Streams

Example

```
return intStream.filter(c -> vowels.contains(Character.toUpperCase(c))).count();
```

Filters based on the given lambda function

- .equals(var) ensures that the variable in stream is equal to the var
- .peek() allows you to apply basic operations on each item in the stream. Generally used for printing them out.
- .map() takes a shortened lambda function **object :: getMethod()**. Allows you to get a class member from the object and apply operations on that.
- .collect() gets everything that you have and puts it in a list. Collectors.toList() is common in the brackets of .collect().

e.g.

```
public static List<String> visitInContinent(List<Country> countries,

            Continent continent) {

        return countries.stream().filter(c -> c.getContinent().equals(continent)).peek(c ->
c.visit()).map(Country::getName).collect(Collectors.toList());

    }
```

Lambda Functions

Lambda functions are on the go functions without defining `public …`

Lambda Example

```
public static List<Person> sortByAgeThenName(List<Person>
people) {
        // write your code here
        people.sort((Person p1, Person p2) -> {
            if (p1.getAge() == p2.getAge()) {
                return
p1.getName().compareTo(p2.getName());
            } else {
                return p1.getAge() - p2.getAge();
            }
        });

        return people;
    }
```

.forEach()

Collection.forEach(lambda function here)
continents.forEach(x -> x.getCountry().equals("Australia"));
returns continents which have a country of Australia inside ^^^

Generics

```
public class Pair<T, K> {
    private T first;
    private K second;

    public Pair(T one, K two) {
        this.first = one;
        this.second = two;
    }

    public T getFirst() {
        return first;
    }

    public K getSecond() {
        return second;
    }

    public String toString() {
        return String.format("(%s, %s)",first.toString(),
second.toString());
    }
}
```

Wildcards

```
Public void printList(List<?> toPrint) {
         For(Object item : toPrint) {
                  …)}}
```

Coupling/Cohesion

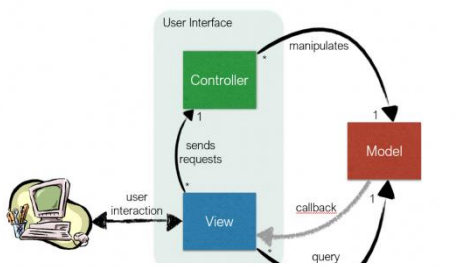
- Want high cohesion and low coupling

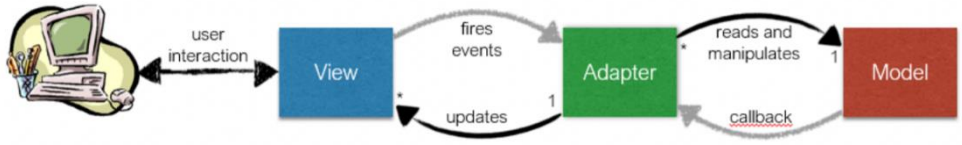
Law of Demeter

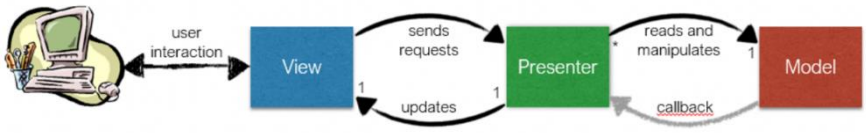
- A method can call other methods in its own class
- A method can call methods on its class' data members but not on the data member's members
- A method can call methods on its parameters
- If a method creates an object, it can call methods on that object.
- Avoid changed messages such as a getB().getC().doSomething().

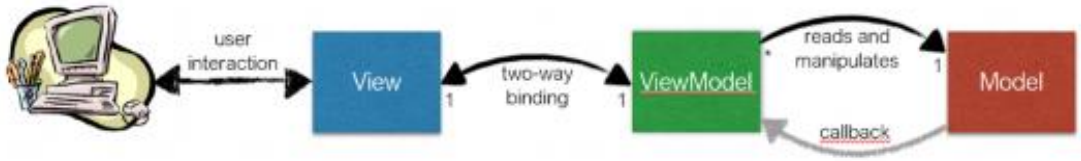
Model view Controller

- **Model view controller** displays information and the controller processes input. Together they make up the interface
- **Model view adapter** isolates the view from the model by using an adapter, the view and model only needs to know about the interfaces implemented by the adapter. In theory the adapter can work with multiple views, but this is less common in practice.
- **Model view presenter** isolate the view by using a presenter. View only knows about the presenter
    - – ViewModel encapsulates view state and logic









File Readers/Writer

BufferedReader(FileReader(filename))

Example saveToFile Method

```
public void saveToFile(Writer writer) {
        // write your code here
        BufferedWriter buffWrite = new
BufferedWriter(writer);
        try {
            buffWrite.write(this.toString());
            buffWrite.close();
        } catch (IOException e) {
            System.out.print("IOException occurred");
        }

    }
```

BufferedReader Example

```
try {
            BufferedReader buffRead = new
BufferedReader(reader);
            while ((line = buffRead.readLine()) != null) {
                String[] toks;
                 toks = line.split(" ");n
                for (String t : toks) {
                    try {
                        double i = Double.parseDouble(t);
                        count += i;
                    }  catch (NumberFormatException e) {}
                }
            }
            return count;
        } catch (IOException e) {
            return 0;
        }
```