Overlappable

Determining whether or not two objects overlap is pretty straight-forward when there is only a single variable of comparison; however, what about the case where you have multiple variables to consider, for example if we have a group of people "People A" who are males between 15 and 25, weighing between 200 and 300 pounds:

```
People A
Ages: 15-25
Gender: M
Weight: 200-300
```

Now consider the case when you have other search rules in the same search and that for the sake of efficiency we want to find overlapping rules and merge them into one (we will only consider the overlap determination here). With two other rules:

```
People B
Ages: 13-20
Gender: M
Weight: 250-400

People C
Ages: 8-10
Gender: M
Weight: 200-300
```

It’s not hard to look at the rules and determine which ones overlap. "People A" overlaps with "People B", while "People C" does not overlap with either of the other two. In code this can be a difficult comparison to do correctly and efficiently. Also, as your number of comparison axes increases, so does the complexity of determining overlap for any given pair of objects.

What is needed is a simple means of determining the overlap of two objects and the best way I have found to do that is to break each object down into its overlap-comparison components, each of which I will call a "Lobe" from here on out. I chose the term lobe because it is defined as: "any rounded projection forming part of a larger structure". Also, terms like element and node are used far too much already in programming.

When you break each object down into it’s lobes, you will have one lobe for Gender, one for Ages, and one for weights. Now you can build your overlap determination based on whether or not each lobe overlaps with its corresponding lobe on the other object. If all of the lobes overlap those of the other object, then the two objects are considered overlapping, otherwise they are not. This allows for a fail-fast comparison since if the comparison of any given lobe fails, you cannot have an overlapping object and no further comparison is necessary.

In code a lobe can be defined by interface as:

```
interface Lobe {
    boolean overlaps( Lobe other )
}
```

Each `Lobe` implementation defines what it means to overlap another `Lobe` of the same type. Using Groovy and some generic logic we can easily come up with a `ComparableLobe` which is based on single values and ranges of `Comparable` objects such as numbers, strings and ranges. This allows us to do things like:

```
new ComparableLobe( 10..20, 50, 75 )
new ComparableLobe( 'a', 'h'..'j', 'm' )

lobeA.overlaps( lobeB )
```

which can make the overlap determination very flexible.

```
def genderLobe = new ComparableLobe( 'M' )
def agesLobe = new ComparableLobe( 15..25 )
def weightsLobe = new ComparableLobe( 200..300 )
```

The next thing we need is a way of comparing these lobes in a simple and repeatable manner and that's where the `Overlappable` trait comes in. The `Overlappable` trait defines an object that can be compared for overlap. The required method is basically the same as that of the Lobe; however, this trait is for the parent object itself. By providing an abstract implementation of this interface we have a nice clean way of providing overlap detection functionality for an object type. You could create a simple `Overlappable` Person object:

```
class People implements Overlappable {
    String gender
    IntRange ages
    IntRange weights

    @Override Lobe[] getLobes() {
        [
            new ComparableLobe(gender),
            new ComparableLobe(ages),
            new ComparableLobe(weights)
        ]
    }
}
```

The overlaps() method uses the provided Lobes to populate an `OverlapBuilder`, which is basically a helper class for performing the actual Lobe-to-Lobe comparison of a given set of Lobes. The `OverlapBuilder` is inspired by the builder in the Apache Commons - Lang API, such as `EqualsBuilder` and `HashCodeBuilder`. You create an instance and append your Lobes to it, then execute the `overlap()` method to perform the comparison.

```
new OverlapBuilder()
    .appendLobe(new ComparableLobe(1..10), new ComparableLobe(5..15))
    .overlap()
```

It also provides an append method for simple comparable cases:

```
overlapBuilder.appendComparable( 20..25, 15..30 )
```

which just wraps each value in a `ComparableLobe`. Now, given a list of People objects, you can determine if any of them overlap any of the others simply by iterating over the list and comparing each element with the others:

```
def list = [
    new Person( gender:'M', ages:15..25, weights:200..300 ),
    new Person( gender:'M', ages:13..20, weights:250..400 ),
    new Person( gender:'M', ages:8..10, weights:200..300 )
]

list[0..-2].eachWithIndex { self, idx->
    list[(idx+1)..(-1)].each { other->
        if( self.overlaps( other ) ){
            println "$self overlaps $other"
        }
    }
}
```

As a final little bonus feature, there is a `ComparableLobe.ANY` object which denotes a `Lobe` that will always be considered to overlap, no matter what the other value is.