

Assignment 2 Bootstrap

The API + Command Line Specification

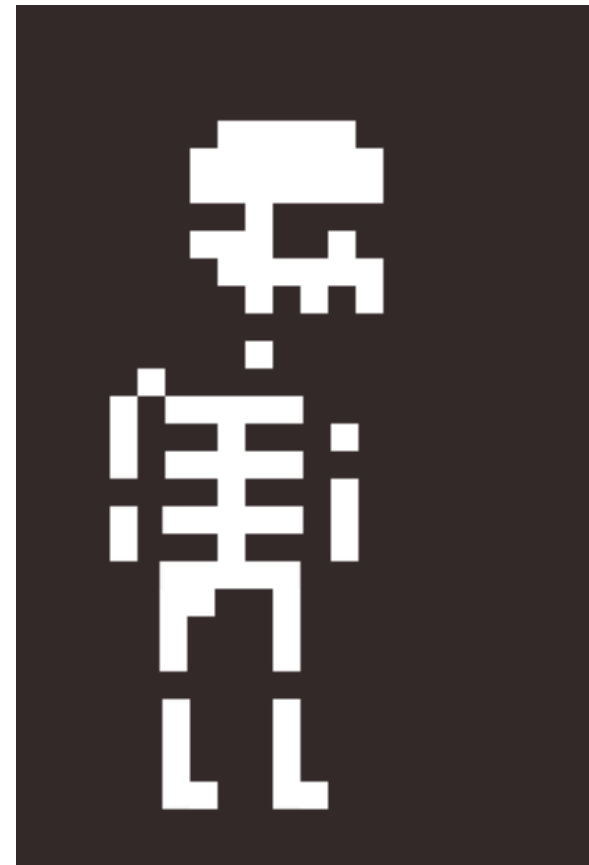
```
addUser(firstName,lastName,age,gender,occupation)
removeUser(userID)
addMovie(title, year, url)
addRating(userID, movieID, rating)
getMovie(movieID)
getUserRatings(userID)
getUserRecommendations(userID)
getTopTenMovies()
load()
write()
```

Walking Skeleton



Walking Skeleton : <http://alistair.cockburn.us/Walking+skeleton>

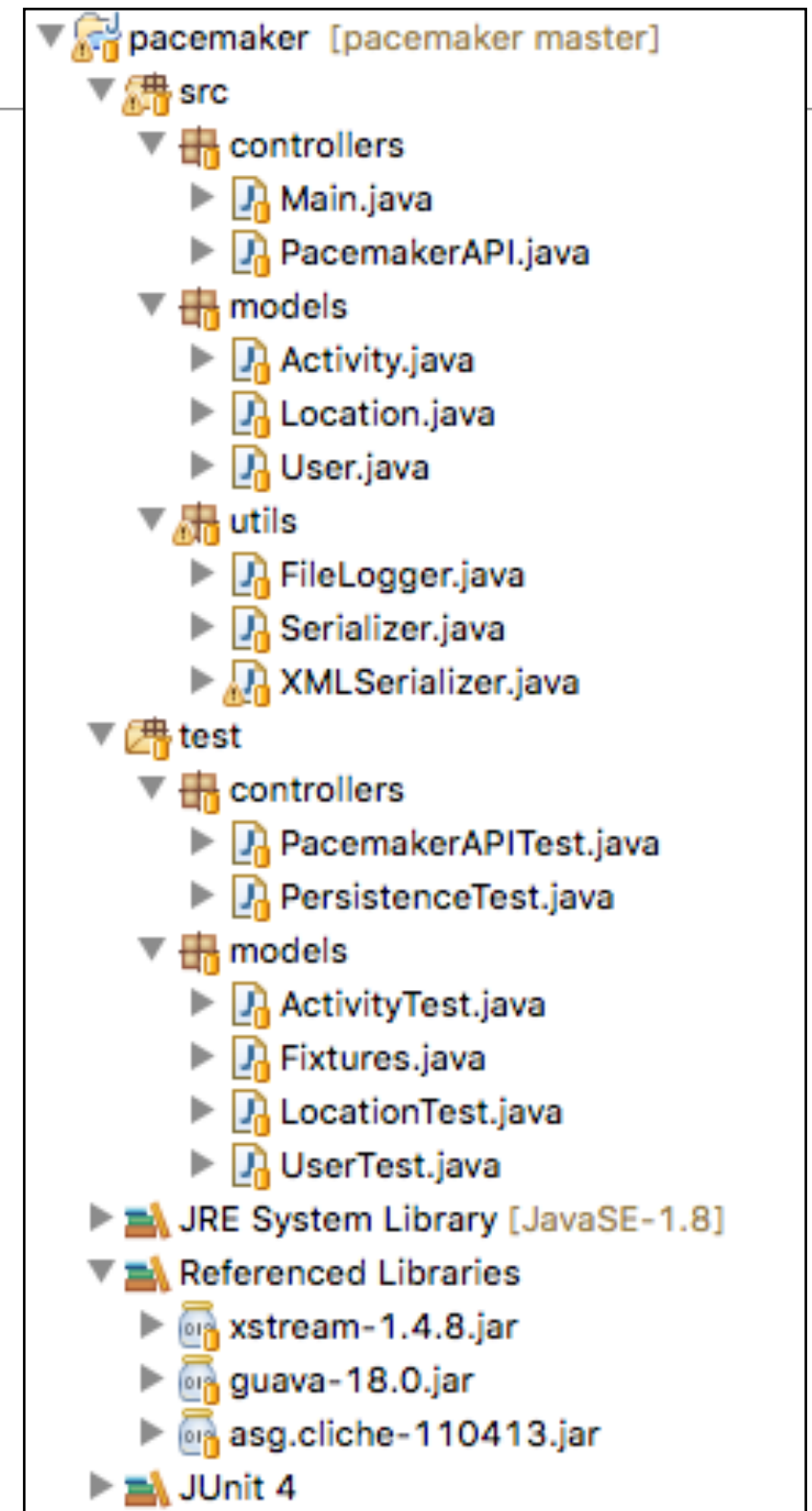
- “A Walking Skeleton is a tiny implementation of the system that performs a small end-to-end function.
- It need not use the final architecture, but it should link together the main architectural components.
- The architecture and the functionality can then evolve in parallel.”



- Useful strategy to get started...
- Enables you to make small incremental improvements from a solid foundation

Walking Skeleton Project

- Create an Eclipse project with:
 - “src” and “test” folders
 - best guess at suitable packages
 - Initial versions classes you think you will need - largely empty for the first version
- Libraries



Building a Walking Skeleton for Assignment 2

- What are the likely initial Objects?
- What data structures will be appropriate?
- How will the API be implemented?
- What strategy is to be used for the command line?
- How will the tests be organised?
- What is the persistence strategy?

```
addUser(firstName,lastName,age,gender,occupation)
removeUser(userID)
addMovie(title, year, url)
addRating(userID, movieID, rating)
getMovie(movieID)
getUserRatings(userID)
getUserRecommendations(userID)
getTopTenMovies()
load()
write()
```



12 Nov 2007

What's in a Project Name?

Give the
Project a
Name!

Since I started at [Vertigo](#), here are a few of the projects I've worked on:

- Michelangelo
- Nash
- Whiskeytown
- Gobstopper

These are our **internal project code names**. The names are chosen alphabetically from a set of items; every new project gets a name from the set. We start with A, and when we finally arrive at Z, we pick a new set of items for project name inspiration. Can you guess which set each of the above project names is from? No cheating!

We've come up with the following loose guidelines for project naming:

1. We prefer one word names.
2. They should be relatively easy to pronounce and easy to spell.
3. They have to be client friendly.
4. They should be globally unique across the company. No duplicates.
5. We need a reasonable number of items in the set to choose from, in A-Z order.

Types of Food

Video games ([Atari 2600](#), [Arcade](#), etc)

Brands of Beer

Roman Emperors

Cartoon characters / shows

Mythological names / gods

Cars

GUIDs (a [personal favorite](#))

[Gemstones](#)

Types of Coffee drinks

States

Counties

Plants

[Hitchcock films](#)

[Dog breeds](#)

[Colors](#)

Famous Explorers

[Trees](#)

[IRS Tax Forms](#)

English [monarchs](#)

Famous People (eg, [Sagan](#))

[Wikipedia](#) article names

Single letters (including unicode)

Radio [alphabet](#)

Candy brands

[Dinosaurs](#)

Historical Sites

City street names

[IKEA](#) product names

Types of Fasteners (nut, bolt, rivet, etc)

[Ski resorts](#)

[National Parks](#)

[Mountain Peaks](#)

[World War II era ships](#)

[Birds](#)

[Beaches](#)

[Bridges](#)

Web 2.0 [names](#)

Warcraft realm names

[Cheeses](#)

Countries

Cereal brands

The logo for LOVEFiLM.COM features the word "LOVE" in red, "FiLM" in black, and ".COM" in red, with a registered trademark symbol (®) to the right. The text is set against a light gray rectangular background.

LOVEFiLM.COM®

likemovie

-
- What are the likely initial Objects?
 - What data structures will be appropriate?
 - How will the API be implemented?
 - What strategy is to be used for the command line?
 - How will the tests be organised?
 - What is the persistence strategy?
 - Where do I start?

Initial Candidate Objects

- Model
 - User
 - Movie
 - Rating
- LikeMoviesAPI
- CommandShell
- Serialisers?

```
addUser(firstName,lastName,age,gender,occupation)
removeUser(userID)
addMovie(title, year, url)
addRating(userID, movieID, rating)
getMovie(movieID)
getUserRatings(userID)
getUserRecommendations(userID)
getTopTenMovies()
load()
write()
```

What data structures will be appropriate?

- Map of userId->User
- Map of movieId->Movie
- Ratings?
 - Each user holds a list of ratings objects

User

Movie

Rating

```
addUser(firstName,lastName,age,gender,occupation)
removeUser(userID)
addMovie(title, year, url)
addRating(userID, movieID, rating)
getMovie(movieID)
getUserRatings(userID)
getUserRecommendations(userID)
getTopTenMovies()
load()
write()
```

How will the API be implemented?

- Define a single LikeMoviesAPI class
- Define the data structured (userIndex, moviesIndex) as members of the API class
- Define a suitable method signature for each of the features listed here
- API does not include any UX

```
addUser(firstName,lastName,age,gender,occupati  
removeUser(userID)  
addMovie(title, year, url)  
addRating(userID, movieID, rating)  
getMovie(movieID)  
getUserRatings(userID)  
getUserRecommendations(userID)  
getTopTenMovies()  
load()  
write()
```

Command Line

- DON'T roll your own
- Use a suitable library

Cliche Command-Line Shell

Cliche is a small Java library enabling *really* simple creation of interactive command-line user interfaces.

It uses metadata and Java Reflection to determine which class methods should be exposed to end user and to provide info for user. Therefore all information related to specific command is kept in only one place: in annotations in method's header. User don't have to organize command loop, write complicated parsers/converters for primitive types, though he can implement custom converters when needed.

How simple? *So simple:*

```
package asg.cliche.sample;

import asg.cliche.Command;
import asg.cliche.ShellFactory;
import java.io.IOException;

public class HelloWorld {

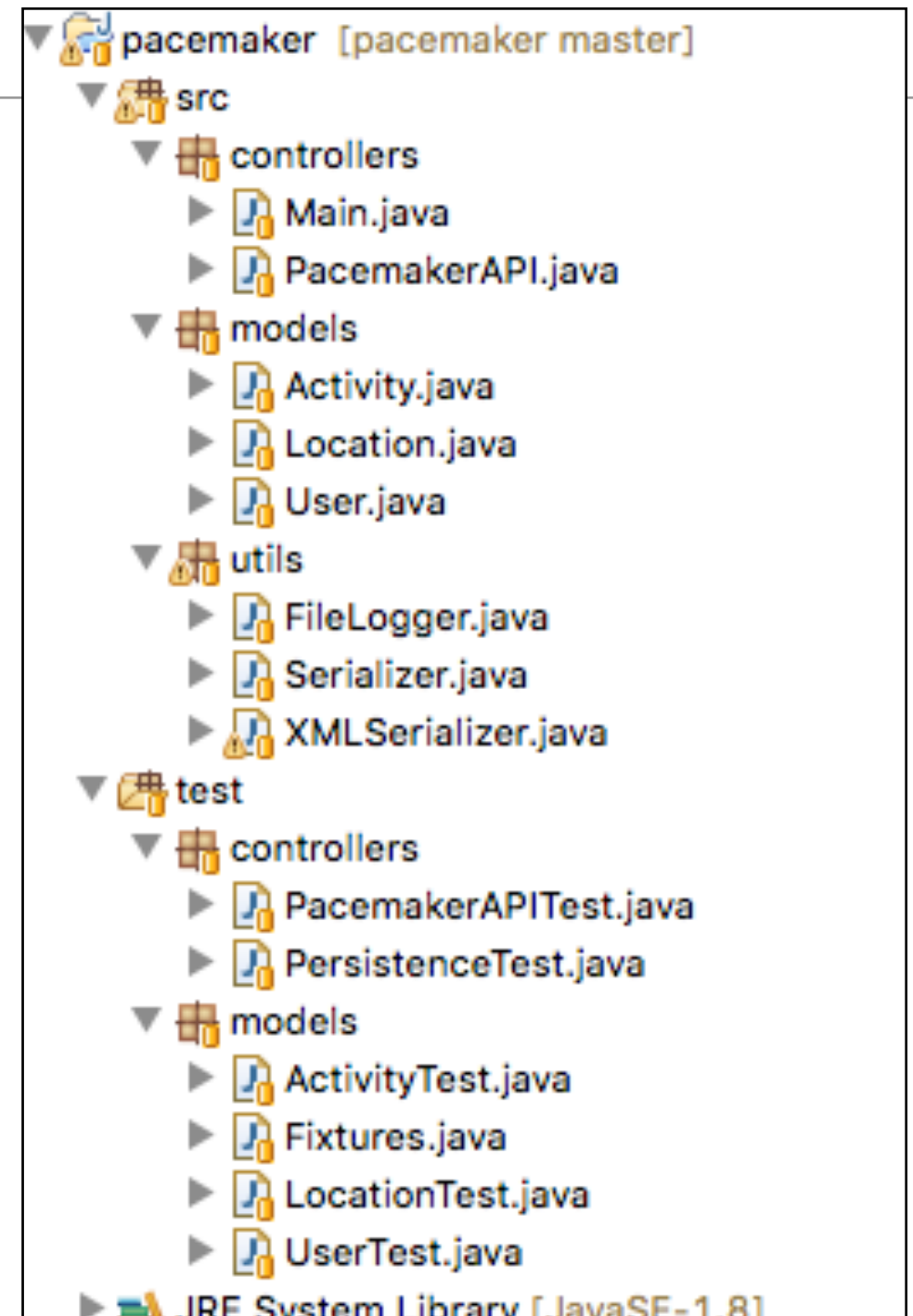
    @Command // One,
    public String hello() {
        return "Hello, World!";
    }

    @Command // two,
    public int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) throws IOException {
        ShellFactory.createConsoleShell("hello", "", new HelloWorld())
            .commandLoop(); // and three.
    }
}
```

Testing

- Create a separate top level 'source folder' for all tests
- Mirror the 'src' package structure in this folder
- Create one test class for each 'src' class



Persistence

- Use a suitable high level library
- Single file to store entire object model
- Alternatives?



About XStream

XStream is a simple library to serialize objects to XML and back again.

Features #features

- **Ease of use.** A high level facade is supplied that simplifies common use cases.
- **No mappings required.** Most objects can be serialized without need for specifying mappings.
- **Performance.** Speed and low memory footprint are a crucial part of the design, making it suitable for large object graphs or systems with high message throughput.
- **Clean XML.** No information is duplicated that can be obtained via reflection. This results in XML that is easier to read for humans and more compact than native Java serialization.
- **Requires no modifications to objects.** Serializes internal fields, including private and final. Supports non-public and inner classes. Classes are not required to have default constructor.
- **Full object graph support.** Duplicate references encountered in the object-model will be maintained. Supports circular references.
- **Integrates with other XML APIs.** By implementing an interface, XStream can serialize directly to/from any tree structure (not just XML).
- **Customizable conversion strategies.** Strategies can be registered allowing customization of how particular types are represented as XML.
- **Security framework.** Fine-control about the unmarshalled types to prevent security issues with manipulated input.
- **Error messages.** When an exception occurs due to malformed XML, detailed diagnostics are provided to help isolate and fix the problem.
- **Alternative output format.** The modular design allows other output formats. XStream ships currently with JSON support and morphing.



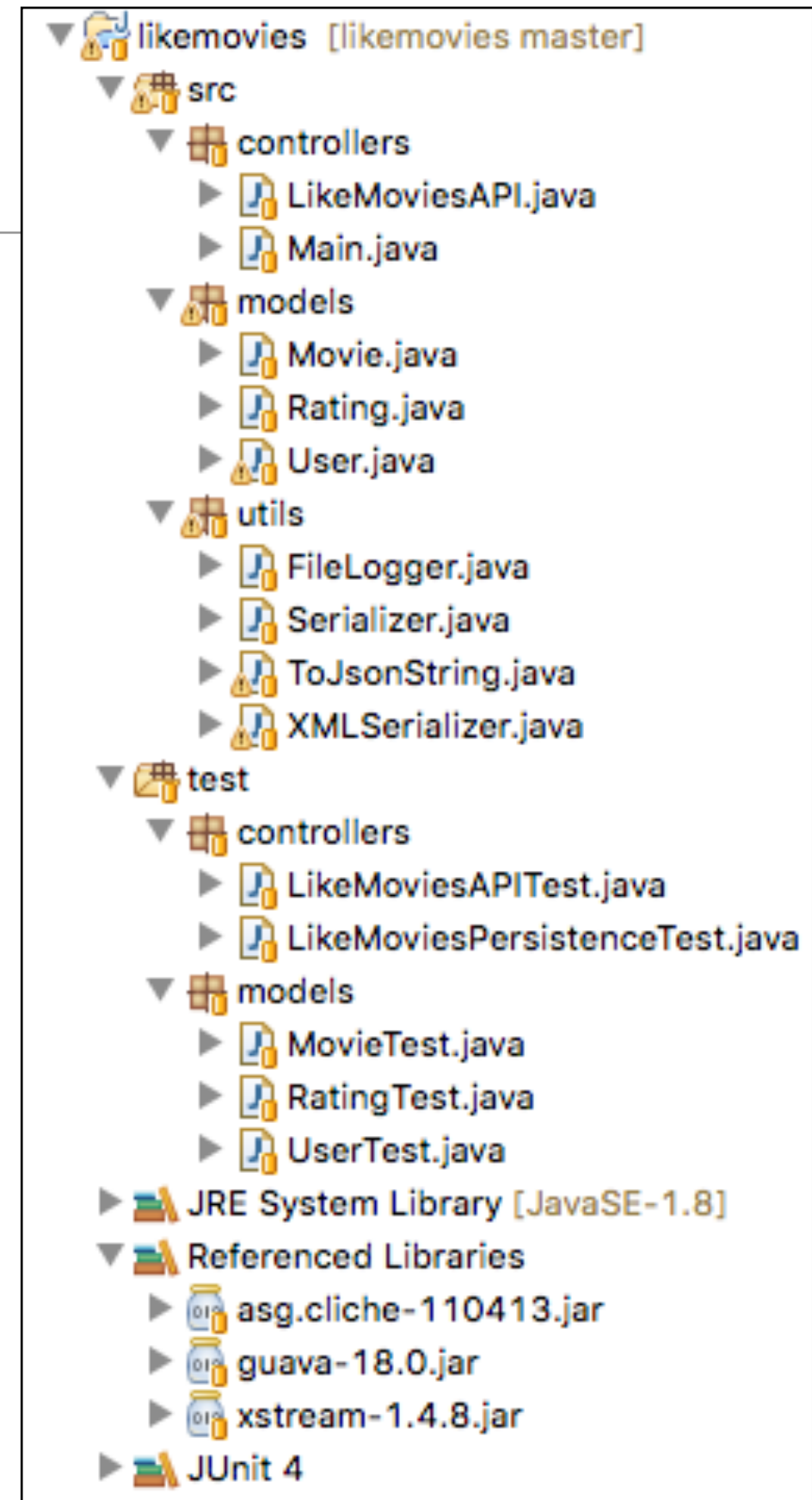
where do I start?

Plan for Assignment 2

- Build Walking Skeleton
- Break features into simple 'Stories'
- Reorder the stories into simplest to implement first
- For each story:
 - Write a test
 - Implement the necessary features
 - Implement and verify the command

Build Walking Skeleton

- Define very simple model objects : User & Movie
- Implement tests for these
- Implement Simple command for add and list all users
- Create skeleton version classes you think you will need (even if they are empty)

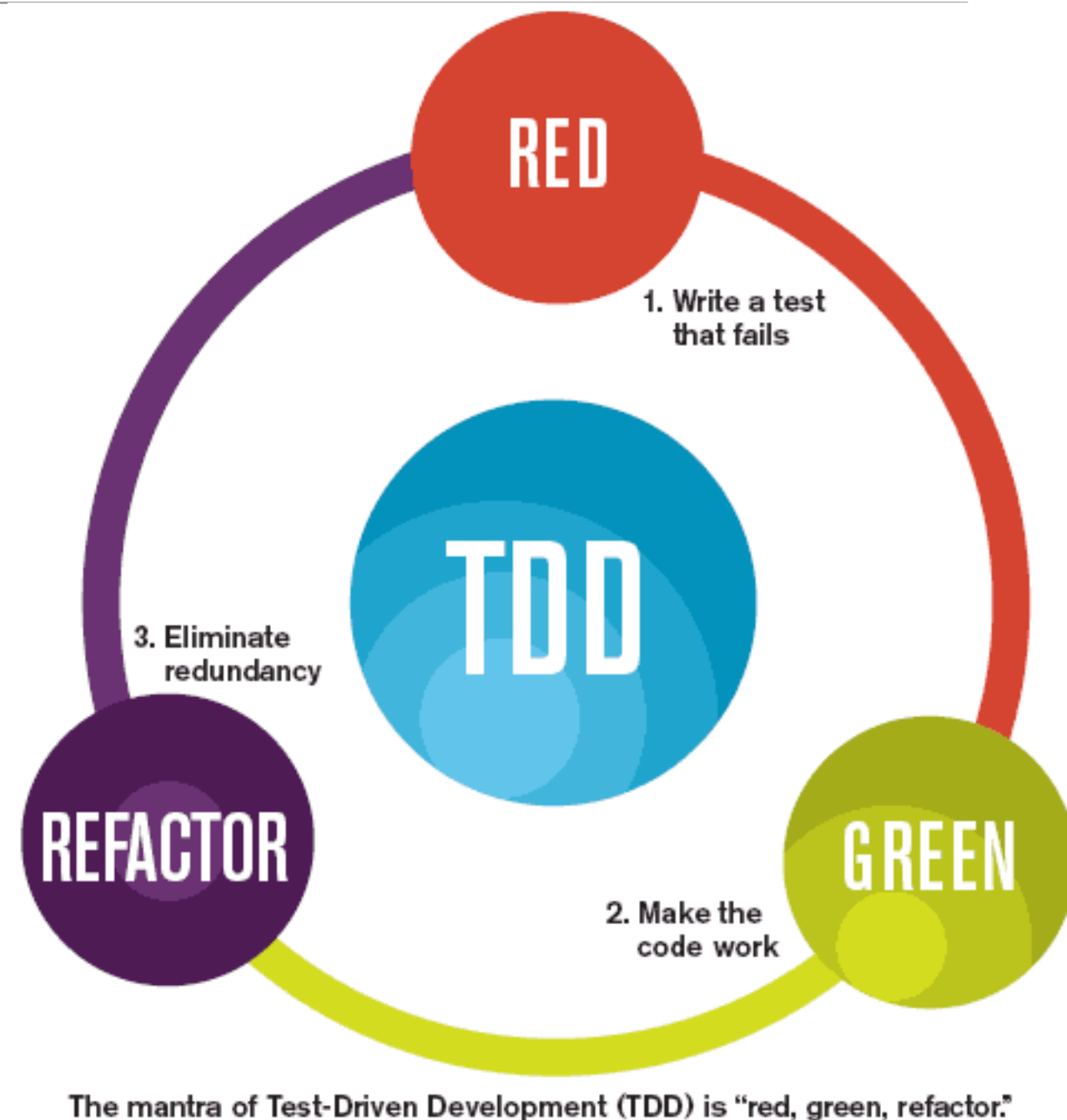


Stories

- add a user
- add a movie
- remove a user
- get a movies details
- rate a movie
- get a users ratings
- get the top ten movies (by all ratings)
- for a given user, get recommendations for that user (recommendation algorithm)
- read the movie db from an external css file
- save / load the main application model

Organise Stories

- Projects are usually implemented in 'Iterations'
- Each Iteration starts with a subset of stories the iteration will tackle
- For each story in the iteration:
 - Write a test
 - Implement sufficient features for the test to pass
 - Refactor to improve the implementation

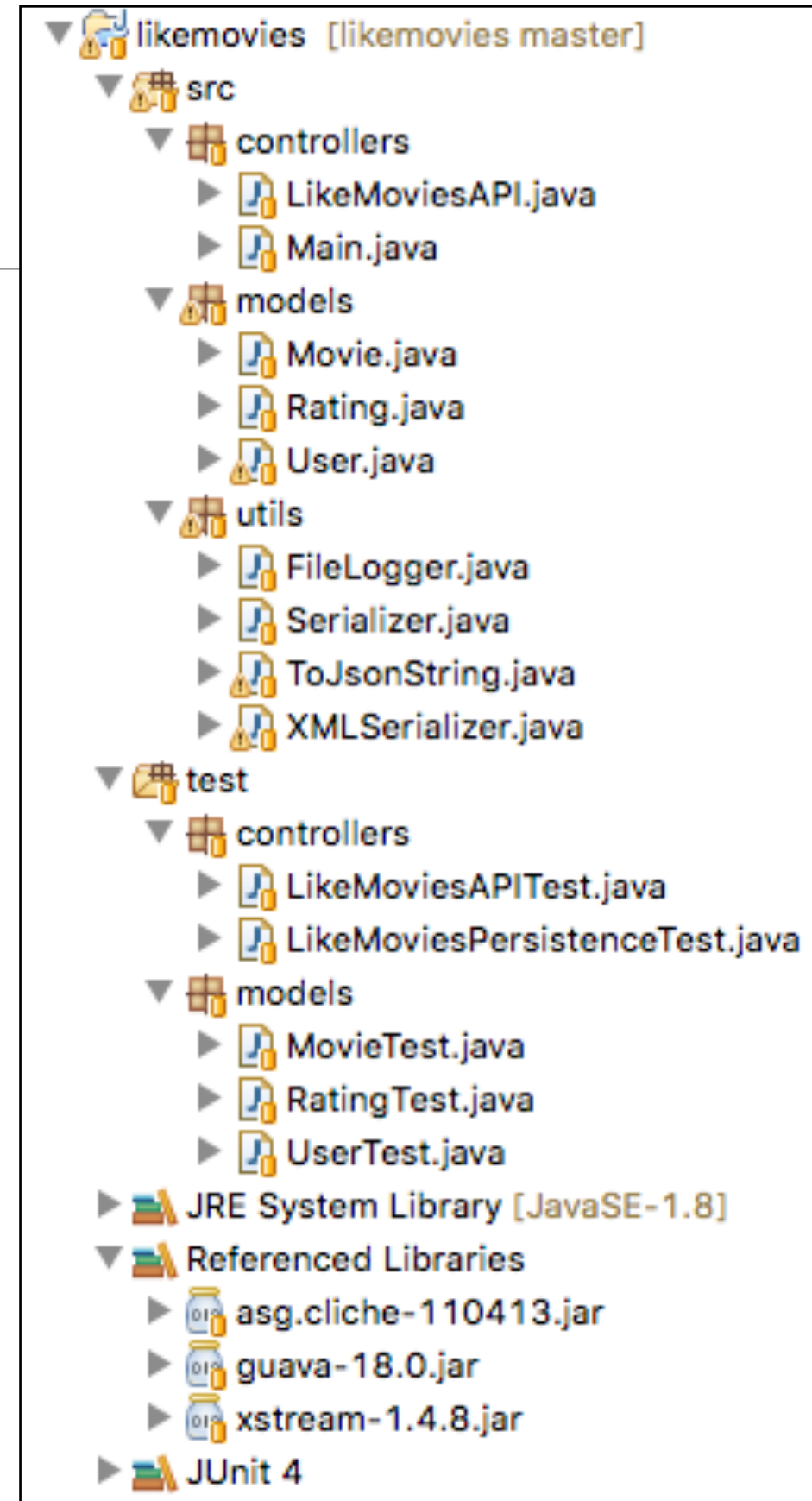


Iteration Plan (suggestion)

- Iteration I
 - add a user
 - add a movie
 - remove a user
 - get a movies details
- Iteration II
 - rate a movie
 - get a users ratings
- Iteration III
 - save / load the main application model (to XML or JSON)
 - read the initial movie db from an external csv file
- Iteration IV
 - get the top ten movies (by all ratings)
 - for a given user, get recommendations for that user (recommendation algorithm)

Walking Skeleton - Extracts

- This skeleton closely modelled on pacemaker
- the 'utils' package can be carried over as is
- API and Main mirror the pacemaker organisation:
 - API - no UI, just manage the data
 - Main - deal with all UI via cliché



Walking Skeleton - Model

- User

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

    public Long    id;
    public String  firstName;
    public String  lastName;
    public String  gender;
    public String  age;
    public String  occupation;

    public List<Rating> ratings = new ArrayList<>();

    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;
    }

    public String toString()
    {
        return new ToStringBuilder(this).append("id", id).append("firstName", firstName).append("lastName", lastName).append("gender", gender).append("age", age).append("occupation", occupation).append("ratings", ratings).toString();
    }

    @Override
    public int hashCode()
    {
        return Objects.hash(lastName, firstName, gender, age);
    }

    @Override
    public boolean equals(final Object obj)
    {
        if (obj instanceof User)
        {
            final User other = (User) obj;
            return Objects.equal(firstName, other.firstName)
                && Objects.equal(lastName, other.lastName)
                && Objects.equal(gender, other.gender)
                && Objects.equal(age, other.age)
                && Objects.equal(occupation, other.occupation)
                && Objects.equal(ratings, other.ratings);
        }
        else
        {
            return false;
        }
    }
}
```

Walking Skeleton - Model

- Movie

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

    public Long    id;

    public String  title;
    public String  year;
    public String  url;

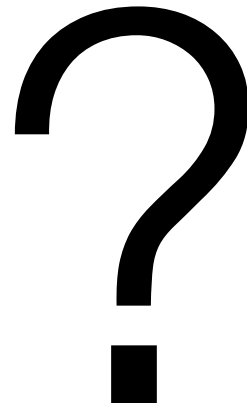
    public Movie(String title, String year, String url)
    {
        this.id      = counter++;
        this.title    = title;
        this.year     = year;
        this.url      = url;
    }

    @Override
    public String toString()
    {
        return new ToString(getClass(), this).toString();
    }

    @Override
    public int hashCode()
    {
        return Objects.hashCode(this.id, this.title, this.year, this.url);
    }

    @Override
    public boolean equals(final Object obj)
    {
        if (obj instanceof Movie)
        {
            final Movie other = (Movie) obj;
            return Objects.equal(title, other.title)
                && Objects.equal(year, other.year)
                && Objects.equal(url, other.url);
        }
        else
        {
            return false;
        }
    }
}
```

Walking Skeleton - User & Movie Tests



(see pacemaker)

Walking Skeleton - LikeMoviesAPI

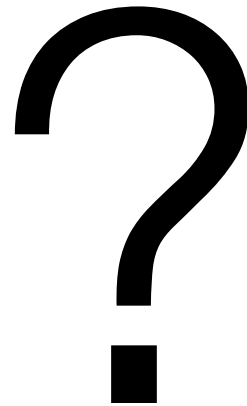
```
public class LikeMoviesAPI
{
    public Map<Long, User>    userIndex    = new HashMap<>();
    public Map<Long, Movie>  movieIndex   = new HashMap<>();

    public LikeMoviesAPI()
    {}

    public User addUser(String firstName, String lastName, String age, String gender, String occupation)
    {
        User user = new User (firstName, lastName, age, gender, occupation);
        userIndex.put(user.id, user);
        return user;
    }

    public Movie addMovie(String title, String year, String url)
    {
        Movie movie = new Movie (title, year, url);
        movieIndex.put(movie.id, movie);
        return movie;
    }
}
```

Walking Skeleton - LikeMoviesAPITest



(see pacemaker)

Walking Skeleton Main (command line)

```
public class Main
{
    public LikeMoviesAPI likeMovies;

    @Command(description="Add a new User")
    public void addUser (@Param(name="first name") String firstName, @Param(name="last name") String lastName,
        @Param(name="age") String age, @Param(name="gender") String gender, @Param(name="occupation") String occupation)
    {
        likeMovies.addUser(firstName, lastName, age, gender, occupation);
    }

    @Command(description="Delete a User")
    public void removeUser (@Param(name="id") Long id)
    {
        likeMovies.removeUser(id);
    }

    @Command(description="Add a Movie")
    public void addMovie (@Param(name="title") String title, @Param(name="year") String year, @Param(name="url") String url)
    {
        likeMovies.addMovie(title, year, url);
    }

    public static void main(String[] args) throws Exception
    {
        Main main = new Main();

        Shell shell = ShellFactory.createConsoleShell("lm", "Welcome to likemovie - ?help for instructions", main);
        shell.commandLoop();

        main.likeMovies.store();
    }
}
```


Command line test -

- Run Script?
- This command lets you run a 'batch' of commands from a file.
- You could have a file with many commands like this
- Have !rs run the file to load the application with test data

```
Welcome to likemovie - ?help for instructions
lm> ?help
This is Cliche shell (http://cliche.sourceforge.net).
To list all available commands enter ?list or ?list-all, the
latter will also show you system commands. To get detailed
info on a command enter ?help command-name
lm> ?list-all
abbrev name    params
!rs !run-script(filename)
!el !enable-logging (fileName)
!dl !disable-logging ()
!gle !get-last-exception ()
!sdt !set-display-time (do-display-time)
?l ?list ()
?l ?list (startsWith)
?h ?help ()
?h ?help (command-name)
?la ?list-all ()
?ghh ?generate-HTML-help (file-name, include-prefixed)
am add-movie (title, year, url)
au add-user (first name, last name, age, gender,
occupation)
lm> add-user homer simpson 45 male genius
lm> add-movie thesimpsons 2005 www.thesimpsons.com
lm> exit
```

Iteration Plan (suggestion)

- Iteration I
 - add a user
 - add a movie
 - remove a user
 - get a movies details
- Iteration II
 - rate a movie
 - get a users ratings
- Iteration III
 - save / load the main application model (to XML or JSON)
 - read the initial movie db from an external csv file
- Iteration IV
 - get the top ten movies (by all ratings)
 - for a given user, get recommendations for that user (recommendation algorithm)

Learn from Pacemaker Repo History



wit-computing / **algorithms-2015-pacemaker**

Unwatch ▾

1

Description

Short description of this repository

Website

Website for this repository (optional)

Save or [Cancel](#)

28 commits

1 branch

1 release

1 contributor



Branch: **master** ▾

algorithms-2015-pacemaker / +



edeleastar Jason toString methods utility class
















Latest commit 55d5273 7 days ago

lib	cliche library + initial default command line	8 days ago
src	Jason toString methods utility class	7 days ago
test	get-users command + datastore.xml sample data from test	8 days ago
.classpath	cliche library + initial default command line	8 days ago
.gitignore	blank project generated by eclipse	2 months ago
.project	blank project generated by eclipse	2 months ago
datastore.xml	get-users command + datastore.xml sample data from test	8 days ago













Help people interested in this repository understand your project by adding a README.

Add a README

Commits on Oct 2, 2015

	Introduce IDs inyo User and API classes edeleastar committed on Oct 2		8365df6	
	User and Pacemaker improvements- uisng maps and guava hashCode edeleastar committed on Oct 2		797c5a0	
	Users toString using Guava library edeleastar committed on Oct 2		6bd23c6	
	introduced PacemakerAPI + simplified Main edeleastar committed on Oct 2		2bffd78	
	guava added edeleastar committed on Oct 2		18ad828	

Commits on Sep 25, 2015

	Xstream introduced to serialize users edeleastar committed on Sep 25		f6c3e3e	
	Logger introduced edeleastar committed on Sep 25		93c835b	
	initial version of User and Main edeleastar committed on Sep 25		d868af2	
	blank project generated by eclipse edeleastar committed on Sep 25		d13d91c	

Commits on Oct 8, 2015



More comprenhenaive createActivity tests

edeleastar committed on Oct 8



8144f1b



Equals methods introduced + extended API tests

edeleastar committed on Oct 8



680ed2a



Initial PacemakerAPI test

edeleastar committed on Oct 8



8617dc7



Activity Test + fixtures for all tests

edeleastar committed on Oct 8



b82b366



Location test reqorded to use fixtures

edeleastar committed on Oct 8



0bf5773



Location test

edeleastar committed on Oct 8



95ffc70



Introduced Activity & Location + revised PacemakerAPI

edeleastar committed on Oct 8



c1d226f



Commits on Oct 5, 2015



First tests + fixture introduced

edeleastar committed on Oct 5



f004d17



Commits on Nov 1, 2015



refactored package structure to align test and app packages

edeleastar committed 19 days ago



6f9cd13



full persistence tests

edeleastar committed 19 days ago



c28a20f



preparing for persistence test - populate method

edeleastar committed 19 days ago



7a0c4e4



Commits on Oct 16, 2015



General Serializer introduced

edeleastar committed on Oct 16



9bf51e6



simple xml serialization introduced to pacemakerAPI










edeleastar committed on Oct 16












90825f4



Commits on Nov 13, 2015

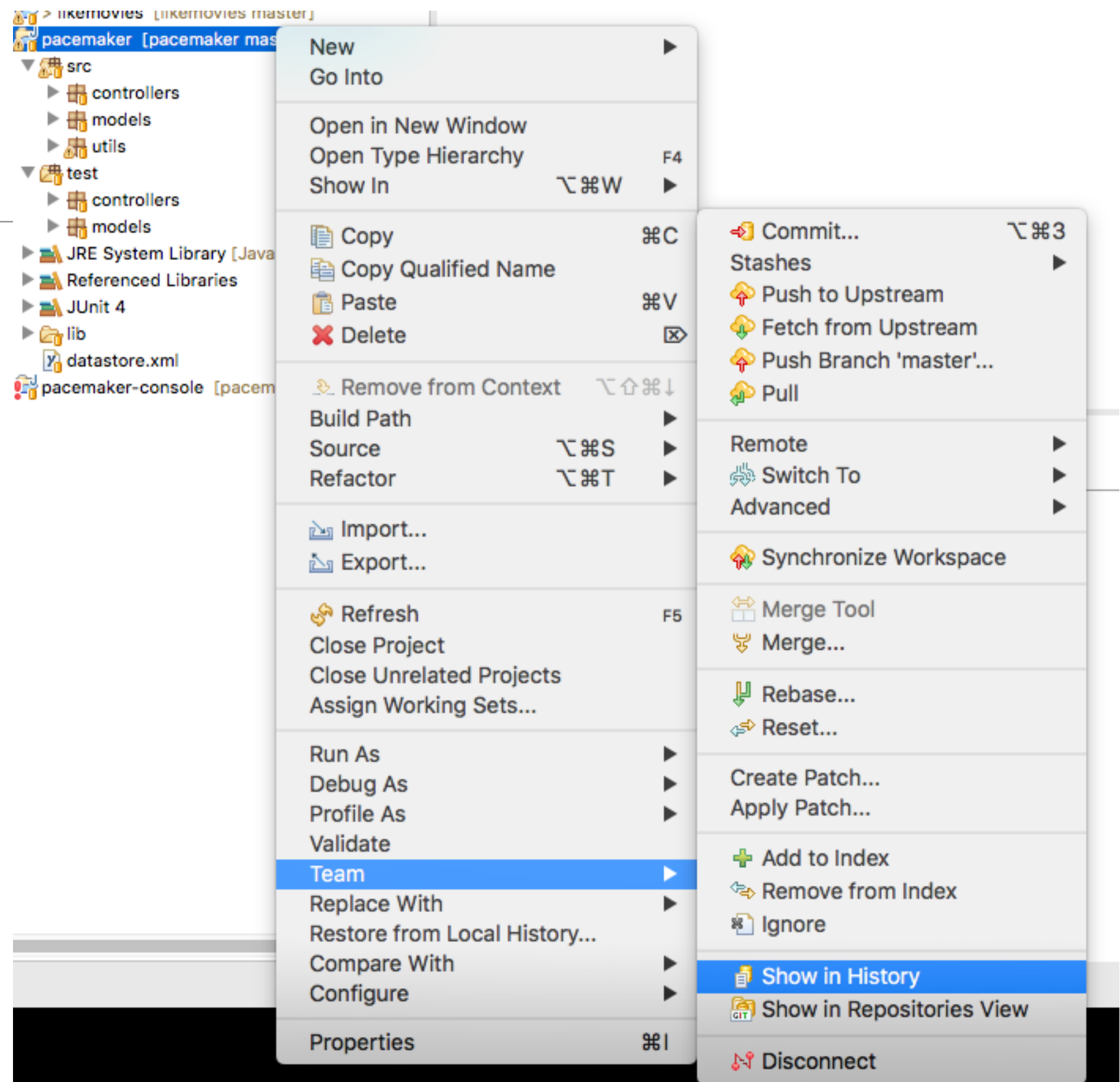
	Jason toString methods utility class edeleastar committed 7 days ago		55d5273	
	include activities in user.toString edeleastar committed 7 days ago		d40082c	
	add user + location commands edeleastar committed 7 days ago		e6e6b25	

Commits on Nov 12, 2015

	user commands edeleastar committed 8 days ago		d8470ea	
	get-users command + datastore.xml sample data from test edeleastar committed 8 days ago		16a3752	
	cliche library + initial default command line edeleastar committed 8 days ago		17b0d72	

Commits on Nov 1, 2015

Reset - in
Eclipse -
to any
specific
revision



Java - Eclipse - /Users/edeleastar/repos/modules/algorithms/ws

Quick Access

Pac Navi JU

- algorithms_assign1_2015 [algorithmn]
- lab01 [lab01 master]
- lab02 [lab02 master]
- lab03 [lab03 master]
- > likemovies [likemovies master]
- pacemaker [pacemaker master]
 - src
 - controllers
 - models
 - utils
 - test
 - controllers
 - models
 - JRE System Library [JavaSE-1.8]
 - Referenced Libraries
 - JUnit 4
 - lib
 - datastore.xml
- pacemaker-console [pacemaker-coi]

Problems Javadoc Declaration Console History

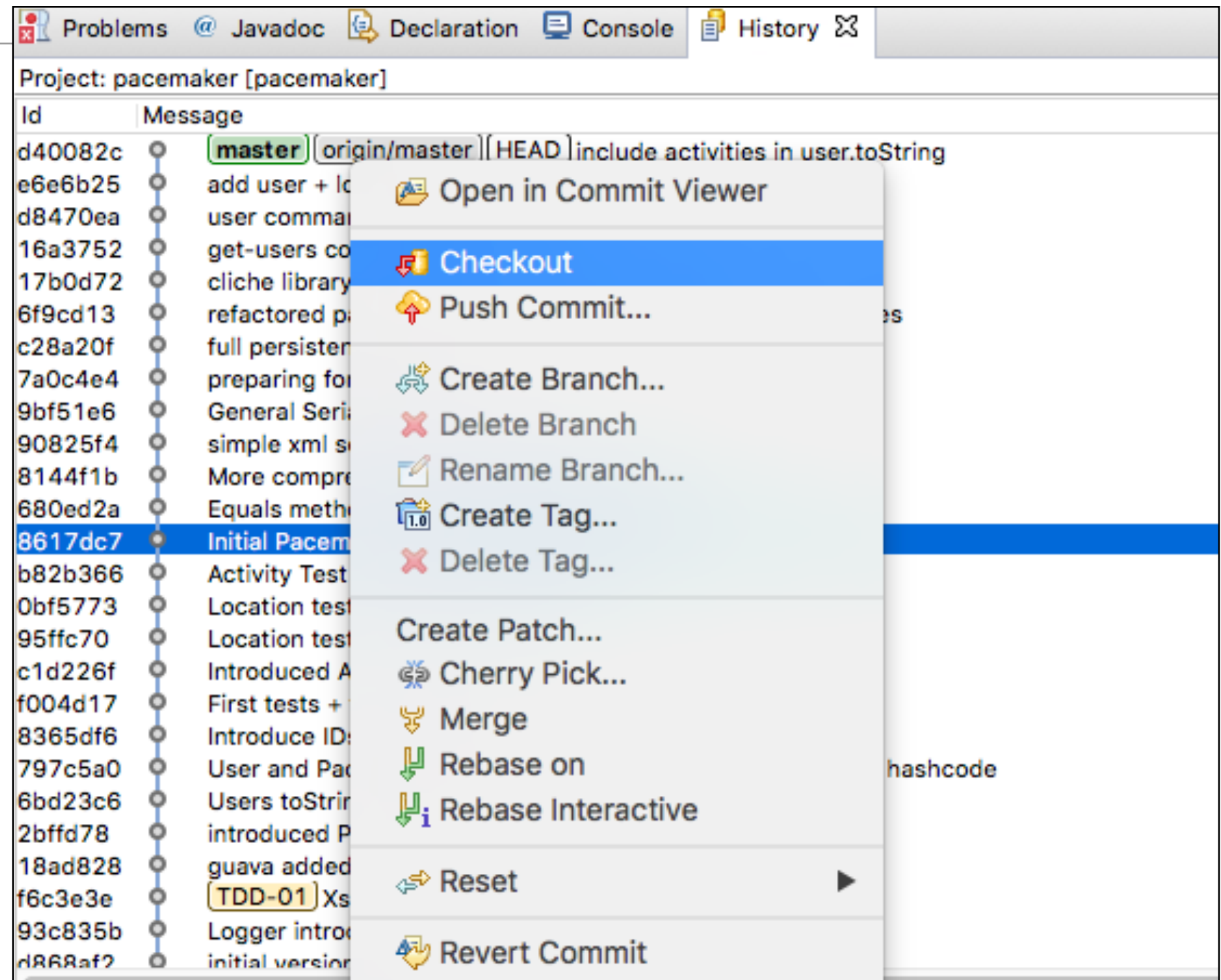
Project: pacemaker [pacemaker]

Id	Message	Author	Authored Date	Co
d40082c	[master] origin/master HEAD include activities in user.toString	Eamonn de Leastar	7 days ago	Ean
e6e6b25	add user + location commands	Eamonn de Leastar	7 days ago	Ean
d8470ea	user commands	Eamonn de Leastar	8 days ago	Ean
16a3752	get-users command + datastore.xml sample data from test	Eamonn de Leastar	8 days ago	Ean
17b0d72	cliche library + initial default command line	Eamonn de Leastar	8 days ago	Ean
6f9cd13	refactored package structure to alight test and app packages	Eamonn de Leastar	3 weeks ago	Ean
c28a20f	full persistence tests	Eamonn de Leastar	3 weeks ago	Ean
7a0c4e4	preparing for persistence test - populate method	Eamonn de Leastar	3 weeks ago	Ean
9bf51e6	General Serializer introduced	Eamonn de Leastar	5 weeks ago	Ean
90825f4	simple xml serialization introduced to pacemakerAPI	Eamonn de Leastar	5 weeks ago	Ean
8144f1b	More comprenhenaive createActivity tests	Eamonn de Leastar	6 weeks ago	Ean
680ed2a	Equals methods introduced + extended API tests	Eamonn de Leastar	6 weeks ago	Ean
8617dc7	Initial PacemakerAPI test	Eamonn de Leastar	6 weeks ago	Ean
b82b366	Activity Test + fixtures for all tests	Eamonn de Leastar	6 weeks ago	Ean
Obf5773	Location test reqorded to use fixtures	Eamonn de Leastar	6 weeks ago	Ean
95ffc70	Location test	Eamonn de Leastar	6 weeks ago	Ean
c1d226f	Introduced Activity & Location + revised PacemakerAPI	Eamonn de Leastar	6 weeks ago	Ean
f004d17	First tests + fixture introduced	Eamonn de Leastar	7 weeks ago	Ean
8365df6	Introduce IDs inyo User and API classes	Eamonn de Leastar	7 weeks ago	Ean
797c5a0	User and Pacemaker improvements- uisng maps and guava hashcode	Eamonn de Leastar	7 weeks ago	Ean
6bd23c6	Users toString using Guava library	Eamonn de Leastar	7 weeks ago	Ean
2bff78	introduced PacemakerAPI + simplified Main	Eamonn de Leastar	7 weeks ago	Ean
18ad828	guava added	Eamonn de Leastar	7 weeks ago	Ean
f6c3e3e	[TDD-01] Xstream introduced to serialize users	Eamonn de Leastar	8 weeks ago	Ean
93c835b	Logger introduced	Eamonn de Leastar	8 weeks ago	Ean
d868af2	initial version of User and Main	Eamonn de Leastar	8 weeks ago	Ean

commit d40082c2f73f24fc93b0598fe1640f5c222c6a51
Author: Eamonn de Leastar <edeleastar@wit.ie> 2015-11-13 10:53:28

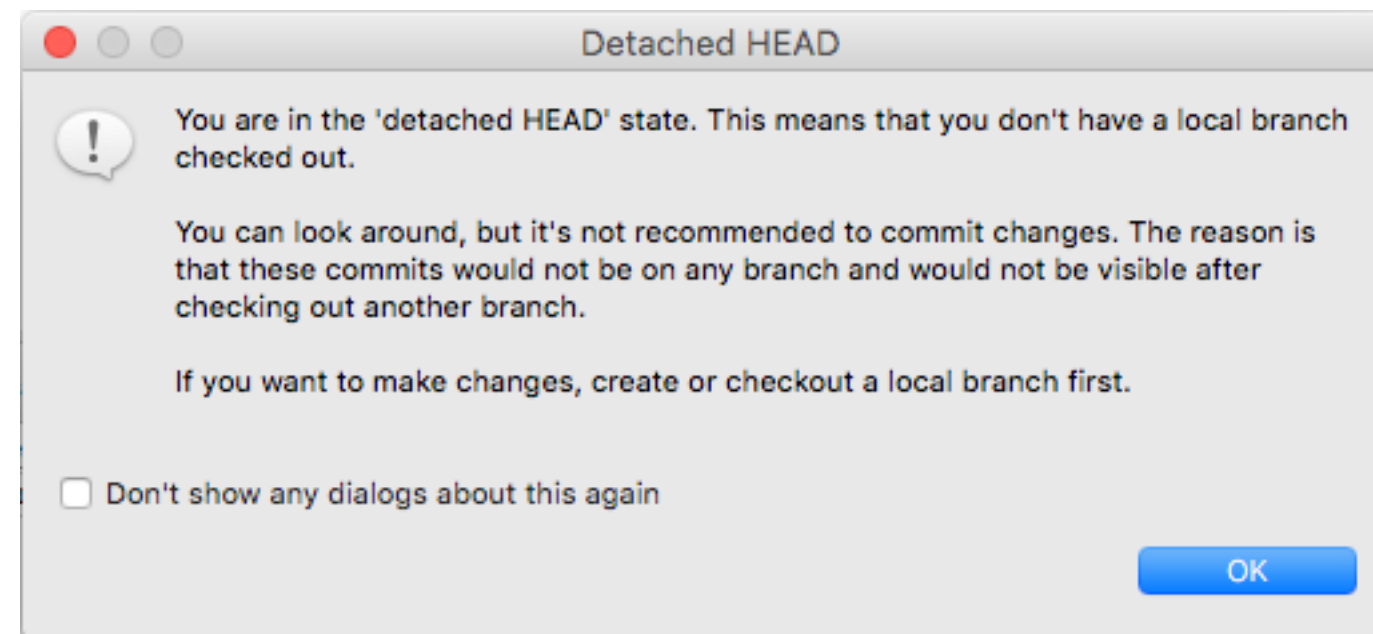
src/models/User.java

- Select a revision - and 'Checkout'
- Project will reset to that particular version



HEAD

- During checkout - HEAD will be DETACHED
- This means you cant make changes at that point, just browse and run the program.



- Explore exactly how each feature was introduced via github site

29 ■■■■■ src/controllers/Main.java View

		@@ -3,11 +3,15 @@
3	3	import java.io.File;
4	4	import java.util.Collection;
5	5	
	6	+import com.google.common.base.Optional;
	7	+
6	8	import utils.Serializer;
7	9	import utils.XMLSerializer;
8	10	import asg.cliche.Command;
	11	+import asg.cliche.Param;
9	12	import asg.cliche.Shell;
10	13	import asg.cliche.ShellFactory;
	14	+import models.Activity;
11	15	import models.User;
12	16	
13	17	
		@@ -34,6 +38,31 @@ public void getUsers ()
34	38	System.out.println(users);
35	39	}
36	40	
	41	+ @Command(description="Create a new User")
	42	+ public void createUser (@Param(name="first name") String firstName, @Param(name="last name") String lastName,
	43	+ @Param(name="email") String email, @Param(name="password") String password)
	44	+ {
	45	+ paceApi.createUser(firstName, lastName, email, password);
	46	+ }
	47	+
	48	+ @Command(description="Get a Users detail")
	49	+ public void getUser (@Param(name="email") String email)
	50	+ {
	51	+ User user = paceApi.getUserByEmail(email);
	52	+ System.out.println(user);
	53	+ }
	54	+
	55	+ @Command(description="Delete a User")
	56	+ public void deleteUser (@Param(name="email") String email)
	57	+ {
	58	+ Optional<User> user = Optional.fromNullable(paceApi.getUserByEmail(email));
	59	+ if (user.isPresent())
	60	+ {
	61	+ paceApi.deleteUser(user.get().id);
	62	+ }
	63	+ }
	64	+
	65	+
37	66	public static void main(String[] args) throws Exception
38	67	{

Alternatively - use Sourcetree

The screenshot displays the Sourcetree application interface for a project named "pacemaker (Git)". The top toolbar includes icons for View, Commit, Checkout, Reset, Stash, Add, Remove, Add/Remove, Fetch, Pull, Push, Branch, Merge, Tag, Show in Finder, Git Flow, Terminal, and Settings.

FILE STATUS

- Working Copy

BRANCHES

- HEAD
- master

TAGS

REMOTES

STASHES

SUBMODULES

SUBTREES

Commit History Table:

Graph	Description	Commit	Author	Date
	General Serializer introduced	9bf51e6	Eamonn de Leasta...	16 Oct 2015, 9:4...
	simple xml serialization introduced to pacemakerAPI	90825f4	Eamonn de Leasta...	16 Oct 2015, 9:1...
	More comprehensive createActivity tests	8144f1b	Eamonn de Leasta...	8 Oct 2015, 4:59...
	Equals methods introduced + extended API tests	680ed2a	Eamonn de Leasta...	8 Oct 2015, 4:54...
	Initial PacemakerAPI test	8617dc7	Eamonn de Leasta...	8 Oct 2015, 4:47...
	Activity Test + fixtures for all tests	b82b366	Eamonn de Leasta...	8 Oct 2015, 4:44...
	Location test required to use fixtures	0bf5773	Eamonn de Leasta...	8 Oct 2015, 4:38...
HEAD	Location test	95ffc70	Eamonn de Leasta...	8 Oct 2015, 4:3...
	Introduced Activity & Location + revised PacemakerAPI	c1d226f	Eamonn de Leasta...	8 Oct 2015, 4:27...
	First tests + fixture introduced	f004d17	Eamonn de Leasta...	5 Oct 2015, 9:28...
	Introduce IDs into User and API classes	8365df6	Eamonn de Leasta...	2 Oct 2015, 7:52...
	User and Pacemaker improvements- using maps and guava hashCode	797c5a0	Eamonn de Leasta...	2 Oct 2015, 7:45...
	Users toString using Guava library	6bd23c6	Eamonn de Leasta...	2 Oct 2015, 7:18...
	introduced PacemakerAPI + simplified Main	2bffd78	Eamonn de Leasta...	2 Oct 2015, 7:08...
	guava added	18ad828	Eamonn de Leasta...	2 Oct 2015, 7:00...
TDD-01	Xstream introduced to serialize users	f6c3e3e	Eamonn de Leasta...	25 Sep 2015, 11...

File Explorer:

- src
 - models
 - User.java

Diff View (src/models/User.java):

Hunk 1: Lines 1-5

```
1 1 package models;
2 2 + import static com.google.common.base.MoreObjects.toStringHelper;
3 3
4 4 public class User
5 5 {
```

Hunk 2: Lines 19-31

```
18 19 this.email = email;
19 20 this.password = password;
20 21 }
22 22 +
23 23 + public String toString()
24 24 + {
25 25 + return toStringHelper(this).addValue(firstName)
26 26 + .addValue(lastName)
27 27 + .addValue(password)
28 28 + .addValue(email)
29 29 + .toString();
30 30 + }
31 31 }
```

No newline at end of file

Commit Information:

Commit: 6bd23c66ccc0f17a86fafbaf9b4b1fecf8c647bd [6bd23c6]

Parents: 2bffd78df1

Author: Eamonn de Leasta <edeleasta@wit.ie>

Date: 2 October 2015 at 7:18:28 a.m. IST

Commit Message: Users toString using Guava library

Status Bar: (detached from 95ffc70) Clean Fetching

Atlassian