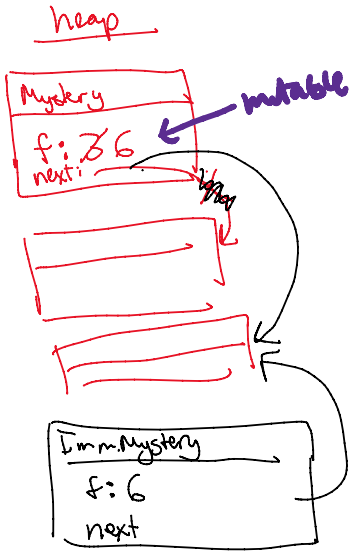
```
public class Boa implements Ianimal
public class Tiger implements IAnimal

// say Boa has a method canStrangle that Tiger doesn't

// say Zoo class has a method that returns IAnimal (acquireAnimal)
public IAnimal acquireAnimal()

IAnimal newAnimal = ourZoo.acquireAnimal(); // could not call
canStrangle on newAnimal, even if returned animal were a Boa
```

Telling that something is immutable

Tuesday, December 13, 2022 11:29 AM

How do we tell that something is immutable?

- In a memory diagram: new objects are only created, never altered
- In code: only the constructor Sets field values
- NO other method alters field values




```
public abstract class SizedImmList implements IList {
    public IList addFirst(String newVal) {
        return new SizedLink(newVal, this);
    }
}

public class SizedEmpty extends SizedImmList {
    @Override
    public int size() {
        return 0;
    }
}

public class SizedLink extends SizedImmList {
    private int length;
    private String first;
    private IList rest;

    public SizedLink(String first, IList rest) {
        this.first = first;
        this.rest = rest;
        this.length = 1 + rest.size();
    }

    @Override
    public int size() {
        return this.length;
    }
}
```

Immutable because the only time the code
[field name] = [something]
Appears is in the constructor

Runtime

Tuesday, December 13, 2022 11:29 AM

```
public abstract class SizedImmList implements IList {
    public IList addFirst(String newVal) {
        return new SizedLink(newVal, this);
    }
}

public class SizedEmpty extends SizedImmList {
    @Override
    public int size() {
        return 0;
    }
}

public class SizedLink extends SizedImmList {
    private int length;
    private String first;
    private IList rest;

    public SizedLink(String first, IList rest) {
        this.first = first;
        this.rest = rest;
        this.length = 1 + rest.size();
    }

    @Override
    public int size() {
        return this.length;
    }
}
```

↳ same as constructor runtime

constant

constant

↳ constant (just returns field)

Runtime, 1000-foot view:

If we increase the "problem size" (input size) by 1, by how much does the solution increase in runtime?

O(N) means that we have to define with "N" is

- The length of a list
- The number of nodes in a tree
- The width and the height of a 2d array (O(W*H))
- The # of vertices and edges for a graph (O(|V| + |E|))

Contrast code above with Link implementation we saw in lecture:

```
public class Link extends IList() {
    private String first;
    private IList rest;

    public Link(String first) {
        this.first = first;
        this.rest = rest;
    }

    public int size() {
        return 1 + this.rest.size();
    }
}
```

Runtime increases for each element in the list by a constant amount (1 + [cost of method call]) because of the recursion

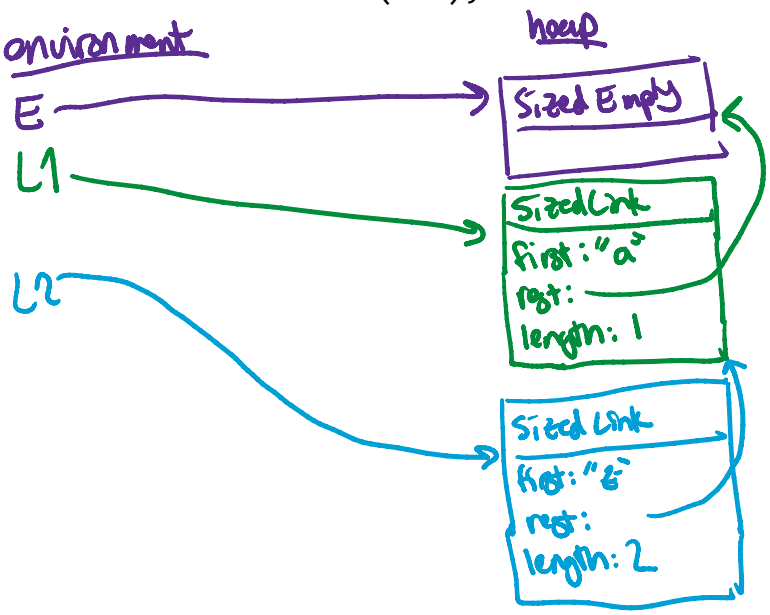
O(N)

Basic memory diagram

Tuesday, December 13, 2022 1:35 PM

```
public abstract class SizedImmList implements IList {
   public IList addFirst(String newVal) {
       return new SizedLink(newVal, this);
   }
}

public class SizedEmpty extends SizedImmList {
   @Override
   public int size() {
       return 0;
   }
}

public class SizedLink extends SizedImmList {
   private int length;
   private String first;
   private IList rest;

   public SizedLink(String first, IList rest) {
       this.first = first;
       this.rest = rest;
       this.length = 1 + rest.size();
   }

   @Override
   public int size() {
       return this.length;
   }
}
```

```
IList E = new SizedEmpty();
IList L1 = new SizedLink("a", E);
IList L2 = L1.addFirst("z");
```



Note that this helps show why the constructor runs in constant time: when L1.addFirst("z") is called, that method runs new SizedLink("z", this) (where this is the green SizedLink). Then, in the constructor, first is "z" and rest is the green SizedLink. Calling rest.size just returns the length field of the green SizedLink, which can be fetched in constant time.

remove

Tuesday, December 13, 2022 12:54 PM

```
// in SizedEmpty
1 public IList remove(String toRemove) {
2    System.out.println("remove reached the end of the list");
3    return this;
4 }
```

throws ItemNotFoundExcn

Need to change the method annotation because we are now throwing an exception inside the method

throw new ItemNotFoundExcn(...)

```
// in SizedLink
1 public IList remove(String toRemove) {
2    if (this.first.equals(toRemove)) {
3        System.out.println("remove found the item!");
4        return this.rest;
5    } else {
6        System.out.println("item " + this.first + " does not match");
7        return new SizedLink(this.first, this.rest.remove(toRemove));
8    }
9 }
```

throws ItemNotFoundExcn

remove has to be an IList method (in order for the recursion to work on both SizedLink and Empty). Since we modified the method annotation for remove in SizedEmpty, remove in both IList and SizedLink also need to have this annotation.

This will also pass the exception to the caller of remove, which is what we want (we don't want to try/catch the exception thrown in SizedEmpty here, because we are still in the model part of the MVC)

Try calling .remove("z") on the list with ["a", "b", "c"]

1) Remove is called, first is "a"
   In SizedLink, statement on line 2 evaluates to false so we enter the else
   Prints "item a does not match"
   Needs to return from this.rest.remove("z") before calling the SizedLink constructor
2) In the recursive call, first is "b"
   Prints "item b does not match" (line 2 is false for same reason as above)
   Needs to return from this.rest.remove("z") before calling the SizedLink constructor
3) In the recursive call, first is "c"
   Prints "item c does not match" (line 2 is false for same reason as above)
   Needs to return from this.rest.remove("z") before calling the SizedLink constructor. Note that this.rest is a SizedEmpty
4) In the recursive call .remove("z") on SizedEmpty
   Prints "remove reached the end of the list"
   throws the exception
5) Computer discards methods waiting to finish on the call stack (from steps 3, 2, 1, and whatever called remove) until it reaches a try/catch block somewhere