Assignment 2 Serialisation

Agenda

- XMLSerializer & ID management
- Ingesting CSV
- Modelling

Serialiser

- An abstraction to encapsulate persistence mechanism
- Push objects onto the stack
- All objects pushed are then saved in a single ‘write’ operation
- If read is called, a persistence state is restored… and can be recovered by popping the stack

```
public interface Serializer
{
  void push(Object o);
  Object pop();
  void write() throws Exception;
  void read() throws Exception;
}
```

```
public class XMLSerializer implements Serializer
{
  private Stack stack = new Stack();
  private File file;

  public XMLSerializer(File file)
  {
    this.file = file;
  }

  public void push(Object o)
  {
    stack.push(o);
  }

  public Object pop()
  {
    return stack.pop();
  }

  @SuppressWarnings("unchecked")
  public void read() throws Exception
  {
    ObjectInputStream is = null;

    try
    {
      XStream xstream = new XStream(new DomDriver());
      is = xstream.createObjectInputStream(new FileReader(fi
      stack = (Stack) is.readObject();
    }
    finally
    {
      if (is != null)
      {
        is.close();
      }
    }
  }
```

```
  public void write() throws Exception
  {
    ObjectOutputStream os = null;

    try
    {
      XStream xstream = new XStream(new DomDriver());
      os = xstream.createObjectOutputStream(new FileWriter(file));
      os.writeObject(stack);
    }
    finally
    {
      if (os != null)
      {
        os.close();
      }
    }
  }
}
```

```
class PacemakerAPI
{
  …
  public void load() throws Exception
  {
    serializer.read();
    activitiesIndex = (Map<Long, Activity>) serializer.pop();
    emailIndex      = (Map<String, User>)   serializer.pop();
    userIndex       = (Map<Long, User>)     serializer.pop();
  }

  public void store() throws Exception
  {
    serializer.push(userIndex);
    serializer.push(emailIndex);
    serializer.push(activitiesIndex);
    serializer.write();
  }
```

XML Serliaizer

ID Management in Pacemaker

```
public class User
{
  static Long   counter = 0l;

  public Long   id;
  public String firstName;
  public String lastName;
  public String email;
  public String password;

  public Map<Long, Activity> activities = new HashMap<>();

  public User(String firstName, String lastName, String gender, String age, String occupation)
  {
    this.id         = counter++;
    this.firstName = firstName;
    this.lastName = lastName;
    this.gender = gender;
    this.age = age;
    this.occupation = occupation;
  }
}
```

- static counter, incremented by 1 as each object is created.

- simple mechanism - but must be integrated into persistence scheme

Bug Symptoms

- Create three users
- List the users (note the ids)
- exit

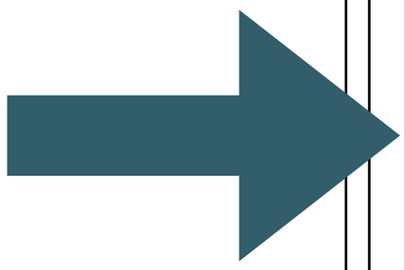
```
Welcome to pacemaker-console - ?help for instructions
pm> gu
[]
pm> cu a a a a
pm> cu b b b b
pm> cu c c c c
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 3,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 3,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 3,
  "id": 2,
  "email": "c"
}]
pm> exit
```

```
Welcome to pacemaker-console - ?help for instructions
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 0,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 0,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 0,
  "id": 2,
  "email": "c"
}]
pm>
```

```
pm> cu e e e e
pm> gu
[models.User
{
  "firstName": "e",
  "lastName": "e",
  "password": "e",
  "activities": {},
  "counter": 1,
  "id": 0,
  "email": "e"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 1,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 1,
  "id": 2,
  "email": "c"
}]
pm>
```

- Start app again
- list all users
- add new user
- Problem!

```
Welcome to pacemaker-console - ?help for instructions
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 0,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 0,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 0,
  "id": 2,
  "email": "c"
}]
pm>
```



```
pm> cu e e e e
pm> gu
[models.User
{
  "firstName": "e",
  "lastName": "e",
  "password": "e",
  "activities": {},
  "counter": 1,
  "id": 0,
  "email": "e"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 1,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 1,
  "id": 2,
  "email": "c"
}]
pm>
```

new value overwrites existing value with ID 0

Bug

- When store() called, push:
  - userIndex
  - emailIndex
  - activitiedIndex
- onto stack and the write out
- When load() called:
  - do the reverse

```
class PacemakerAPI
{
  …
  public void load() throws Exception
  {
    serializer.read();
    activitiesIndex = (Map<Long, Activity>) serializer.pop();
    emailIndex      = (Map<String, User>)   serializer.pop();
    userIndex       = (Map<Long, User>)     serializer.pop();
  }

  public void store() throws Exception
  {
    serializer.push(userIndex);
    serializer.push(emailIndex);
    serializer.push(activitiesIndex);
    serializer.write();
  }
```

- What about counter?

```
public class User
{
  static Long   counter = 0l;

  public Long   id;
  public String firstName;
  public String lastName;
  public String email;
  public String password;
  ...
```

- counter not serialized, so will be reset to 0 each time program is launched

Bugfix

- Write and read the counters as part of the serialisation mechanism

- When app restarts, the counter will continue from last value (as opposed to zero)

```
public void load() throws Exception
{
  serializer.read();
  activitiesIndex = (Map<Long, Activity>) serializer.pop();
  emailIndex      = (Map<String, User>)   serializer.pop();
  userIndex       = (Map<Long, User>)     serializer.pop();
  User.counter    = (Long) serializer.pop();
}

public void store() throws Exception
{
  serializer.push(User.counter);
  serializer.push(userIndex);
  serializer.push(emailIndex);
  serializer.push(activitiesIndex);
  serializer.write();
}
```

```
Welcome to pacemaker-console - ?help for instructions
pm> cu a a a a
pm> cu b b b b
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 2,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 2,
  "id": 1,
  "email": "b"
}]
pm> exit
```

New Element inserted here

```
Welcome to pacemaker-console - ?help for instructions
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 2,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 2,
  "id": 1,
  "email": "b"
}]
pm> cu c c c c
pm> gu
[models.User
{
  "firstName": "a",
  "lastName": "a",
  "password": "a",
  "activities": {},
  "counter": 3,
  "id": 0,
  "email": "a"
}, models.User
{
  "firstName": "b",
  "lastName": "b",
  "password": "b",
  "activities": {},
  "counter": 3,
  "id": 1,
  "email": "b"
}, models.User
{
  "firstName": "c",
  "lastName": "c",
  "password": "c",
  "activities": {},
  "counter": 3,
  "id": 2,
  "email": "c"
}]
```

Other counters?

- Need to store counter value for each model class

```
public void load() throws Exception
{
  serializer.read();
  activitiesIndex = (Map<Long, Activity>) serializer.pop();
  emailIndex      = (Map<String, User>)   serializer.pop();
  userIndex       = (Map<Long, User>)     serializer.pop();
  User.counter    = (Long) serializer.pop();
}

public void store() throws Exception
{
  serializer.push(User.counter);
  serializer.push(userIndex);
  serializer.push(emailIndex);
  serializer.push(activitiesIndex);
  serializer.write();
}
```

Unit Test

- Why was this bug not detected earlier?
- What should the first course of action be when the bug is uncovered?
  - Write unit test to reproduce the bug
  - The proceed to explore various solutions, with the test as a guide

Longer Term Solution

- Database Technology
  - SQL Based
    - MySQL + Object Relational Mapping (ORM)

      Hibernate (play framework)
  - NoSQL
    - Key/Value Stores
    - Document Database

      MongoDb
    - Graph Databases

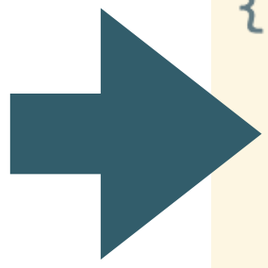
      neo4J

Hibernate/SQL Example

- Persistence mechanisms often have an ID generation capability to handle and manage unique IDs for objects

- In Play Framework / Hibernate the annotation “GeneratedValue” serves this purpose

- The Database will generated the value for the annotated value..



```java
package models;

import java.util.List;

import javax.persistence.*;
import play.db.ebean.*;

import com.google.common.base.Objects

@SuppressWarnings("serial")
@Entity
@Table(name="my_user")
public class User extends Model
{
  @Id
  @GeneratedValue
  public Long   id;
  public String firstname;
  public String lastname;
  public String email;
  public String password;

  public User()
  {
  }
```

New to Graph Databases?

Explore the World of Graphs – From Query Efficiency to Business Performance

Dive into Code

From data modeling and drivers to scalability and Cypher queries, learn more about graph database development.

Get Me Started

Witness Bottom-Line Impact

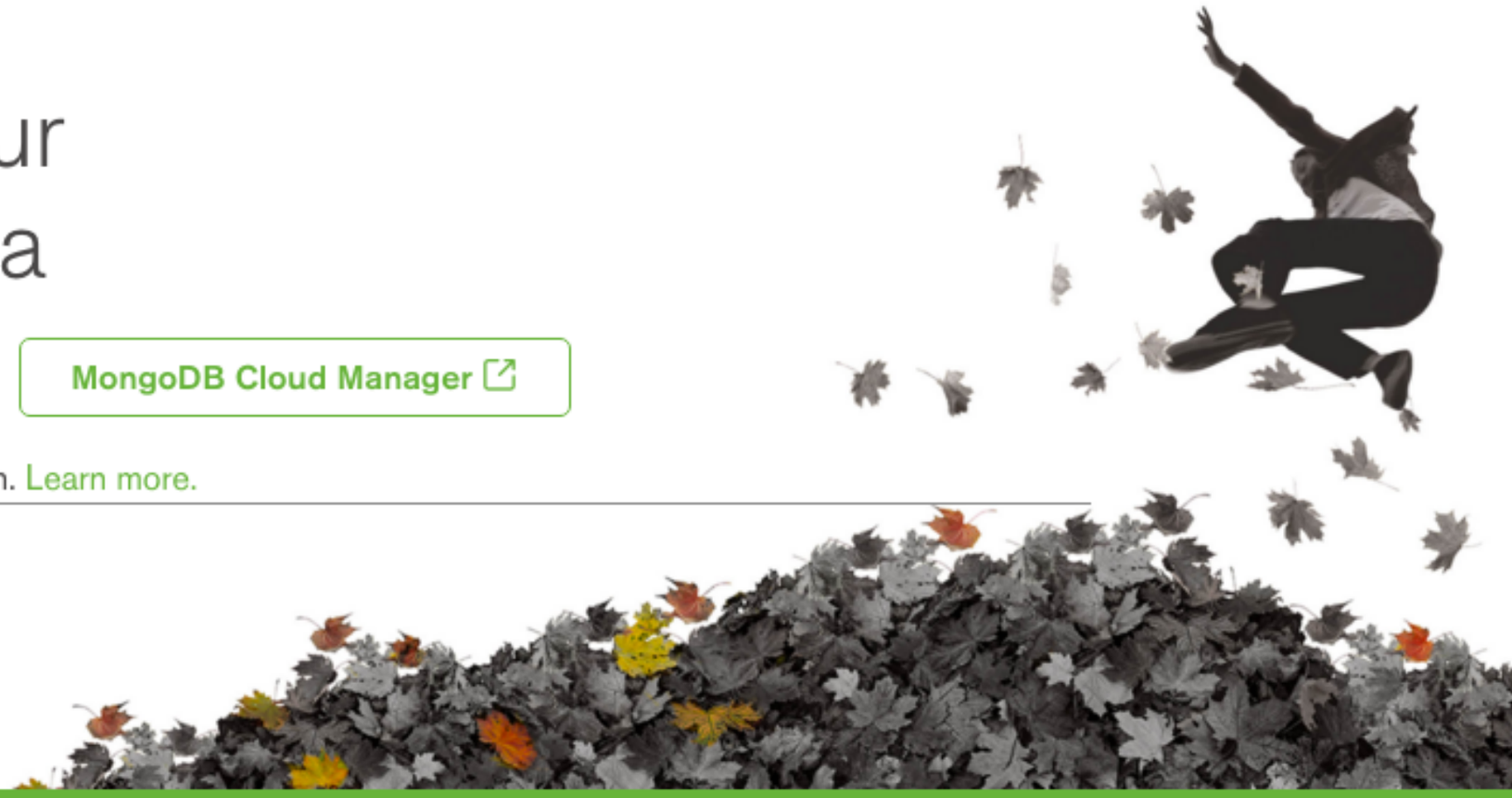
Leverage your big data relationships with graph technology that delivers enterprise-level insights all in real time.

Why Graph Databases?

Launch your **GIANT** idea

Download MongoDB | MongoDB Cloud Manager

MongoDB **v3.2** is coming soon. Learn more.



MongoDB Professional with **Cloud Manager**

Run smoothly with expert support and a comprehensive management platform that includes monitoring, automation, and cloud backups.

Learn More

Insert Data with Java Driver

Overview

You can use the insertOne method to add documents to a collection in MongoDB. If you attempt to add documents to a collection that does not exist, MongoDB will create the collection for you.

Prerequisites

Follow the Connect to MongoDB step to connect to a running MongoDB instance and declare and define the variable **db** to access the **test** database.

Include the following **import** statement.

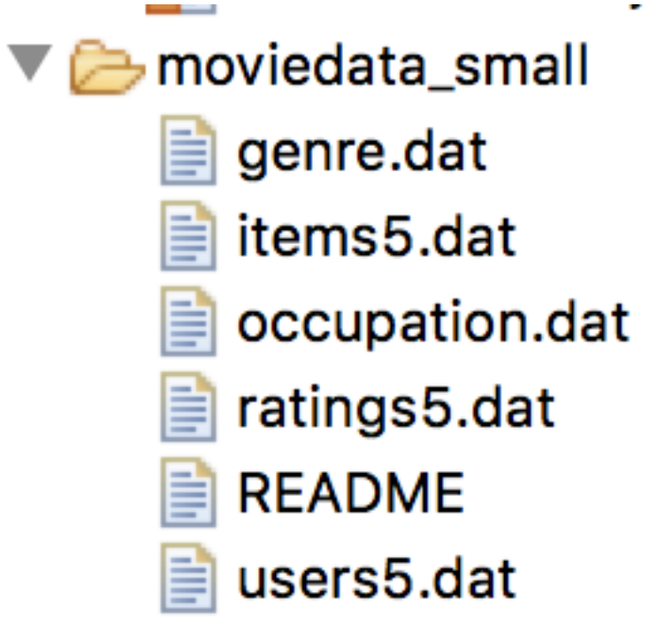
```
import org.bson.Document;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Locale;

import static java.util.Arrays.asList;
```

Insert a Document

Insert a document into a collection named **restaurants**. The operation will create the collection if the

Ingesting CSV

Ingesting CSV




```
1|Leonard|Hernandez|24|M|technician|85711
2|Melody|Roberson|53|F|other|94043
3|Gregory|Newton|23|M|writer|32067
4|Oliver|George|24|M|technician|43537
5|Jenna|Parker|33|F|other|15213
```

- ‘prime’ command could read this css file and store the users in our user index.

Strategy?

- Think about ‘testability’:
  - Design a solution that can be easily independently tested prior to integration into your application
  - This involves isolation the key functionality & making sure there is a simple interface against which unit tests can be composed

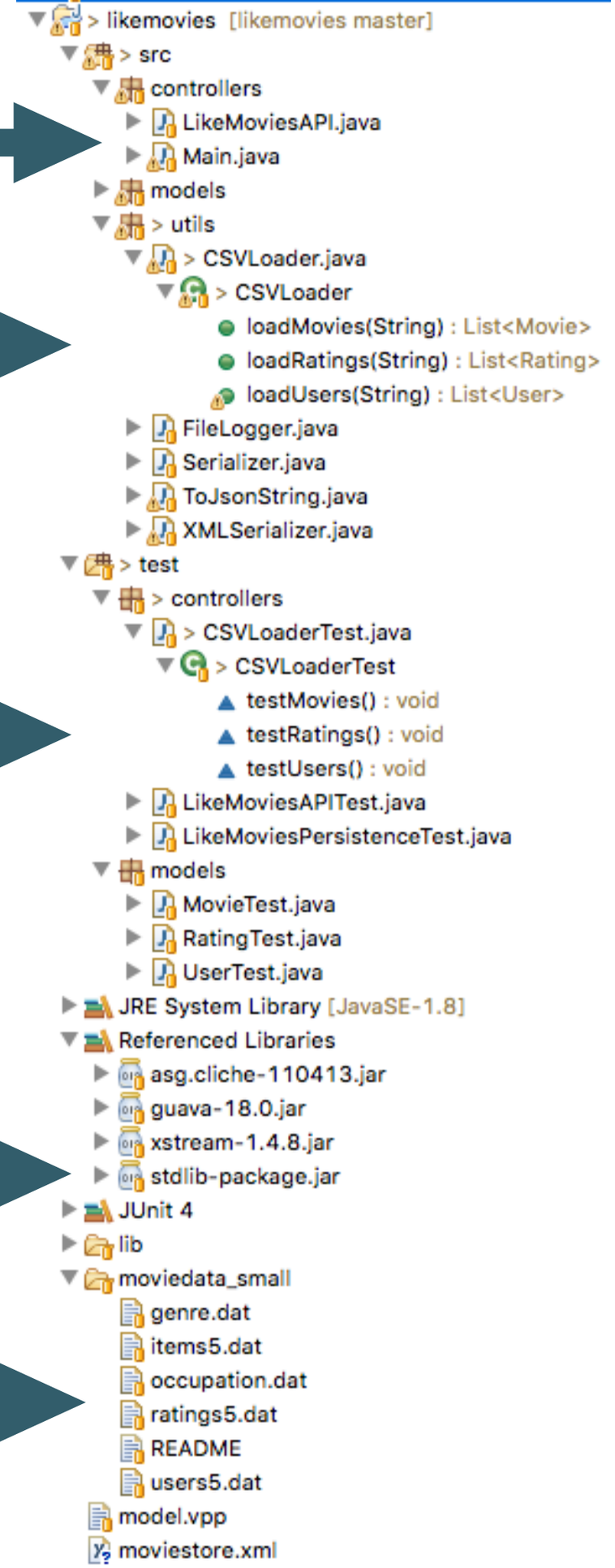
5: New ‘prime’ command

4: CVSLoader Implementation

3: CSVLoaderTest

2: Support library

1: test data



CSVLoaderTest

- Each test should:
  - Verify that the test .dat file exists
  - Call the appropriate method to load the dat file contents into an array of Users/Movies/Ratings
  - Verify that the objects in the array matches the contents of the dat file (use fixture for this)

```
public class CSVLoaderTest
{

    @Test
    void testUsers()
    {

    }

    @Test
    void testMovies()
    {

    }

    @Test
    void testRatings()
    {

    }
}
```

CVSLoader

- Propagate exceptions
- Return array of model objects created as each line is read from the css file

```
public class CSVLoader
{
  public List<User> loadUsers(String filename) throws Exception
  {
    return null;
  }

  public List<Movie> loadMovies (String filename) throws Exception
  {
    return null;
  }

  public List<Rating> loadRatings (String filename) throws Exception
  {
    return null;
  }
}
```

LikeMoviesAPI

```
public void prime() throws Exception
{
  CSVLoader loader = new CSVLoader();
  List <User> users = loader.loadUsers("moviedata_small/users5.dat");
  for (User user : users)
  {
    userIndex.put(user.id,user);
  }

  // Load Movies
  // Load Ratings...
}
```

- Load the uses into a local array
- Add the contents of this array to the userIndex

Main

```
@Command(description="prime")
public void prime () throws Exception
{
  likeMovies.prime();
}
```

- Initial Prime command

- What if the dat file is not present?

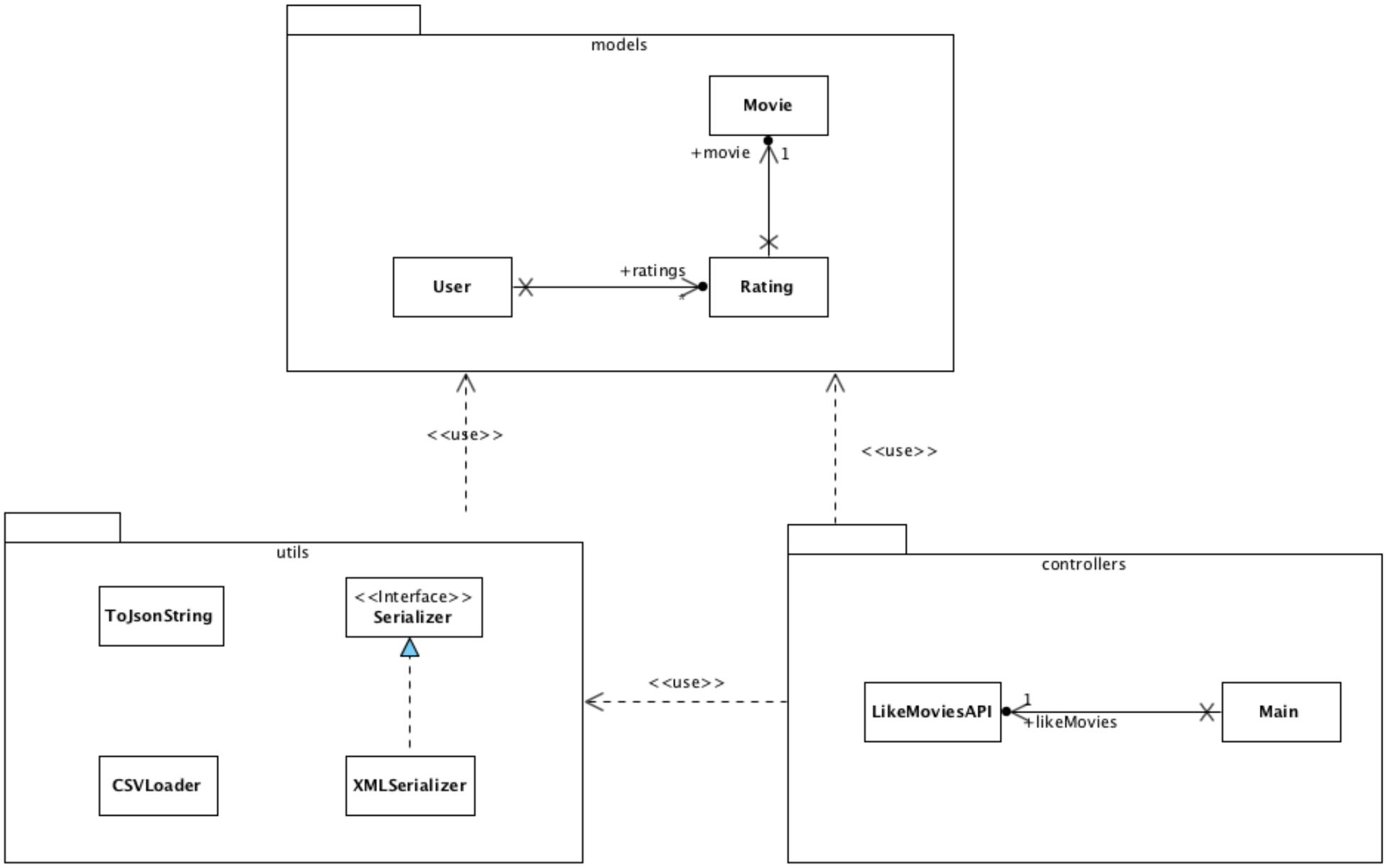
  - Catch the exception in the prime command and give the user a useful error message

  - Recover from file not found error? Perhaps prompt user for alternative file name and try again. Change API to do this

Parsing CSV File Example

```
public List<User> loadUsers(String filename) throws Exception
{
  File usersFile = new File(filename);
  In inUsers = new In(usersFile);

  String delims = "[|]";
  List<User> users = new ArrayList<User>();
  while (!inUsers.isEmpty())
  {
    String userDetails  = inUsers.readLine();
    String[] userTokens = userDetails.split(delims);
    if (userTokens.length == 7)
    {
      String id         = userTokens[0];
      String firstName  = userTokens[1];
      String lastName   = userTokens[2];
      String age        = userTokens[3];
      String gender     = userTokens[4];
      String occupation = userTokens[6];

      users.add(new User(firstName, lastName, gender, age, occupation));
    }
    else
    {
      throw new Exception("Invalid member length: " + userTokens.length);
    }
  }
  return users;
}
```

Model


models
Movie
+movie
1
User
+ratings
*
Rating
<<use>>
<<use>>
utils
ToJsonString
<<Interface>>
Serializer
CSVLoader
XMLSerializer
<<use>>
controllers
LikeMoviesAPI
1
+likeMovies
Main