# Introducing the Recommender

Filtering Recommendations

Duke
UNIVERSITY

## Generality and Efficiency

- Average ratings for all movies: too general
  - Recommendations for new movies: after 2012
  - For Action, Romantic, Comedies
  - Movies under two-hours directed by Spielberg
- More efficient Movie and Rating access
  - Use HashMap with MovieID rather than search through ArrayList of movies or ratings
- Refactor program for efficiency, minimize changes in existing code: Open/Closed

Duke
UNIVERSITY

## Refactoring Rater.java

- Refactoring code doesn't add functionality, or change API/external interface
  - More readable, more efficient, maintainable
- Rater.java stores ratings in ArrayList
  - Loop over list to find rating for 1201607
  - Inefficient with thousands of raters and hundreds of ratings for each rater

# Create Interface for Rater

- First step to create efficient Rater: interface
    - Make copy of Rater.java, PlainRater.java
    - Turn Rater.java into an interface

```
public interface Rater
{
    public void addRating(String item, double rating);
    public boolean hasRating(String item);
    public String getID();
    public double getRating(String item);
    public int numRatings();
    public ArrayList<String> getItemsRated();

}
```

## Testing Existing Framework

- Programs already use Rater in reading and constructing average recommendations
    - Continue to use Rater, but change code
    - `Rater rater = new PlainRater();`
    - No other changes needed! Open/Closed!
- Now create EfficientRater.java
    - Same interface, but uses HashMap for efficiency in searching for movieID
    - `Rater rater = new EfficientRater();`

# Filtering Movies

- Use Filter.java interface to get recommendations for movies after 2012, or over three hours long, or Action Adventure

```
public interface Filter {
  public boolean satisfies(String id);
}
```

## Filtering Movies

- Use Filter.java interface to get recommendations for movies after 2012, or over three hours long, or Action Adventure
- Creating YearAfterFilter or GenreFilter?

```
public class YearAfterFilter implements Filter {
  private int myYear;
  public YearAfterFilter(int year) {
    myYear = year;
  }
…
```

## Filtering Movies

- Use Filter.java interface to get recommendations for movies after 2012, or over three hours long, or Action Adventure
- Creating YearAfterFilter or GenreFilter?

```
public class GenreFilter implements Filter {
  private String myGenre;
  public GenreFilter(String genre) {
    myGenre = genre;
  }
…
```

## Filtering Movies

- Use Filter.java interface to get recommendations for movies after 2012, or over three hours long, or Action Adventure
- Creating YearAfterFilter or GenreFilter?
    - How to access movie information?

```
public class YearAfterFilter implements Filter {
  private int myYear;
  public boolean satisfies(String id){
    // access year of movie with given id
    // compare to myYear, return value
  }
```

## Filtering Movies

- Use Filter.java interface to get recommendations for movies after 2012, or over three hours long, or Action Adventure
- Creating YearAfterFilter or GenreFilter?
  - How to access movie information?
- Must either pass movie info in via filter constructors or provide other access
  - Create MovieDatabase class, efficiency and functionality!

# MovieDatabase Class

- Use MovieDatabase class for efficiency and functionality
  - Same concept as used in EfficientRater to look up ratings given movie ID as key in HashMap
  - MovieID (string) is key, Movie is value
- MovieDatabase.java uses static methods, helps with Filter and other classes
  - Similar to a "real" database, given ID return Movie, or all IDs satisfying a Filter class

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies

```
public ArrayList<Rating> getAverageRatings(Filter f,
                                           int minimalRaters){
    ArrayList<Rating> list = new ArrayList<Rating>();
    ArrayList<String> movies = MovieDatabase.filterBy(f);
    for(String id : movies) {
        // calculate average for id
```

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs

```
public ArrayList<Rating> getAverageRatings(Filter f,
                                            int minimalRaters){
    ArrayList<Rating> list = new ArrayList<Rating>();
    ArrayList<String> movies = MovieDatabase.filterBy(f);
    for(String id : movies) {
        // calculate average for id
```

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter

```
public ArrayList<Rating> getAverageRatings(Filter f,
                                                   int minimalRaters){
    ArrayList<Rating> list = new ArrayList<Rating>();
    ArrayList<String> movies = MovieDatabase.filterBy(f);
    for(String id : movies) {
        // calculate average for id
```

## Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance

```
YearAfterFilter yf = new YearAfterFilter(2012);
GenreFilter gf = new GenreFilter("Romance");
AllFilters af = new AllFilters();
af.addFilter(yf);
af.addFilter(gf);
ArrayList<Rating> list = getAverageRatings(af,10);
```

## Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance

```
YearAfterFilter yf = new YearAfterFilter(2012);
GenreFilter gf = new GenreFilter("Romance");
AllFilters af = new AllFilters();
af.addFilter(yf);
af.addFilter(gf);
ArrayList<Rating> list = getAverageRatings(af,10);
```

Duke
UNIVERSITY

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance

```
YearAfterFilter yf = new YearAfterFilter(2012);
GenreFilter gf = new GenreFilter("Romance");
AllFilters af = new AllFilters();
af.addFilter(yf);
af.addFilter(gf);
ArrayList<Rating> list = getAverageRatings(af,10);
```

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
    - Use MovieDatabase.filterBy to get IDs
    - To get all movies, use TrueFilter
    - Use other filters as needed: 2012 and Romance

```
YearAfterFilter yf = new YearAfterFilter(2012);
GenreFilter gf = new GenreFilter("Romance");
AllFilters af = new AllFilters();
af.addFilter(yf);
af.addFilter(gf);
ArrayList<Rating> list = getAverageRatings(af,10);
```

## Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance

```
YearAfterFilter yf = new YearAfterFilter(2012);
GenreFilter gf = new GenreFilter("Romance");
AllFilters af = new AllFilters();
af.addFilter(yf);
af.addFilter(gf);
ArrayList<Rating> list = getAverageRatings(af,10);
```

# Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance

```
 1  8.281   The Theory of Everything
 2  8.211   Her
 3  8.188   The Perks of Being a Wallflower
 4  8.136   Silver Linings Playbook
 5  8.118   About Time
 6  8.033   The Fault in Our Stars
 7  7.705   The Great Gatsby
 8  7.250   Warm Bodies
 9  6.313   Transcendence
10  5.364   The Other Woman
```

## Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance
- Power of Interfaces, refactoring, using existing code in new contexts

## Using Filters, Interfaces, Database

- Movies not accessed via field myMovies
  - Use MovieDatabase.filterBy to get IDs
  - To get all movies, use TrueFilter
  - Use other filters as needed: 2012 and Romance
- Power of Interfaces, refactoring, using existing code in new contexts
  - More efficient code too!