Filtering Data

Interfaces to Avoid Duplication

Duke
UNIVERSITY

Filter by Magnitude

```
public ArrayList<QuakeEntry>
filterByMagnitude(ArrayList<QuakeEntry> quakeData,
                  double magMin) {

   ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
   for(QuakeEntry qe : quakeData) {
     if (qe.getMagnitude() >= magMin) {
       answer.add(qe);
     }
   }
   return answer;
}
```

- First: look at two filters
  - See similarities

Filter by Distance

```
public ArrayList<QuakeEntry>
filterByDistanceFrom(ArrayList<QuakeEntry> quakeData,
                     double distMax,
                     Location from) {
   ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
   for(QuakeEntry qe : quakeData) {
     if (qe.getLocation().distanceTo(from) <= distMax) {
       answer.add(qe);
     }
   }
   return answer;
}
```

- First: look at two filters
  - See similarities

Filter by Distance

```
public ArrayList<QuakeEntry>
filterByDistanceFrom(ArrayList<QuakeEntry> quakeData,
                     double distMax,
                     Location from) {
   ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
   for(QuakeEntry qe : quakeData) {
     if (qe.getLocation().distanceTo(from) <= distMax) {
       answer.add(qe);
     }
   }
   return answer;
}
```

- First: look at two filters
  - See similarities

Filter by Distance

```
public ArrayList<QuakeEntry>
filterByDistanceFrom(ArrayList<QuakeEntry> quakeData,
                     double distMax,
                     Location from) {
    ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
    for(QuakeEntry qe : quakeData) {
      if (qe.getLocation().distanceTo(from) <= distMax) {
        answer.add(qe);
      }
    }
    return answer;
}
```

- First: look at two filters
  - See similarities

Filter by Distance

```
public ArrayList<QuakeEntry>
filterByDistanceFrom(ArrayList<QuakeEntry> quakeData,
                     double distMax,
                     Location from) {
   ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
   for(QuakeEntry qe : quakeData) {
     if (qe.getLocation().distanceTo(from) <= distMax) {
       answer.add(qe);
     }
   }
   return answer;
}
```

- First: look at two filters
  - See similarities

Idea: Parameterize—How to Filter

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

- Generic method: parameterized by Filter

Idea: Parameterize—How to Filter

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

- Generic method: parameterized by Filter

Idea: Parameterize—How to Filter

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

- Generic method: parameterized by Filter

Idea: Parameterize—How to Filter

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

But how do you make Filters with different .satisfies() methods?

- Generic method: parameterized by Filter

Interface: Type That Promises Method(s)

```
public interface Filter {
        public boolean satisfies(QuakeEntry qe);

}
```

- Filter is an **interface**

Interface: Type That Promises Method(s)

```
public interface Filter {
        public boolean satisfies(QuakeEntry qe);

}
```

- Filter is an **interface**

Interface: Type That Promises Method(s)

```
public interface Filter {
        public boolean satisfies(QuakeEntry qe);

}
```

- Filter is an **interface**
  - Type which promises certain methods

Interface: Type That Promises Method(s)

```
public interface Filter {
        public boolean satisfies(QuakeEntry qe);

}
```

- Filter is an **interface**
  - Type which promises certain methods
  - Classes can **implement** an interface
    - Must define the promised methods
    - Can be treated as the interface type

Classes Can Implement Interfaces

```
public class MinMagFilter implements Filter {
    private double magMin;
    public MinMagFilter(double min) {
        magMin = min;
    }
    public boolean satisfies(QuakeEntry qe) {
        return qe.getMagnitude() >= magMin;
    }
}
```

Classes Can Implement Interfaces

```
public class MinMagFilter implements Filter {
    private double magMin;
    public MinMagFilter(double min) {
        magMin = min;
    }
    public boolean satisfies(QuakeEntry qe) {
        return qe.getMagnitude() >= magMin;
    }
}
```

Classes Can Implement Interfaces

```
public class MinMagFilter implements Filter {
    private double magMin;
    public MinMagFilter(double min) {
        magMin = min;
    }
    public boolean satisfies(QuakeEntry qe) {
        return qe.getMagnitude() >= magMin;
    }
}
```

Classes Can Implement Interfaces

```
public class MinMagFilter implements Filter {
    private double magMin;
    public MinMagFilter(double min) {
        magMin = min;
    }
    public boolean satisfies(QuakeEntry qe) {
        return qe.getMagnitude() >= magMin;
    }
}
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
Filter f = new MinMagFilter(4.0);
ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
Filter f = new MinMagFilter(4.0);
ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
Filter f = new MinMagFilter(4.0);
ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
Filter f = new MinMagFilter(4.0);
ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
 Filter f = new MinMagFilter(4.0);
 ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
f = new DistanceFilter(myLoc,100);
ArrayList<QuakeEntry> shallowQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
 Filter f = new MinMagFilter(4.0);
 ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
f = new DistanceFilter(myLoc,100);
ArrayList<QuakeEntry> shallowQuakes  = filter(list, f);
```

Compatible Types

```
public ArrayList<QuakeEntry>
filter(ArrayList<QuakeEntry> quakeData,
       Filter f) {
  ArrayList<QuakeEntry> answer = new ArrayList<QuakeEntry>();
  for(QuakeEntry qe : quakeData) {
    if (f.satisfies(qe)) {
      answer.add(qe);
    }
  }
  return answer;
}
```

```
 Filter f = new MinMagFilter(4.0);
 ArrayList<QuakeEntry> largeQuakes  = filter(list, f);
f = new DistanceFilter(myLoc,100);
ArrayList<QuakeEntry> shallowQuakes  = filter(list, f);
```