

Midterm results

- 90 points total
- Exams will be returned in mailboxes
- Solution will be posted by Thursday

Midterm results

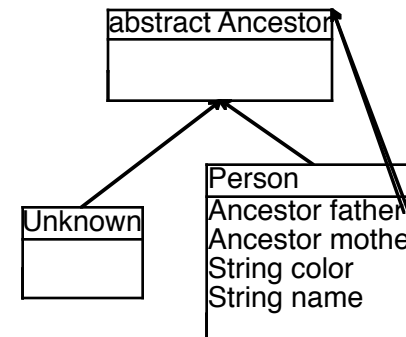
- Range :: Number at range
 - Approximate letter grade
- 90 - 80 :: 4
 - A+ to A
- 79 - 70 :: 4
 - A- to B+
- 69 - 60 :: 4
 - B to B-
- 59 - 50 :: 2
 - C+ to C
- 49 - 40 :: 2
 - C- to D

Last time

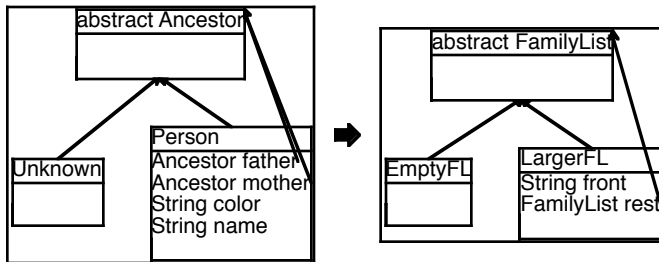
- Insertion sort
- Linear search
- Binary search
 - Requires properly balanced tree

Trees to Lists

Our family tree geneticist wants to compile a list of family members with the same eye color



Trees to Lists



withEye(String color)

```

abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    FamilyList withEye( String c ) {
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ... this.father.AncestorMethod( ... )
        // ... this.mother.AncestorMethod( ) ... c
    }
}

```

withEye(String color)

```

abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> new EmptyFL()
    FamilyList withEye( String c ) {
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> new LargerFL("Fred", new EmptyFL())
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    //     new Person("Marie","brown",new Unknown(),new Unknown())).withEye("brown")
    // -> new LargerFL("Betty", new LargerFL("Marie", new EmptyFL()))
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ) ... c
    }
}

```

withEye(String color)

```

abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> _____
    FamilyList withEye( String c ) {
        return new EmptyFL();
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> _____
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    //     new Person("Marie","brown",new Unknown(),new Unknown())).withEye("brown")
    // -> _____
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ) ... c
    }
}

```

withEye(String color)

```
abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> _____
    FamilyList withEye( String c ) {
        return new EmptyFL();
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> _____
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    // new Person("Marie","brown",new Unknown(),new Unknown()).withEye("brown")
    // -> _____
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ) ... c
        if ( _____ )
            return _____;
        else
            return _____;
    }
}
```

withEye(String color)

```
abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> _____
    FamilyList withEye( String c ) {
        return new EmptyFL();
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> _____
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    // new Person("Marie","brown",new Unknown(),new Unknown()).withEye("brown")
    // -> _____
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ) ... c
        if ( this.color.equals(c) )
            return _____;
        else
            return _____;
    }
}
```

withEye(String color)

```
abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> _____
    FamilyList withEye( String c ) {
        return new EmptyFL();
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> _____
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    // new Person("Marie","brown",new Unknown(),new Unknown()).withEye("brown")
    // -> _____
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ) ... c
        if ( this.color.equals(c) )
            return new LargerFL(this.name, this.father.withEye(c) ... this.mother.withEye(c) ...);
        else
            return _____;
    }
}
```

Putting two lists together

- new LargerFL("John",new LargerFL("Marie",new EmptyFL())).append(new LargerFL("Peter",new EmptyFL()))
 - new LargerFL("John",new LargerFL("Marie",new LargerFL("Peter",new EmptyFL())))
- Let's write append

Back to withEye(String color)

```

abstract class Ancestor {
    //To produce a list of family members of this Ancestor with c eyes
    abstract FamilyList withEye( String c );
}

class Unknown extends Ancestor {
    //To produce a list of family members of this Unknown with c eyes
    // new Unknown().withEye("blue") -> _____
    FamilyList withEye( String c ) {
        return new EmptyFL();
    }
}

class Person extends Ancestor {
    //To produce a list of family members of this Person with c eyes
    //new Person("Fred","blue",new Unknown(),new Unknown()).withEye("blue")
    // -> _____
    //new Person("Betty","brown",new Person("Sam","hazel",new Unknown(),new Unknown())
    // new Person("Marie","brown",new Unknown(),new Unknown()).withEye("brown")
    // -> _____
    FamilyList withEye( String c ) {
        // ... this.color ... this.name ...
        // ... this.father.AncestorMethod( ... ) this.mother.AncestorMethod( ... ) c
        if ( this.color.equals(c) )
            return new LargerFL(this.name, this.father.withEye(c).append(this.mother.withEye(c)));
        else
            return this.father.withEye(c).append(this.mother.withEye(c));
    }
}

```

What all do we know how to do with a list?

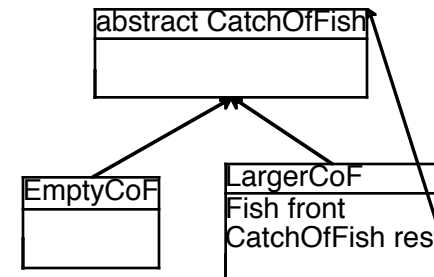
- Find the length
- Find the sum of the contents
- Make a new list, with different properties
- Determine if all elements of a list satisfy a condition
- Determine if any element of the list satisfies a condition
- Sort it

The lists

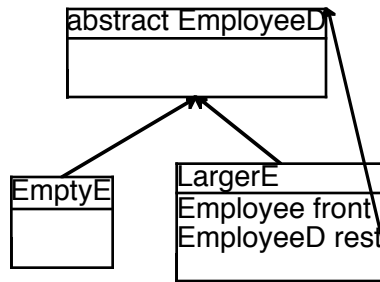
- When we needed to keep track of a lot of Fish ...
 - We used a Catch of fish -- a list
- When we needed to keep track of a lot of Employees ...
 - We used an EmployeeD -- a list
- When we needed to keep track of the members of a subset of a family ...
 - We used a FamilyList -- a list

What's different about these lists?

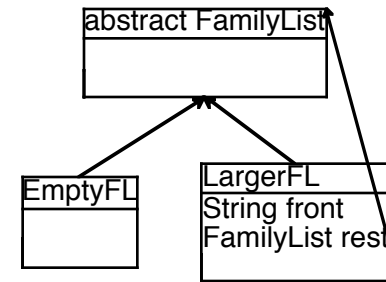
List data



List data



List data

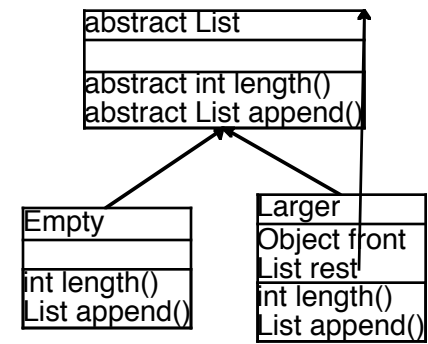


Graduation Time!



Welcome to ProfessorJ: Intermediate

A very general list



A very general list

```
abstract class List {  
}  
class Empty extends List {  
  Empty() {} // Can now be skipped if you would like  
}  
class Larger extends List {  
  Object first;  
  List rest;  
  Larger( Object first, List rest) { // Cannot be skipped  
    this.first = first;  
    this.rest = rest;  
  }  
}
```

What is Object?

- Fish is a subclass of Object
- Fisherman is a subclass of Object
- Pizza is a subclass of Object

Every class is a subclass of Object

... even Strings are Objects

Making a general list

- A list of Fish, with weights 3,4,and 5
- A list of Strings, "hi", "low"

Generality cost us...

- How can we have a list of int's?
 - class IntWrapper { int val; ... }
- How can we sort a list?

Object

- Object allows general data structures
- Hides specific information about the data contained in those structures

Like List hides specific information about Larger

Totalable

Let's get back the ability to total up a list, without having umpteen lists

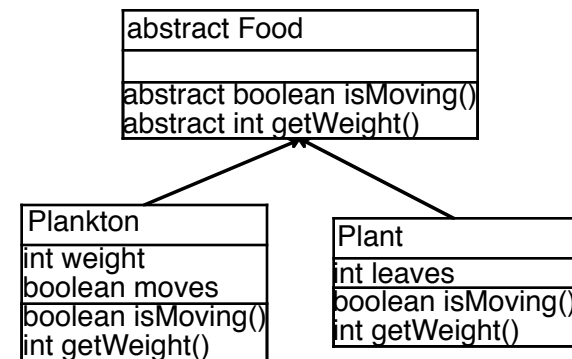
What must we know to sum a list?

Object -> int

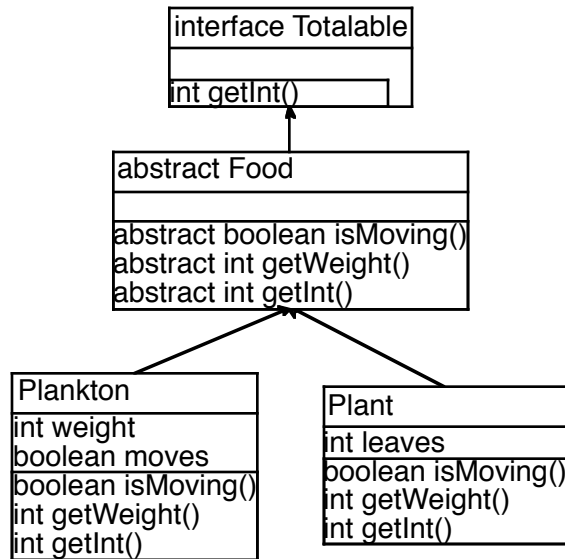
We need a method to return the correct integer from an Object, which Object does not have

We do not want to modify existing inheritances

An old friend



interface



interface

```
interface Totalable {
    int getInt();
}
abstract class Food implements Totalable {
    ...
}
```

A totalable list

```
abstract class ListT {
    abstract int total();
}

class EmptyT extends ListT {
    int total() {
        return 0;
    }
}

class LargerT extends ListT {
    Totalable first;
    ListT rest;
    ...
    int total() {
        return this.first.getInt() + this.rest.total();
    }
}
```

How is this better?

Fish can be a Totalable
Employee can be a Totalable
Toy can be a Totalable