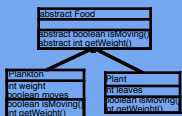


## Data

- Simple data
- Compound data
- Compound data using other compound data
- Conditional Compound Data



- Recursive Data



## Computation

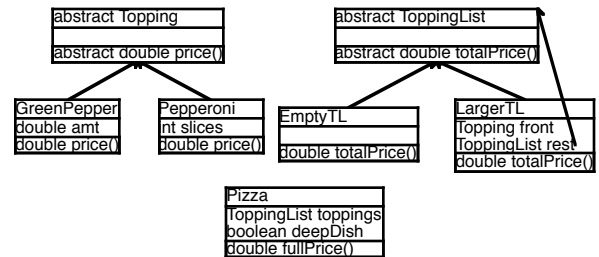
- if
  - if ( CONDITION ) STATEMENT else STATEMENT
  - CONDITION is an expression producing a boolean
  - STATEMENT is either if or 'return ...;'
- Methods on recursive data
  - Counting the items in a Catch
  - Determine if all items in a Catch satisfy a condition

## Explanation revisited

- new EmptyCatch().allOver10lbs() == true
  - Makes the program work out, but is also mathematically correct
- Assume new EmptyCatch().allOver10lbs() == false
  - Indicates that some item in EmptyCatch() is under 10 lbs
  - We cannot produce one
- In absence of a counter example, new EmptyCatch().allOver10lbs() is true

## Making a Pizza Pie

A pizza normally has many Toppings: here's an upgrade of our Lab pizza



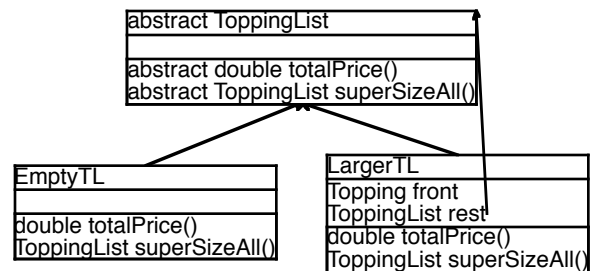
## A look at computing the price of a Pizza

To DrScheme

## More, more, more...

Our restaurant has decided to add a superSize option to the pizza. A superSize(ed) topping is twice the old topping.

Now lets add a method to superSize all of the toppings



## Purpose, header, examples

```

abstract class ToppingList {
  //To produce a ToppingList similar to this ToppingList, with superSized toppings
  abstract ToppingList superSizeAll();
}

class EmptyTL extends ToppingList {
  //new EmptyTL().superSizeAll() -> new EmptyTL()
  //To produce a ToppingList similar to this EmptyTL, with superSized toppings
  ToppingList superSizeAll() {
  }
}

class LargerTL extends ToppingList {
  //new LargerTL( new Pepperoni(5),
  // new LargerTL( new GreenPepper(.5), new EmptyTL()).superSizeAll() --> ???
  //new LargerTL( new Pepperoni(2), new EmptyTL() ).superSizeAll() --> ???
  //To produce a ToppingList similar to this LargerTL, with superSized toppings
  ToppingList superSizeAll() {
  }
}
  
```

## Templates & body

```

abstract class ToppingList {
  //To produce a ToppingList similar to this ToppingList, with superSized toppings
  abstract ToppingList superSizeAll();
}

class EmptyTL extends ToppingList {
  //new EmptyTL().superSizeAll() -> new EmptyTL()
  //To produce a ToppingList similar to this EmptyTL, with superSized toppings
  ToppingList superSizeAll() {
    return new EmptyTL();
  }
}

class LargerTL extends ToppingList {
  //new LargerTL( new Pepperoni(5),
  // new LargerTL( new GreenPepper(.5), new EmptyTL()).superSizeAll() --> ???
  //new LargerTL( new Pepperoni(2), new EmptyTL() ).superSizeAll() --> ???
  //To produce a ToppingList similar to this LargerTL, with superSized toppings
  ToppingList superSizeAll() {
    // ... this.front.ToppingMethod ... this.rest.ToppingListMethod ...
    return .... this.front.superSize() ... this.rest.superSizeAll() ... ;
  }
}
  
```

## ToppingList superSizeAll()

```

abstract class ToppingList {
    //To produce a ToppingList similar to this ToppingList, with superSized toppings
    abstract ToppingList superSizeAll();
}

class EmptyTL extends ToppingList {
    //new EmptyTL().superSizeAll() -> new EmptyTL()
    //To produce a ToppingList similar to this EmptyTL, with superSized toppings
    ToppingList superSizeAll() {
        return new EmptyTL();
    }
}

class LargerTL extends ToppingList {
    //new LargerTL( new Pepperoni(5),
    //    new LargerTL( new GreenPepper(5), new EmptyTL()))
    //new LargerTL( new Pepperoni(2), new EmptyTL() ).superSizeAll() --> ???
    //To produce a ToppingList similar to this LargerTL, with superSized toppings
    ToppingList superSizeAll() {
        // ... this.front.ToppingMethod ... this.rest.ToppingListMethod ...
        return new LargerTL( this.front.superSize(), this.rest.superSizeAll());
    }
}

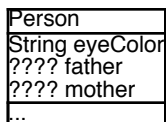
```

## No meat!

- For our vegetarian customers, we need to remove meat from some topping lists
  - We therefore need to be able to distinguish between vege and meat toppings

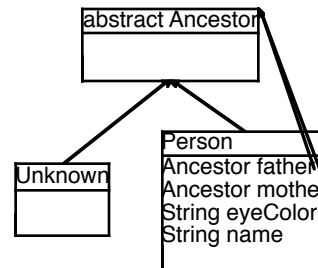
## Family trees

A geneticist wants to examine eye-color in a family line



- What kind of data should father and mother be?
- How should we get to grandparents, great-grandparents, great-great... ?
- How do we typically draw a family tree?

## Family tree



## In Java

```
abstract class Ancestor {
}
class Unknown extends Ancestor {
    Unknown(){}
}
class Person extends Ancestor {
    Ancestor father;
    Ancestor mother;
    String eyeColor;
    String name;
    Person( String name, String eyeColor, Ancestor father, Ancestor mother ) {
        this.name = name;
        this.eyeColor = eyeColor;
        this.father = father;
        this.mother = mother;
    }
}
```

## Representing a family

- Representing my family --
  - me (blue): parents Jim (grey) & Ruth (blue):
  - grands Jim (blue) & Opal (green), Charles (blue) & Eleanor (blue)
  - ```
new Person("Kathy", "blue",
            new Person("Jim", "grey",
                      new Person("Jim", "blue", new Unknown(), new Unknown()),
                      new Person("Opal", "green", new Unknown(), new Unknown()))),
            new Person("Ruth", "blue",
                      new Person("Charles", "blue", new Unknown(), new Unknown()),
                      new Person("Eleanor", "blue", new Unknown(), new Unknown()))))
```
- And now for my brother
  - ... that's going to be a lot of retyping --- there has to be a better way

## Naming data for reuse

- You can name data for use in tests, exploration, and examples
  - i.e. inside of Test classes, example boxes (if you're using testcase boxes), and interactions window
- `double pi = 3.1415926535;`
- `Unknown none = new Unknown();`
- `Person grandad = new Person("Charles", "blue", none, none);`

## Let's make Family trees

- A family with at least three "brown" eyed people
- A family with only one "blue" eyed person
- A family for someone who doesn't know their parents

## Using the data

How many people in a family?

```
abstract class Ancestor {
    //To count the number of known people in this Ancestor's family
    abstract int size();
}
class Unknown extends Ancestor {
    Unknown() {}
    //To count the number of known people in this Unknown's family
    int size() {
    }
}
class Person extends Ancestor {
    String eyeColor;
    String name;
    Ancestor father;
    Ancestor mother;
    Person( String eyeColor, String name, Ancestor father, Ancestor mother ) {
        this.eyeColor = eyeColor;
    }
    ...
    //To count the number of known people in this Person's family
    int size() {
    }
}
```

## Examples

## Body template

```
abstract class Ancestor {
    //To count the number of known people in this Ancestor's family
    abstract int size();
}
class Unknown extends Ancestor {
    Unknown() {}
    //To count the number of known people in this Unknown's family
    int size() {
        // ...
    }
}
class Person extends Ancestor {
    String eyeColor;
    String name;
    Ancestor father;
    Ancestor mother;
    Person( String eyeColor, String name, Ancestor father, Ancestor mother ) {
        this.eyeColor = eyeColor;
    }
    ...
    //To count the number of known people in this Person's family
    int size() {
        //... this.eyeColor ... this.name ...
        //... this.father.AncestorMethod() ... this.mother.AncestorMethod()
    }
}
```

## Body

```
abstract class Ancestor {
    //To count the number of known people in this Ancestor's family
    abstract int size();
}
class Unknown extends Ancestor {
    Unknown() {}
    //To count the number of known people in this Unknown's family
    int size() {
        return 0;
    }
}
class Person extends Ancestor {
    String eyeColor;
    String name;
    Ancestor father;
    Ancestor mother;
    Person( String eyeColor, String name, Ancestor father, Ancestor mother ) {
        this.eyeColor = eyeColor;
    }
    ...
    //To count the number of known people in this Person's family
    int size() {
        return 1 + this.father.count() + this.mother.count();
    }
}
```

## Stepping through

## Methods on trees

- Count the number of people in a family with a given eye color
  - Practice

## new Tree representations

Our geneticist, named Mary, inadvertently represented everyone named Mary with her parents, instead of their own

Implement a means to create new representations with the Marys having the correct ancestry

## correction( Ancestor pop, Ancestor mom)

```
abstract class Ancestor {
    //To produce a correct family tree, based on this Ancestor's family
    abstract Ancestor correction( Ancestor pop, Ancestor mom);
}

class Unknown extends Ancestor {
    //To produce a correct family tree, based on this Unknown's family
    Ancestor correction( Ancestor pop, Ancestor mom) {
    }
}

class Person extends Ancestor {
    //To produce a correct family tree, based on this Person's family
    Ancestor correction( Ancestor pop, Ancestor mom) {
    }
}
```

- Examples ...
- Template ...

## full implementation

```

abstract class Ancestor {
    //To produce a correct family tree, based on this Ancestor's family
    abstract Ancestor correction( Ancestor pop, Ancestor mom);
}

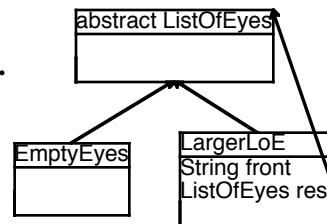
class Unknown extends Ancestor {
    //To produce a correct family tree, based on this Unknown's family
    Ancestor correction( Ancestor pop, Ancestor mom) {
        return this;
    }
}

class Person extends Ancestor {
    //To produce a correct family tree, based on this Person's family
    Ancestor correction( Ancestor pop, Ancestor mom) {
        // ... this.name ... this.eyeColor ... this.father.Method() ...this.mother.Method()
        if (this.name.equals("Mary"))
            return new Person(this.name,this.eyeColor,pop,mom);
        else
            return new Person(this.name,this.eyeColor,
                this.father.correction(pop,mom),
                this.mother.correction(pop,mom) );
    }
}

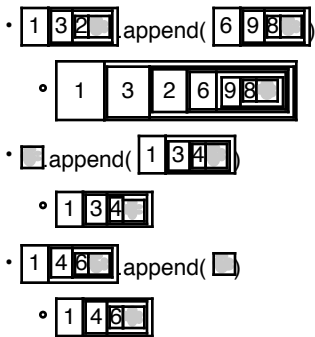
```

## Appending two Lists

The geneticist needs to merge two lists of eye-color together



## Appending two Lists



## append Eyes

```

abstract class ListOfEyes {
    //To append this ListOfEyes onto another
    abstract ListOfEyes append(ListOfEyes another);
}

class EmptyEyes extends ListOfEyes {
    //To append this EmptyEyes onto another
    //new EmptyEyes().append(new LargerLoE("blue",new EmptyEyes())) -> ??
    //new EmptyEyes().append(new EmptyEyes()) -> ???
    ListOfEyes append(ListOfEyes another) {
    }
}

class LargerLoE extends ListOfEyes {
    //To append this ListOfEyes onto another
    //new LargerLoE("blue", new EmptyEyes()).append(new EmptyEyes()) -> ???
    //new LargerLoE("blue",new EmptyEyes()).append(new LargerLoE("red",new EmptyEyes())) -> ?
    ListOfEyes append(ListOfEyes another) {
    }
}

```

## append Eyes

```
abstract class ListOfEyes {
  //To append this ListOfEyes onto another
  abstract ListOfEyes append(ListOfEyes another);
}

class EmptyEyes extends ListOfEyes {
  //To append this EmptyEyes onto another
  //new EmptyEyes().append(new LargerLoE("blue",new EmptyEyes())) -> ??
  //new EmptyEyes().append(new EmptyEyes()) -> ???
  ListOfEyes append(ListOfEyes another) {
    // ... another.ListOfEyesMethod() ...
  }
}

class LargerLoE extends ListOfEyes {
  //To append this ListOfEyes onto another
  //new LargerLoE("blue", new EmptyEyes()).append(new EmptyEyes()) -> ???
  //new LargerLoE("blue",new EmptyEyes()).append(new LargerLoE("red",new EmptyEyes())) -> ???
  ListOfEyes append(ListOfEyes another) {
    //... this.front ... this.rest.ListOfEyesMethod() ... another ...
  }
}
```

## append Eyes

```
abstract class ListOfEyes {
  //To append this ListOfEyes onto another
  abstract ListOfEyes append(ListOfEyes another);
}

class EmptyEyes extends ListOfEyes {
  //To append this EmptyEyes onto another
  //new EmptyEyes().append(new LargerLoE("blue",new EmptyEyes())) -> ??
  //new EmptyEyes().append(new EmptyEyes()) -> ???
  ListOfEyes append(ListOfEyes another) {
    // ... another.ListOfEyesMethod() ...
    return another;
  }
}

class LargerLoE extends ListOfEyes {
  //To append this ListOfEyes onto another
  //new LargerLoE("blue", new EmptyEyes()).append(new EmptyEyes()) -> ???
  //new LargerLoE("blue",new EmptyEyes()).append(new LargerLoE("red",new EmptyEyes())) -> ?
  ListOfEyes append(ListOfEyes another) {
    //... this.front ... this.rest.ListOfEyesMethod() ... another ...
    return new LargerLoE(this.front, this.rest.append( another ) );
  }
}
```