Filtering Data

MatchAll

What If...

```
public class MagDepthLocFilter implements Filter {
   //fields and constructor elided
   public boolean satisfies(QuakeEntry qe) {
    return (qe.getMagnitude() >= magMin &&
       qe.getLocation().distanceTo(where)
             <= distance &&
       qe.getDepth() >= minDepth &&
        qe.getDepth() <= maxDepth);
     }
}
```

- Suppose you wanted to match combination:
  - Depth **and** location **and** magnitude

What If...

```
public class MagDepthLocFilter implements Filter {
   //fields and constructor elided
   public boolean satisfies(QuakeEntry qe) {
    return (qe.getMagnitude() >= magMin &&
       qe.getLocation().distanceTo(where)
          <= distance &&
       qe.getDepth() >= minDepth &&
       qe.getDepth() <= maxDepth);
    }
}
```

Duplicates Code!

- Suppose you wanted to match combination:
  - Depth **and** location **and** magnitude

Reuse Existing Filters?

```
MatchAllFilter maf = new MatchAllFilter();
maf.addFilter(new MinMagFilter(4.0));
maf.addFilter(new DepthFilter(-2000,-1000));
maf.addFilter(new DistanceFilter(myLoc, 100));
ArrayList<QuakeEntry> quakes
          = filter(list, maf);
```

- Already have filters that do these tasks
  - Could you write one to combine them?
  - Could the combination be generic?
    - Combine other filters, match all of them

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    private ArrayList<Filter> filters;
    public MatchAllFilter() {
       filters = new ArrayList<Filter>();
    }
    public void addFilter(Filter f) {
       filters.add(f);
    }
    //we'll see .satisfies in a second
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    private ArrayList<Filter> filters;
    public MatchAllFilter() {
       filters = new ArrayList<Filter>();
    }
    public void addFilter(Filter f) {
       filters.add(f);
    }
    //we'll see .satisfies in a second
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    private ArrayList<Filter> filters;
    public MatchAllFilter() {
        filters = new ArrayList<Filter>();
    }
    public void addFilter(Filter f) {
        filters.add(f);
    }
    //we'll see .satisfies in a second
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    private ArrayList<Filter> filters;
    public MatchAllFilter() {
       filters = new ArrayList<Filter>();
    }
    public void addFilter(Filter f) {
       filters.add(f);
    }
    //we'll see .satisfies in a second
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    private ArrayList<Filter> filters;
    public MatchAllFilter() {
        filters = new ArrayList<Filter>();
    }
    public void addFilter(Filter f) {
        filters.add(f);
    }
    //we'll see .satisfies in a second
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
      for (Filter f: filters) {
        if (!f.satisfies(qe)) {
           return false;
        }
      }
      return true;
    }
}
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
        for (Filter f: filters) {
            if (!f.satisfies(qe)) {
                return false;
            }
        }
        return true;
    }
}
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
        for (Filter f: filters) {
            if (!f.satisfies(qe)) {
                return false;
            }
        }
        return true;
    }
}
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
      for (Filter f: filters) {
        if (!f.satisfies(qe)) {
           return false;
        }
      }
      return true;
    }
}
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
      for (Filter f: filters) {
        if (!f.satisfies(qe)) {
          return false;
        }
      }
      return true;
    }
}
```

Writing MatchAllFilter

```
public class MatchAllFilter implements Filter{
    //fields and constructors elided
    public boolean satisfies(QuakeEntry qe) {
      for (Filter f: filters) {
        if (!f.satisfies(qe)) {
           return false;
        }
      }
      return true;
    }
}
```

MatchAll: Flexible Combination

- MatchAllFilter:
  - Can combine any set of Filters
  - Uses **`satisfies`** to check all Filters it has stored in ArrayList<Filter>
- Could write other combinations
  - MatchAnyFilter