AGILE SOFTWARE DEVELOPMENT LABORATORY

(CSX-358)

PRACTICALS RECORD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)
2017

Submitted To:	*Submitted By:*
Mr. Paramvir Singh	*saiganesh munduru*
Assistant Professor	*14103021*
Dept. of CSE	*cse department*

LAB- 1

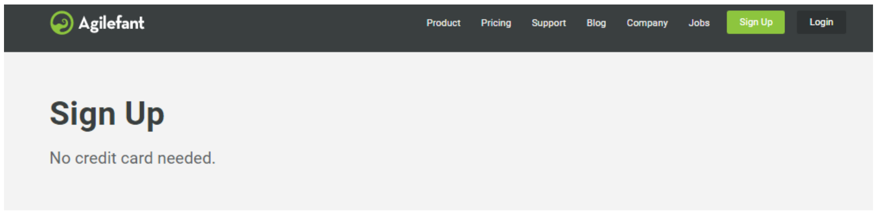
Aim: - *To learn various features of Agilefant using the website https://www.agilefant.com/ as a Scrum Master*.

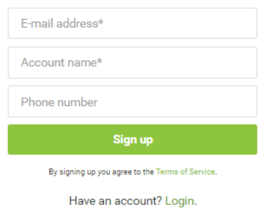
Introduction: -

Agilefant is a versatile software tool for tracking and planning that enables organizations to unleash the full potential of their employees, partners and existing tools.

- ✓ **Sign up:**

Signing up for a brand new Agilefant account takes less than 30 seconds. Account has a 30 day free trial, with unlimited users and features. After this, it will automatically switch to the solo plan unless you subscribe to one of the available plans.





- ✓ *Fill out the personal information*

After landing in Agilefant, you will be asked to update your full name, display name used in Agilefant, and set your password. When this is done, one can also invite co-workers. All the added users will receive an email containing instructions on how to log in. After finishing the previous steps, please use a minute to go through the introduction tour. It explains the main views of Agilefant. Now the initial setup is done. You can either take a look at different views or just start building your first backlog.

- ✓ *Create a backlog*

There are three kinds of backlogs in Agilefant: Products, Projects, and Iterations. The product backlog contains everything that may be needed to be done. Out of this, you split and plan a set of stories to be done in a project, and move the stories into the project..

- ✓ *Click on the top left 'Create new' button and hit 'Product'.*
- ✓ *Set a name for the product.*
- ✓ *Define which teams have access to that product; you can change these later on.*
- ✓ *Click on 'Create and open' and the product will open for you.*

1. Create a Products

Product is the generic term used for something the organization is developing such as a piece of software or service. In Agilefant you can have as many products as you want.
Some organizations using Agilefant use products to represent specific customers – or even entire business areas. To create new products:

- ✓ *Click the 'Create new' button in the top left corner of the page.*
- ✓ *When creating a product, define at least one team that has access to that product. Otherwise, only admin users can see and access the created product.*
- ✓ *It can be changed later from Administration -> Access rights.*

2. Create a Projects

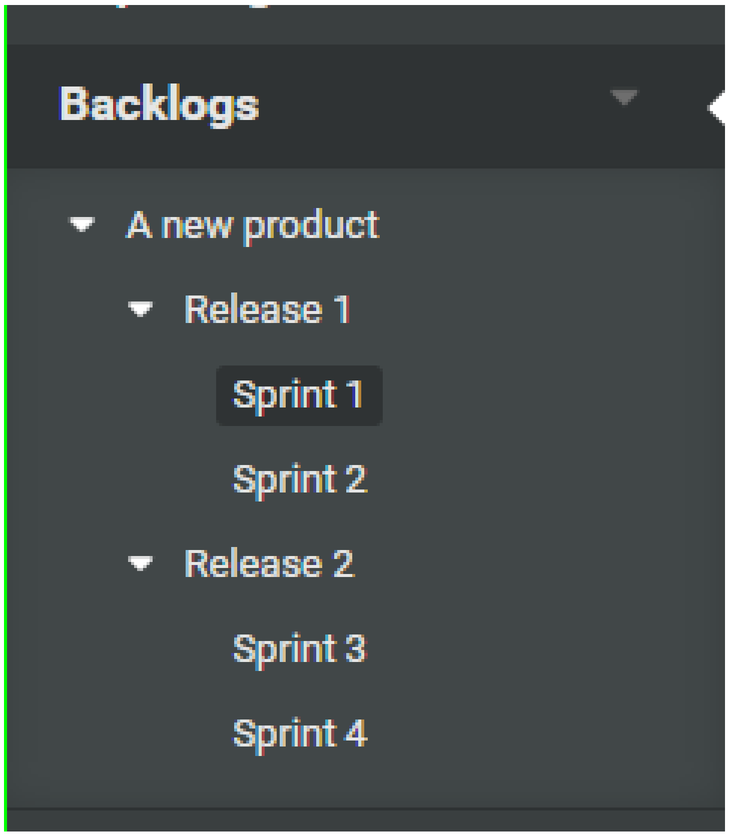
Products are developed in projects. One can think of projects as 'major releases' or important milestones. As projects have a start and an end date, think about a scope which is important to you to track regarding progress versus a deadline, and make that as your project.
Projects can be created from the Create new button. Projects are always related to a single product, so one cannot create projects before products (making this more flexible is in the roadmap). However, there can be parallel projects under a single product. For example, it's also possible to handle different project by each team.

3. Create an Iteration

Iteration is a timeboxed period of time during of which development (or any activity) takes place. In Scrum, iterations are called "sprints". Stories are presented as a list instead of a tree and iteration may only have stories which have no child stories.
In this view, you can manage stories and tasks that are going to be implemented during the iteration. Additionally, you can follow the progress of stories and tasks, and estimate whether all the planned work will get done during the iteration. To create iteration:

- ✓ *To create iteration, click Create new on the top left corner and then Iteration.*
- ✓ *Set a name for the iteration.*
- ✓ *Select a project to where the iteration is created.*
- ✓ *Set the iteration's start and end dates.*
- ✓ *Click 'Create and open' and the iteration is created.*

You can also create iterations which don't belong to any particular project. These are called standalone iterations. Look up the user guide for more information.



4. *Move stories to an iteration*

You can move stories to iteration similarly as into a project. Just drag and drop them to the iteration, located in the left hand side backlog menu.

- ✓ *Create stories in an iteration*
- ✓ *You can create stories by clicking the green '+' button and then*
- ✓ *Set a name for the story.*
- ✓ *Press Create and the story will appear on the top of the list.*

5. *Estimating stories and tasks*

You can estimate stories using story points. If you want to estimate effort left in man-hours, you need tasks within the story. To create tasks into the story, follow the following steps:

- ✓ *Click the arrow icon on the left side of the story to open story's details.*
- ✓ *Scroll down a bit and click the Add task button.*
- ✓ *Set a name for the task.*
- ✓ *Press Save and the task will appear in the list.*

6. States

Stories (and tasks) have pre-defined states. You can think about their meaning as follows: The following states have 'special effects':

- ***Done** – the final state of a task/story after it's been completed. Affects the related metrics.*
- ***Deferred** – the task/story has been decided to be skipped in this project/iteration; the effort left / points are omitted in all metrics. This can be used to quickly scope out stories / tasks without having to move them to a different backlog.*

The other states are:

- *Not Started – No work has yet been put into realizing this story*
- *In Progress – ongoing and some work has already been put in*
- *Pending – waiting for something external that can reasonably be expected to happen without us taking any further action*
- *Blocked – can't proceed; most likely some action must be taken by 'us' before work can proceed*
- *Ready – otherwise done, but some relatively minor definition-of-done criteria are yet to be met; e.g. the story must be demoed to the product owner / released to the public / brought up in the stand-up …and so on.*

7. Value

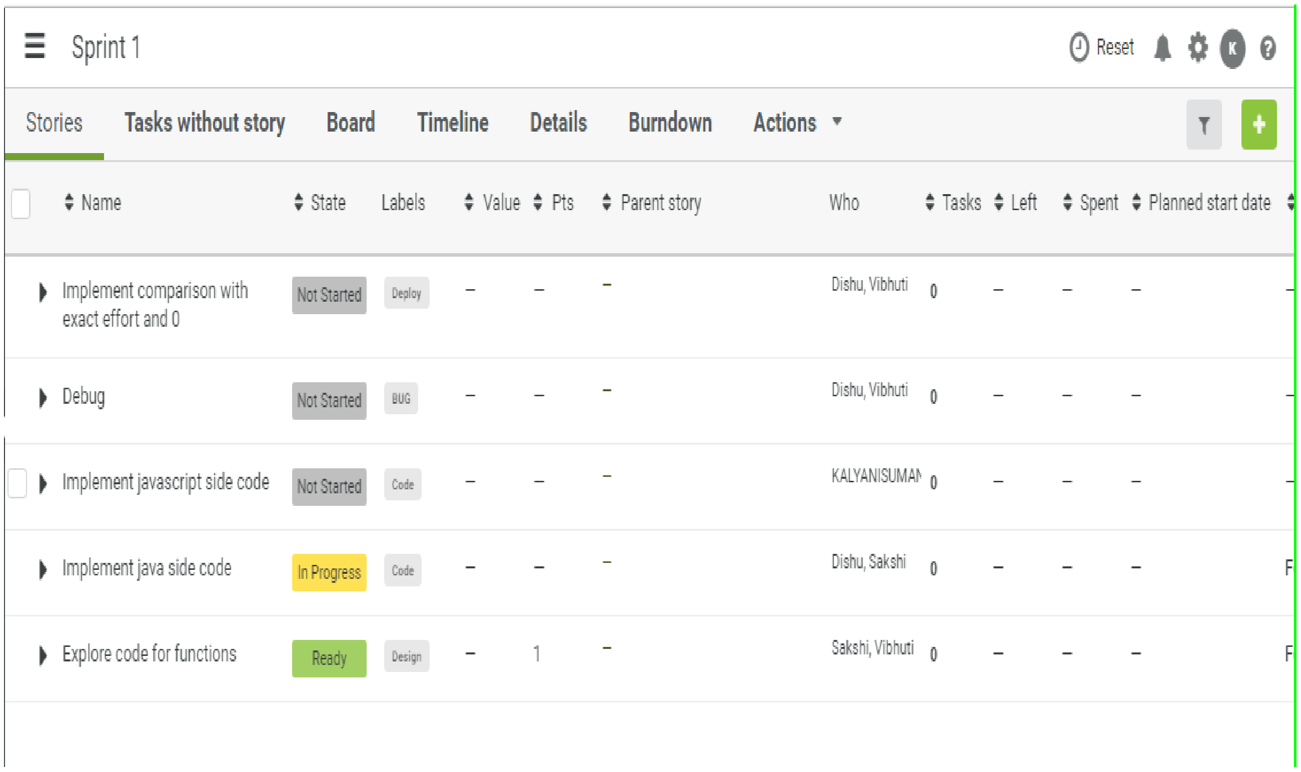
Stories may also be assigned a business value ('value' in Agilefant). Values are not currently used in any calculations, but those can be used as a support of decision making, for example, in planning.

8. Start, planned start, end, and due dates

A start date is the moment of time when a story is set from 'Not started' state to any other state. If the state is changed back to 'Not started' state, the start date will be cleared.
An end date is the moment of time when a story is set from any other state to either 'Done' or 'Deferred' state. If the state is changed back to another state than Done or Deferred, the end date will be cleared.
A planned start date is manually entered, used to tell the date when a story should be started. In a story details view, if story is started.

9. Labels

Stories can be labeled. For example, you might want to label stories according to whether they are bugs, usability improvements, strategic new cool things, being planned for release this-and-that, and so on. This can be done from the 'story details' view. Additionally, if you are looking for 'Themes', labels are what you need.



Sprint 1

Reset

Stories | Tasks without story | Board | Timeline | Details | Burndown | Actions

Name	State	Labels	Value	Pts	Parent story	Who	Tasks	Left	Spent	Planned start date
Implement comparison with exact effort and 0	Not Started	Deploy	–	–	–	Dishu, Vibhuti	0	–	–	–
Debug	Not Started	BUG	–	–	–	Dishu, Vibhuti	0	–	–	–
Implement javascript side code	Not Started	Code	–	–	–	KALYANISUMAN	0	–	–	–
Implement java side code	In Progress	Code	–	–	–	Dishu, Sakshi	0	–	–	–
Explore code for functions	Ready	Design	–	1	–	Sakshi, Vibhuti	0	–	–	–

LAB- 2

Aim: - *To learn various features of GitHub.*

Introduction: -

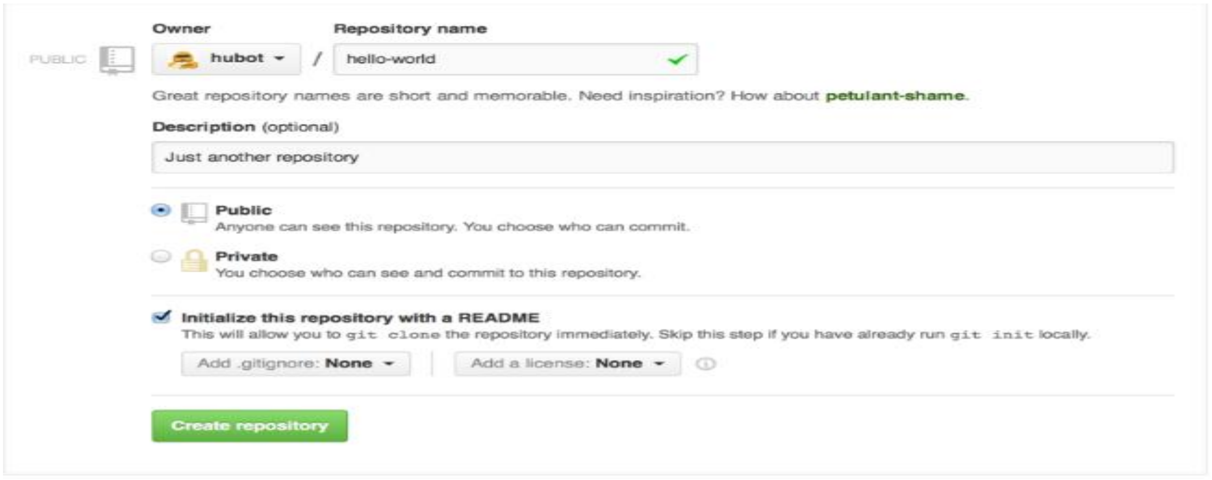
GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

Step1. Create a Repository

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a README, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. It also offers other common options such as a license file.
Your hello-world repository can be a place where you store ideas, resources, or even share and discuss things with others.
To create a new repository-

- ✓ In the upper right corner, next to your avatar or ident icon, click and then select New repository.
- ✓ Name your repository hello-world.
- ✓ Write a short description.
- ✓ Select Initialize this repository with a README.
- ✓ new-repo-form

Click Create repository.



Owner: hubot / Repository name: hello-world

Great repository names are short and memorable. Need inspiration? How about petulant-shame.

Description (optional): Just another repository

● Public — Anyone can see this repository. You choose who can commit.

○ Private — You choose who can see and commit to this repository.

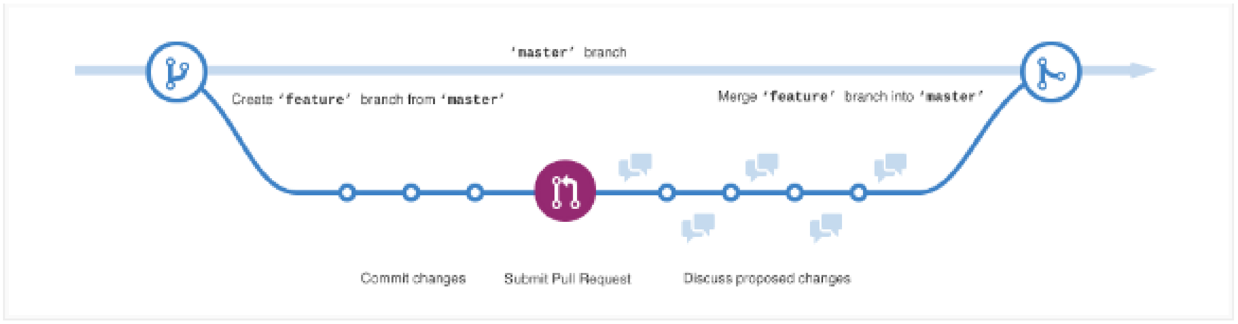
✓ Initialize this repository with a README — This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: None | Add a license: None

Step2. Create a Branch

Branching is the way to work on different versions of a repository at one time.
By default your repository has one branch named master which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to master.
When you create a branch off the master branch, you're making a copy, or snapshot, of master as it was at that point in time. If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.
This diagram shows:

- ✓ The master branch
- ✓ A new branch called feature (because we're doing 'feature work' on this branch)
- ✓ The journey that feature takes before it's merged into master


'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

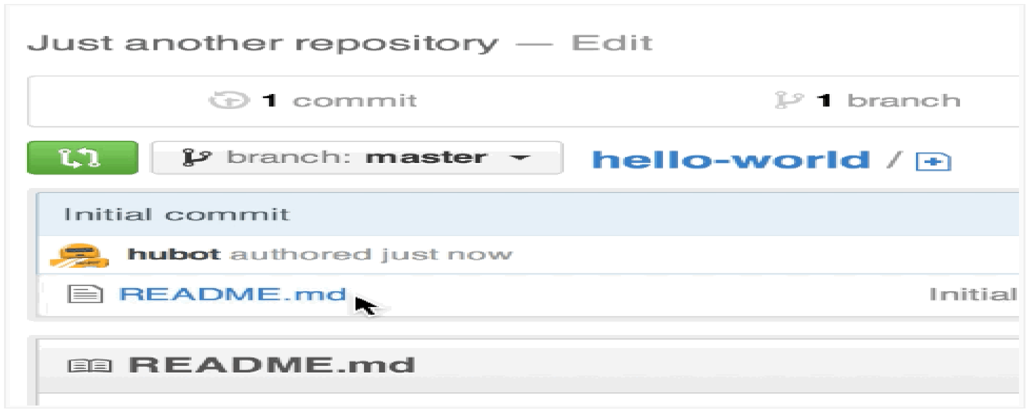
Have you ever saved different versions of a file? Something like:

- ✓ *story.txt*
- ✓ *story-joe-edit.txt*
- ✓ *story-joe-edit-reviewed.txt*

Branches accomplish similar goals in GitHub repositories.

Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our master (production) branch. When a change is ready, they merge their branch into master.
To create a new branch

- ✓ *Go to your new repository hello-world.*
- ✓ *Click the drop down at the top of the file list that says branch: master.*
- ✓ *Type a branch name, readme-edits, into the new branch text box.*
- ✓ *Select the blue Create branch box or hit "Enter" on your keyboard.*



Now you have two branches, master and readme-edits. They look exactly the same, but not for long! Next we'll add our changes to the new branch.

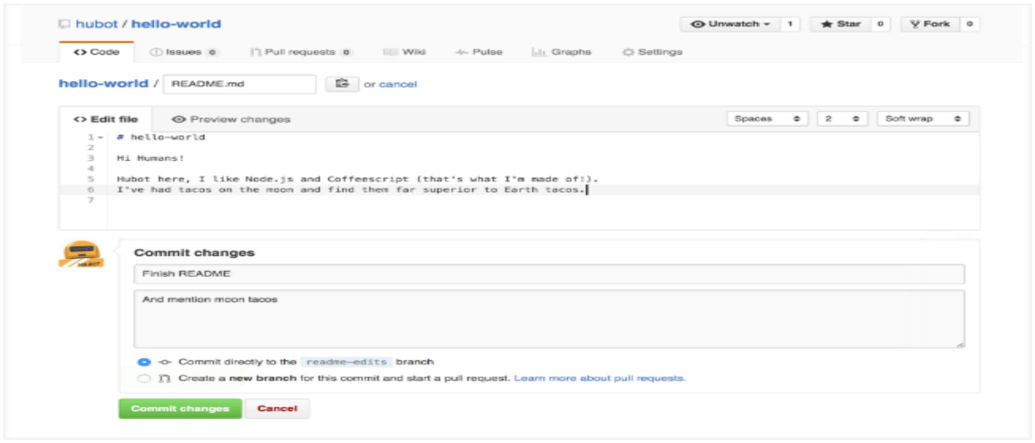
Step3. Make and commit changes.

Bravo! Now, you're on the code view for your readme-edits branch, which is a copy of master. Let's make some edits.

On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

Make and commit changes

- ✓ *Click the README.md file.*
- ✓ *Click the pencil icon in the upper right corner of the file view to edit.*
- ✓ *In the editor, write a bit about yourself.*
- ✓ *Write a commit message that describes your changes.*
- ✓ *Click Commit changes button.*



```
# hello-world

Hi Humans!

Hubot here, I like Node.js and Coffeescript (that's what I'm made of!).
I've had tacos on the moon and find them far superior to Earth tacos.
```

Commit changes

Finish README

And mention moon tacos

[✓ Commit directly to the readme-edits branch]

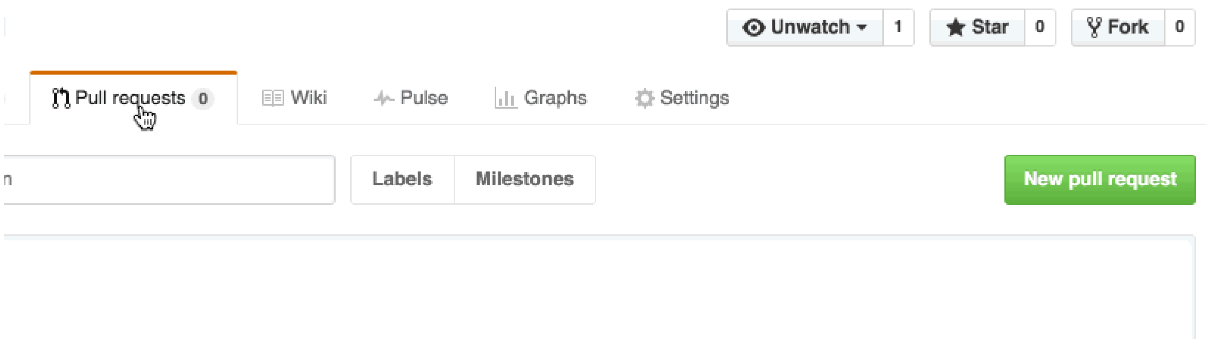
Create a new branch for this commit and start a pull request. Learn more about pull requests.

Step 4. Open a Pull Request

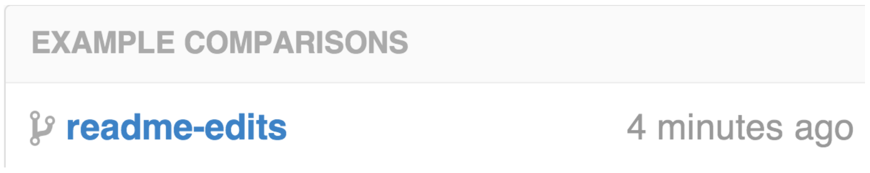
Nice edits! Now that you have changes in a branch off of master, you can open a pull request. Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you're proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.
As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.
By using GitHub's @mention system in your pull request message, you can ask for feedback from specific people or teams, whether they're down the hall or 10 time zones away.
You can even open pull requests in your own repository and merge them yourself. It's a great way to learn the GitHub Flow before working on larger projects.

Open a Pull Request for changes to the README

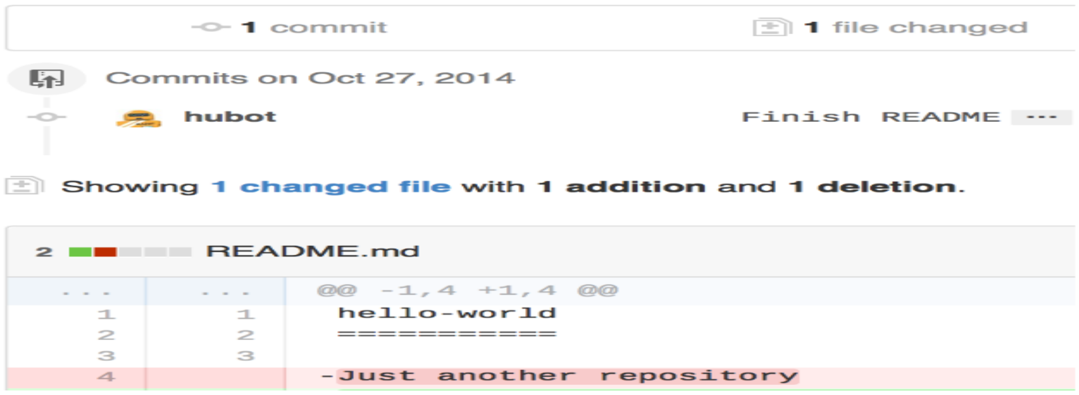
- ✓ *Click the Pull Request tab, and then from the Pull Request page, click the green New pull request button.*



- ✓ *Select the branch you made, readme-edits, to compare with master (the original).*



- ✓ *Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.*



1 commit 1 file changed

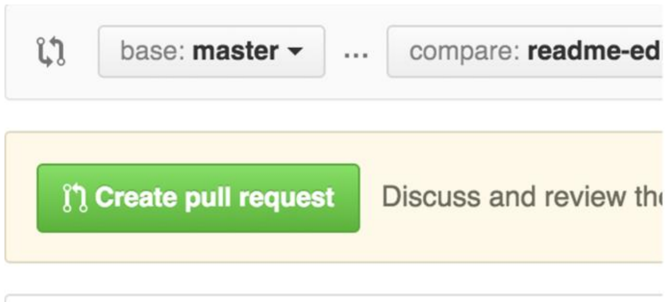
Commits on Oct 27, 2014

hubot Finish README

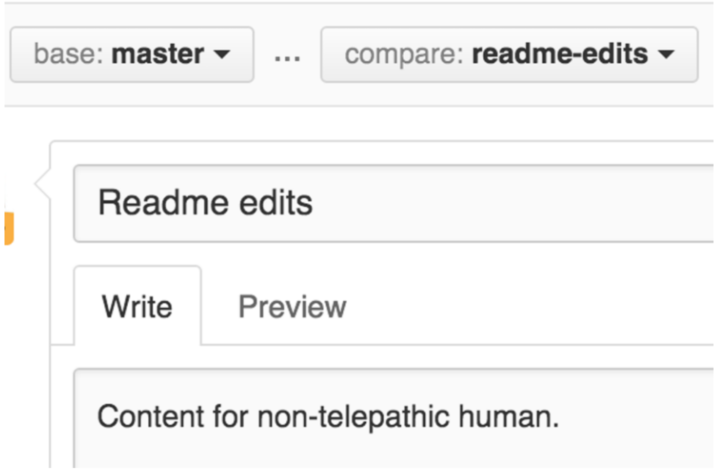
Showing 1 changed file with 1 addition and 1 deletion.

```
2 README.md
@@ -1,4 +1,4 @@
 hello-world
 ===========

-Just another repository
```

- ✓ *When you’re satisfied that these are the changes you want to submit, click the big green Create Pull Request button.*



- ✓ *Give your pull request a title and write a brief description of your changes.*



Readme edits

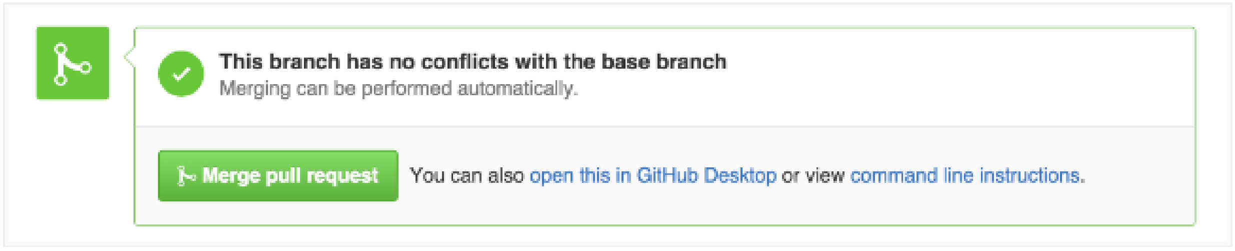
Content for non-telepathic human.

- ✓ *When you’re done with your message, click Create pull request!*

Step 5. Merge your Pull Request

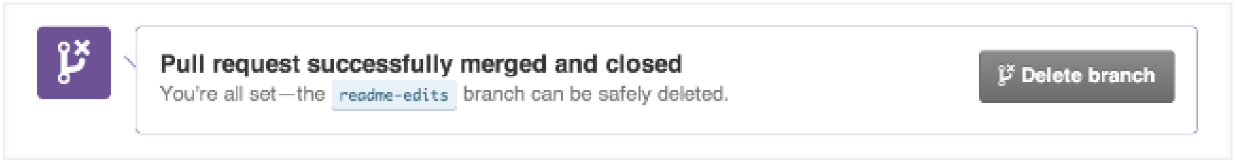
In this final step, it's time to bring your changes together – merging your readme-edits branch into the master branch.

- ✓ *Click the green Merge pull request button to merge the changes into master.*
- ✓ *Click Confirm merge.*
- ✓ *Go ahead and delete the branch, since its changes have been incorporated, with the Delete branch button in the purple box.*



This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

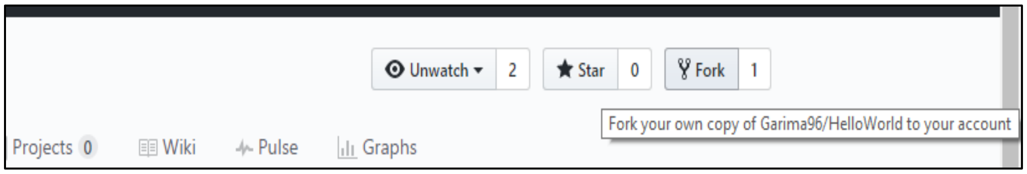


Pull request successfully merged and closed
You're all set—the `readme-edits` branch can be safely deleted.

Step 6. Forking a repository

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

- ✓ *It is a twostep process. On GitHub, navigate to the any other repository to want to fork.*
- ✓ *In the top-right corner of the page, click Fork.*



Fork your own copy of Garima96/HelloWorld to your account

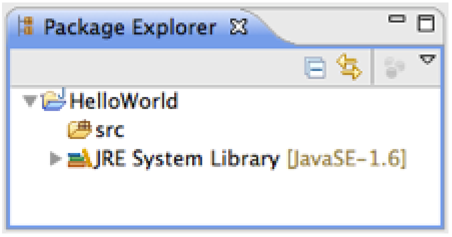
Now, we have a fork of the original repository.

LAB- 3

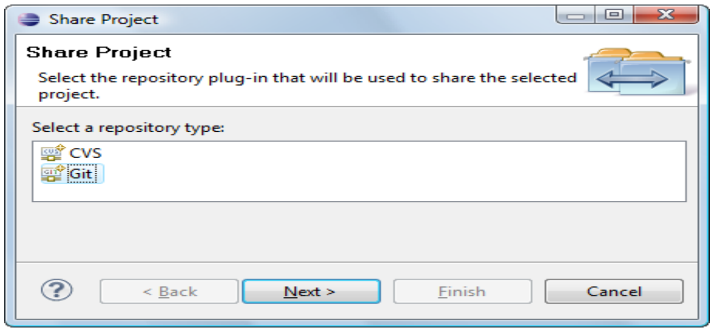
Aim : Enable Git repository in Eclipse

Create a new repository :

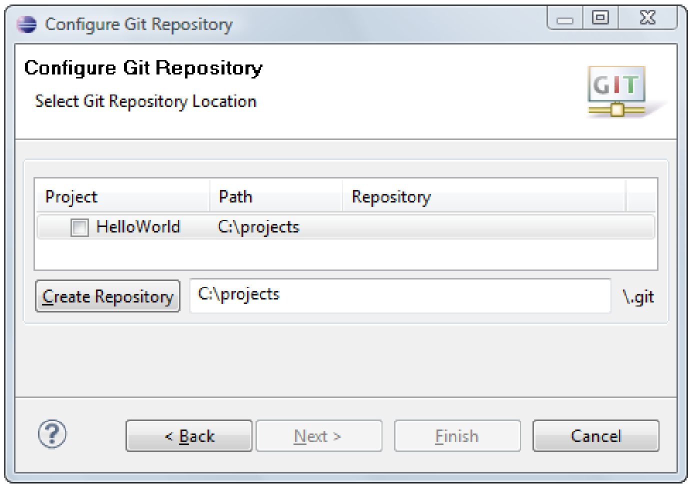
- ✓ *Create a new Java project HelloWorld.java.*



- ✓ *Select the project, click File > Team > Share Project.*
- ✓ *Select repository type Git and click Next.*


Share Project
Share Project
Select the repository plug-in that will be used to share the selected project.
Select a repository type:
CVS
Git
< Back
Next >
Finish
Cancel

- ✓ *To configure the Git repository select the new project HelloWorld.*



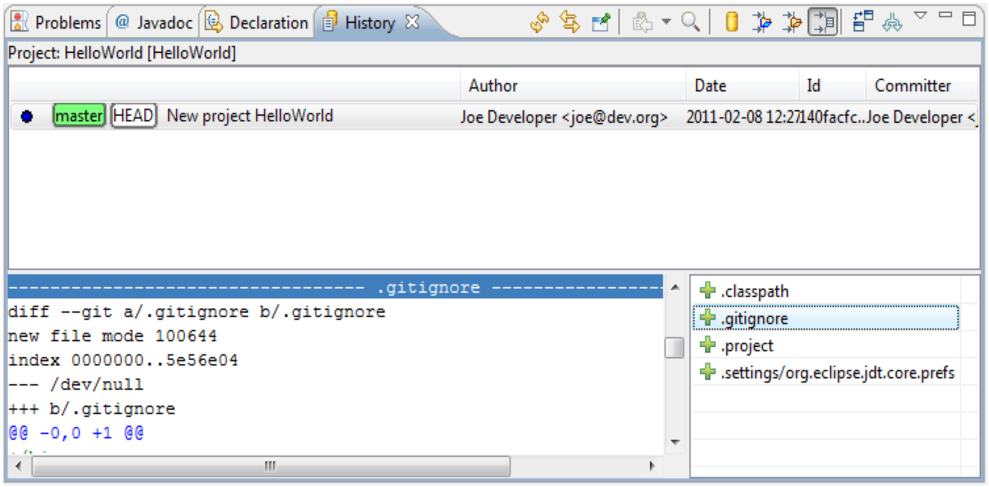
Project	Path	Repository
☐ HelloWorld	C:\projects	

Create Repository: C:\projects \.git

- ✓ *Click Create Repository to initialize a new Git repository for the HelloWorld project. If your project already resides in the working tree of an existing Git repository the repository is chosen automatically.*
- ✓ *Click Finish to close the wizard.*

Inspect history of a repository:

- ✓ *Click Team --> Show in History from the context menu to inspect the history of a resource.*



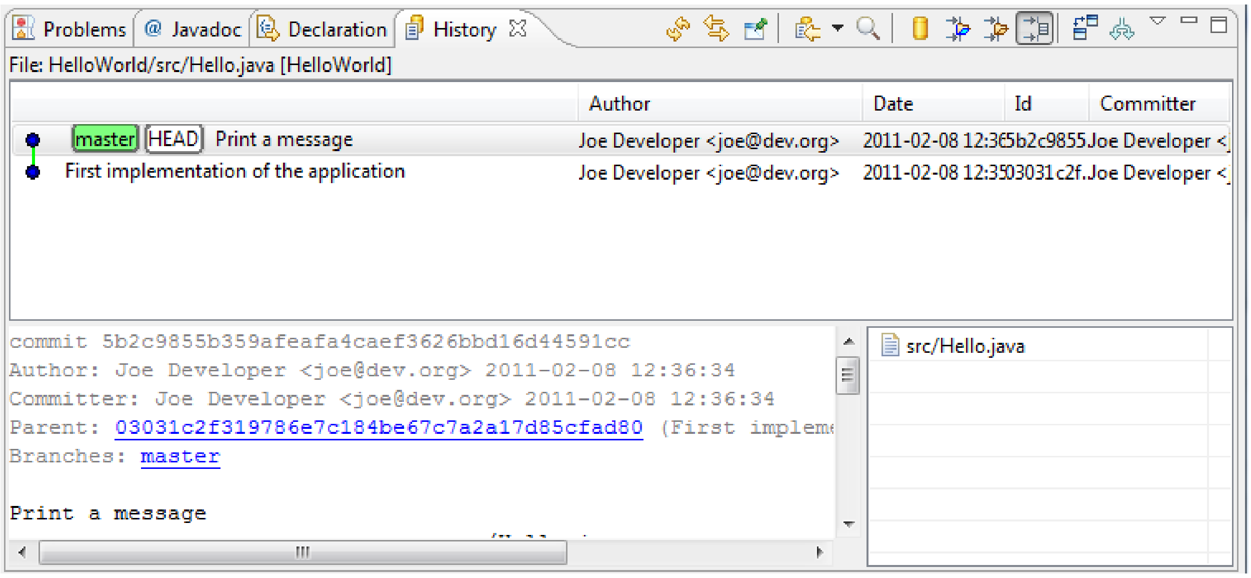
Project: HelloWorld [HelloWorld]

	Author	Date	Id	Committer
master HEAD New project HelloWorld	Joe Developer <joe@dev.org>	2011-02-08 12:27	140facfc..	Joe Developer <

```
---------------------------------- .gitignore ----------------
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..5e56e04
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
```

.classpath
.gitignore
.project
.settings/org.eclipse.jdt.core.prefs

- ✓ *Create a new Java class Hello.java and implement it.*
- ✓ *Add it to version control and commit your change.*
- ✓ *Improve your implementation and commit the improved class.*
- ✓ *The resource history should now show 2 commits for this class.*



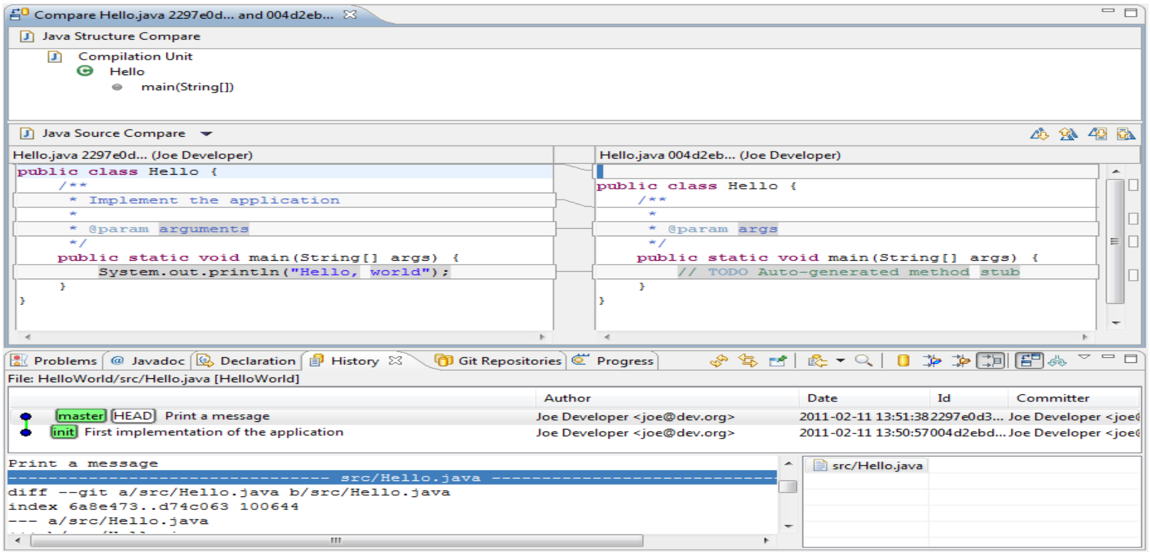
File: HelloWorld/src/Hello.java [HelloWorld]

	Author	Date	Id	Committer
master HEAD Print a message	Joe Developer <joe@dev.org>	2011-02-08 12:36	5b2c9855	Joe Developer <
First implementation of the application	Joe Developer <joe@dev.org>	2011-02-08 12:35	03031c2f.	Joe Developer <

```
commit 5b2c9855b359afeafa4caef3626bbd16d44591cc
Author: Joe Developer <joe@dev.org> 2011-02-08 12:36:34
Committer: Joe Developer <joe@dev.org> 2011-02-08 12:36:34
Parent: 03031c2f319786e7c184be67c7a2a17d85cfad80 (First impleme
Branches: master

Print a message
```

src/Hello.java

- ✓ *Click the Compare Mode toggle button in the History View.*



Compare Hello.java 2297e0d... and 004d2eb...

Java Structure Compare

Compilation Unit
Hello
main(String[])

Java Source Compare

Hello.java 2297e0d... (Joe Developer)

```
public class Hello {
    /**
     * Implement the application
     *
     * @param arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
```

Hello.java 004d2eb... (Joe Developer)

```
public class Hello {
    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

File: HelloWorld/src/Hello.java [HelloWorld]

	Author	Date	Id	Committer
master HEAD Print a message	Joe Developer <joe@dev.org>	2011-02-11 13:51:38	2297e0d3...	Joe Developer <joe
init First implementation of the application	Joe Developer <joe@dev.org>	2011-02-11 13:50:57	004d2ebd...	Joe Developer <joe

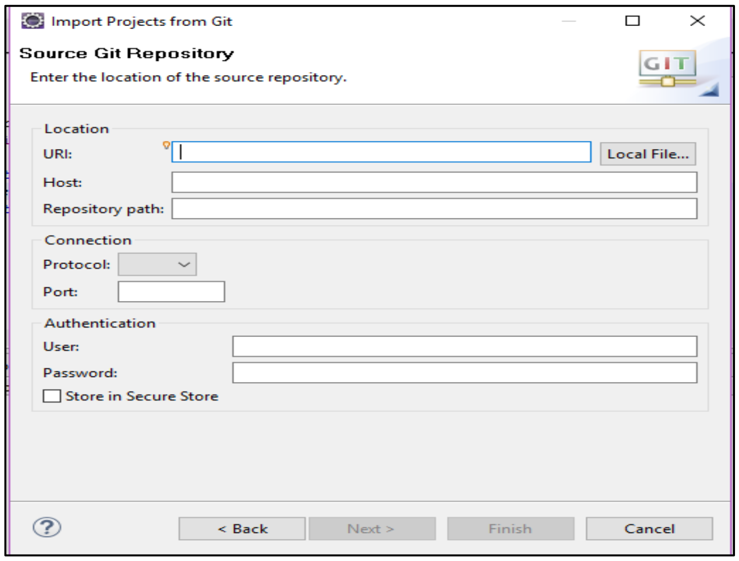
```
Print a message
-------------------------------- src/Hello.java --------------------------------
diff --git a/src/Hello.java b/src/Hello.java
index 6a8e473..d74c063 100644
--- a/src/Hello.java
```

src/Hello.java

- ✓ *Double click src/Hello.java in the Resource list of the History View to open your last committed change in the Compare View.*

Cloning remote repository :

1) *Click on File → Imports →Git→clone URL.*
2) *Following dialog box will appear.*



Import Projects from Git

Source Git Repository

Enter the location of the source repository.

Location
URI:
Host:
Repository path:

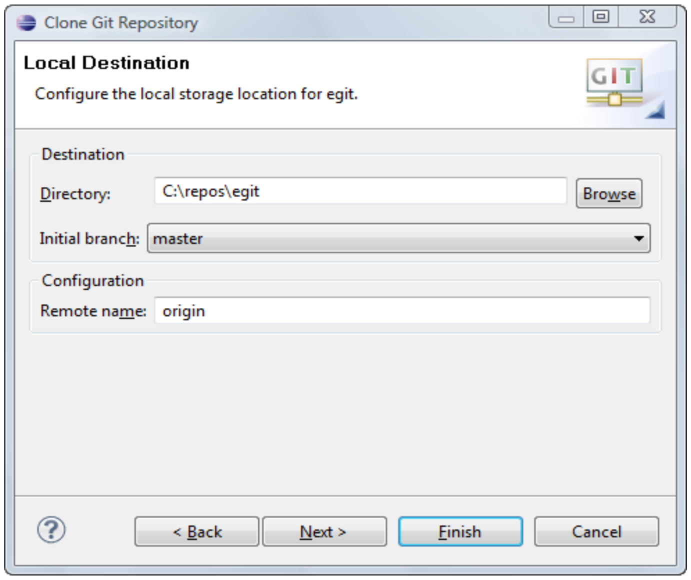
Connection
Protocol:
Port:

Authentication
User:
Password:
☐ Store in Secure Store

- ✓ *Enter the URL location of the remote repository.*

User - The user name used for authentication.
Password - The password used for authentication.

- ✓ *Click on Next.*
- ✓ *On the next page choose which branches shall be cloned from the remote repository.*
- ✓ *On the next page define where you want to store the repository on the local file system and define some initial settings.*



Local Destination
Configure the local storage location for egit.

Destination
Directory: C:\repos\egit Browse
Initial branch: master

Configuration
Remote name: origin

3) *Click on finish button.*

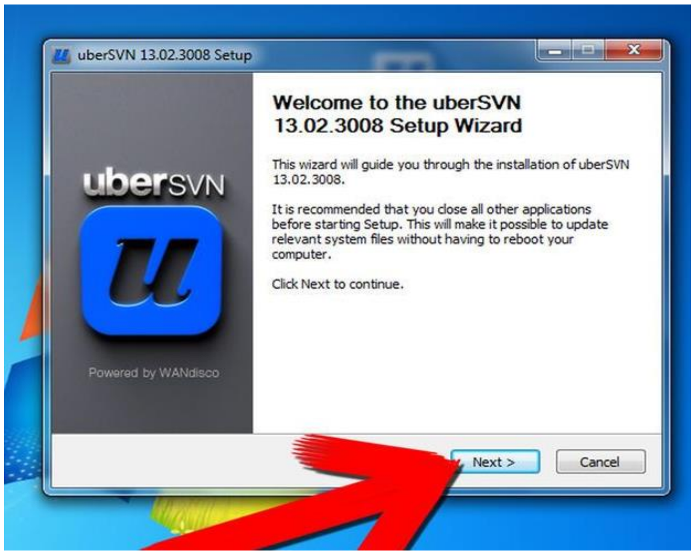
LAB- 4

Aim:- *Introduction to SVN.*

VisualSVN Server is a freeware Apache Subversion server package for Windows. A SVN repository (or Subversion repository) is a collection of files and directories, bundled together in a special database that also records a complete history of all the changes that have ever been made to these files. Subversion is a free/open source version control system (VCS). That is, Subversion manages files and directories, and the changes made to them, over time. This allows you to recover older versions of your data or examine the history of how your data changed.

Step – 1 : Install SVN server

- ✓ *Open Apache Subversion site.*
- ✓ *Download the latest apache subversion release.*
- ✓ *Run the installer, making sure that you have admin permissions.*
- ✓ *If an Open File prompt appears, click the Run button.*



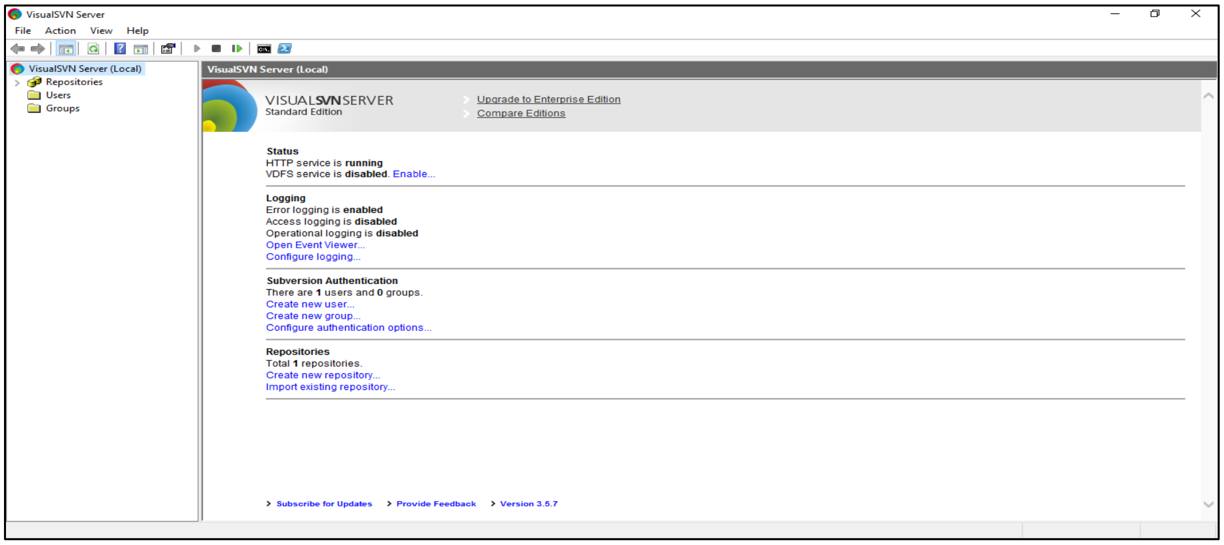
Welcome to the uberSVN 13.02.3008 Setup Wizard

This wizard will guide you through the installation of uberSVN 13.02.3008.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

- ✓ *The Welcome to SVN Setup Wizard starts. Just click Next>.*
- ✓ *Wait until all the files have been extracted. Click Next >.*
- ✓ *Successfully installed SVN.*



Status
HTTP service is **running**
VDFS service is **disabled**. Enable...

Logging
Error logging is **enabled**
Access logging is **disabled**
Operational logging is **disabled**
Open Event Viewer...
Configure logging...

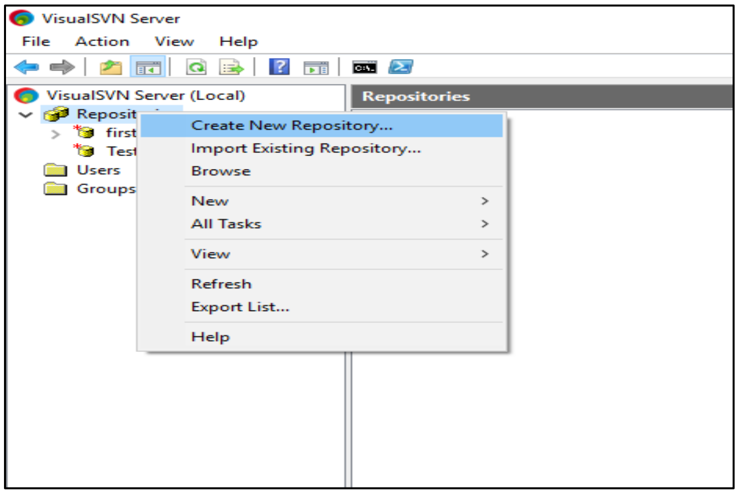
Subversion Authentication
There are **1** users and **0** groups.
Create new user...
Create new group...
Configure authentication options...

Repositories
Total **1** repositories.
Create new repository...
Import existing repository...

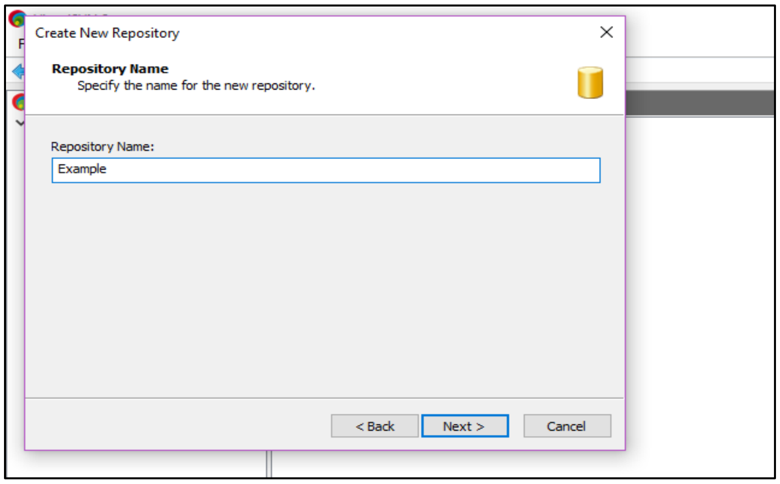
› Subscribe for Updates › Provide Feedback › Version 3.5.7

Step – 2 : Create a new repository in SVN

- ✓ *Right click on repository option on the left hand side tab.*
- ✓ *Click on create new repository.*

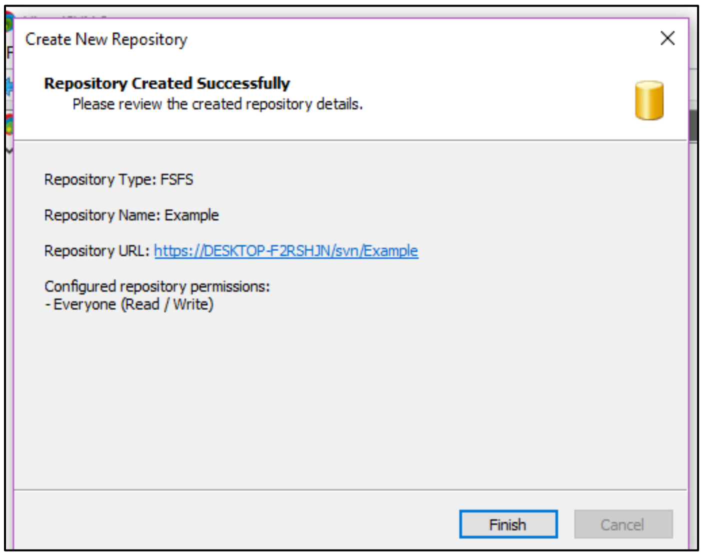


Enter the repository name in the prompt box.



Repository Name: Example

1) *Click on next -> next in the dialog boxes.*
2) *Following screen appears after the successful creation of repository.*


Create New Repository
Repository Created Successfully
Please review the created repository details.
Finish
Cancel

Repository Type: FSFS

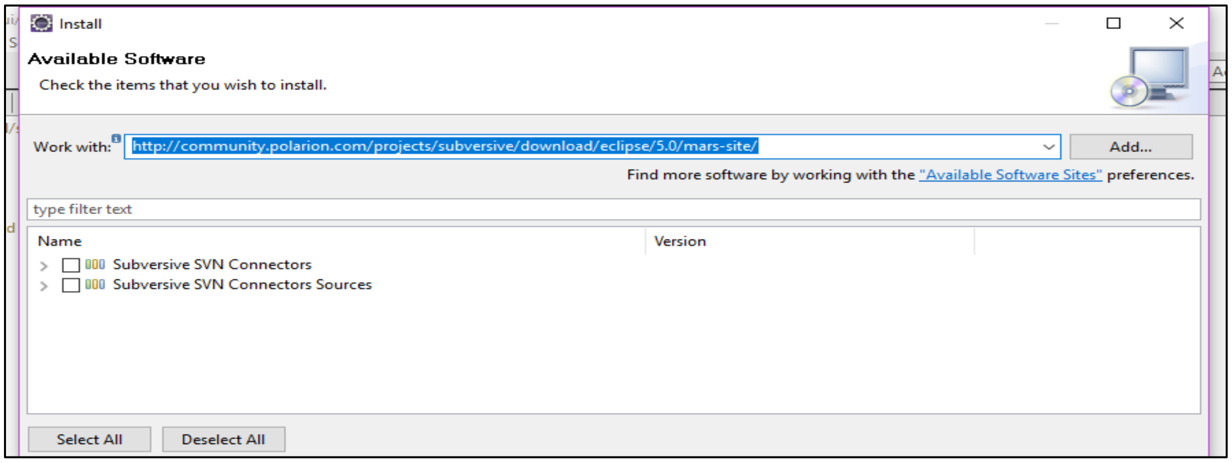
Repository Name: Example

Repository URL: https://DESKTOP-F2RSHJN/svn/Example

Configured repository permissions:
- Everyone (Read / Write)

Step – 3 : Enable Subversion (SVN) in Eclipse IDE

- ✓ *Open Eclipse IDE.*
- ✓ *Click on Help -> Install new software*
- ✓ *Following screen appears .Select the components you want to install and click ADD. Restart Eclipse to take the effect.*



Available Software

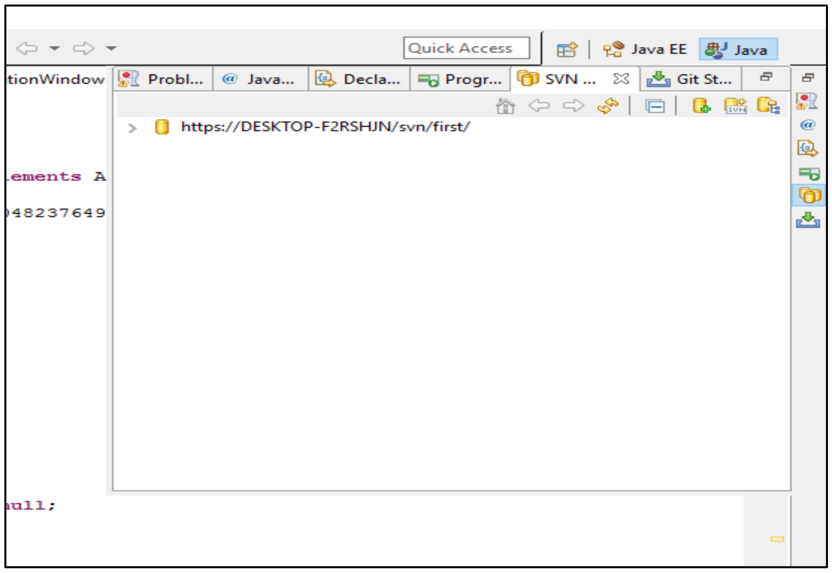
Check the items that you wish to install.

Work with: http://community.polarion.com/projects/subversive/download/eclipse/5.0/mars-site/

Find more software by working with the "Available Software Sites" preferences.

Name	Version
Subversive SVN Connectors	
Subversive SVN Connectors Sources	

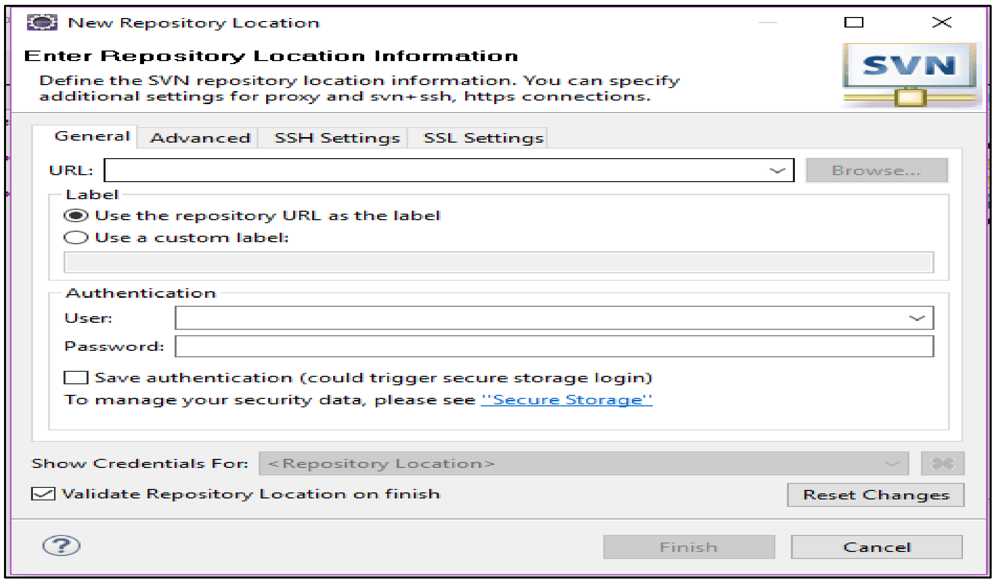
- ✓ *Select "Windows" –> "Open Perspective" –> "Other…", choose "**SVN Repositories**".*
- ✓ *Now, you can perform SVN functionality in this "SVN Repositories" perspective.*



https://DESKTOP-F2RSHJN/svn/first/

Step – 4 : Add a SVN repository in Eclipse IDE

- ✓ *Click on Small (+) button in the SVN repository tab.*
- ✓ *Enter the URL of the repository you want to add and click on Finish.*



New Repository Location

Enter Repository Location Information

Define the SVN repository location information. You can specify additional settings for proxy and svn+ssh, https connections.

General | Advanced | SSH Settings | SSL Settings

URL:

Label
- ● Use the repository URL as the label
- ○ Use a custom label:

Authentication

User:

Password:

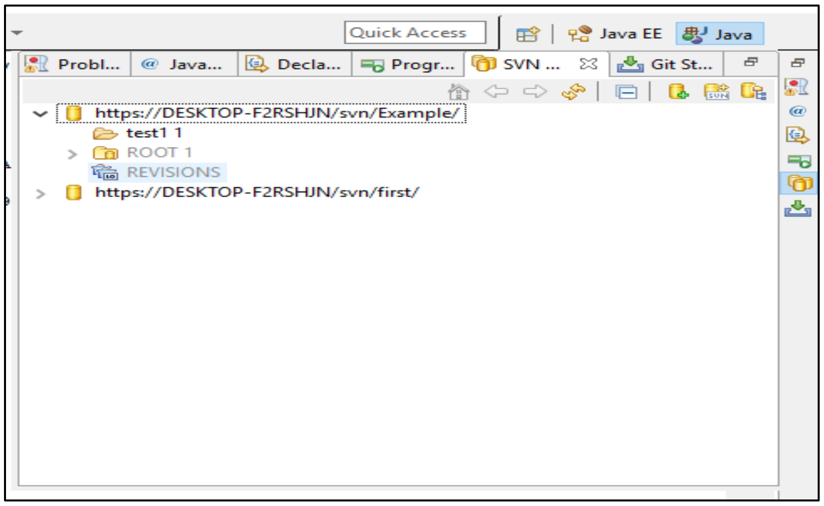
☐ Save authentication (could trigger secure storage login)

To manage your security data, please see "Secure Storage"

Show Credentials For: <Repository Location>

☑ Validate Repository Location on finish

- ✓ *Repository will get added .Here; Example is the repository which I added.*



```
https://DESKTOP-F2RSHJN/svn/Example/
    test1 1
    ROOT 1
    REVISIONS
https://DESKTOP-F2RSHJN/svn/first/
```

- ✓ *Now we can make changes as we want. Different versions of the same file will get stored accordingly.*

LAB-5

Aim : *Introduction to **Junit***

Introduction

JUnit is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit test framework provides the following important features

- ✓ *Fixtures*
- ✓ *Test suites*
- ✓ *Test Runners*
- ✓ *Junit classes*

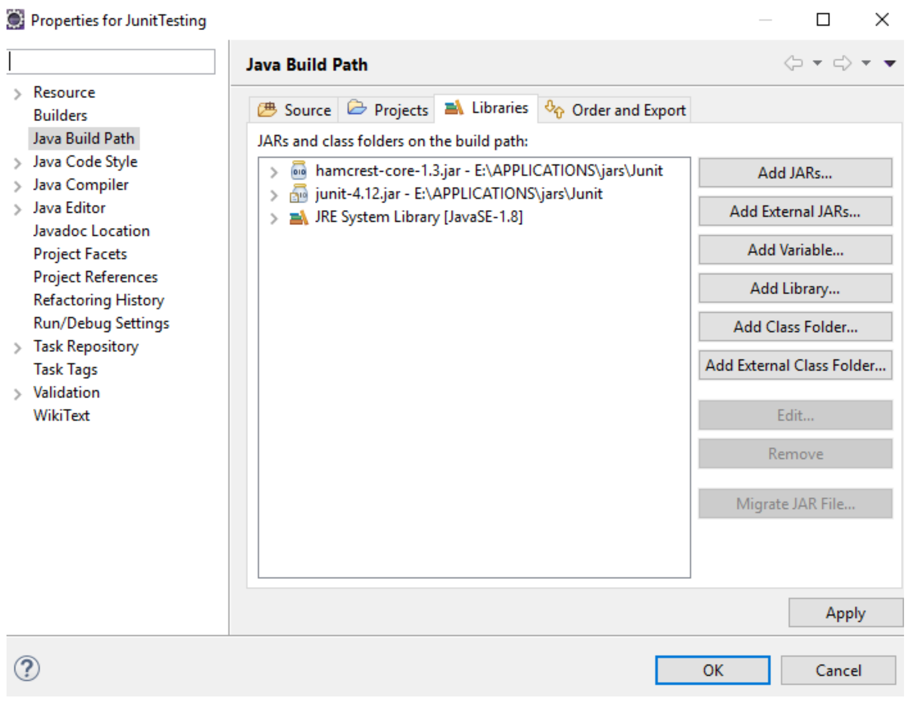
JUnit in Eclipse

I. Download Jar Files

Download latest jar files from "https://github.com/junit-team/junit4/wiki/Download-and-Install"

II. Import Jar to Build Path

Right click on your project, goto Build path>Configure Build path


Properties for JunitTesting

Java Build Path

JARs and class folders on the build path:

- hamcrest-core-1.3.jar - E:\APPLICATIONS\jars\Junit
- junit-4.12.jar - E:\APPLICATIONS\jars\Junit
- JRE System Library [JavaSE-1.8]

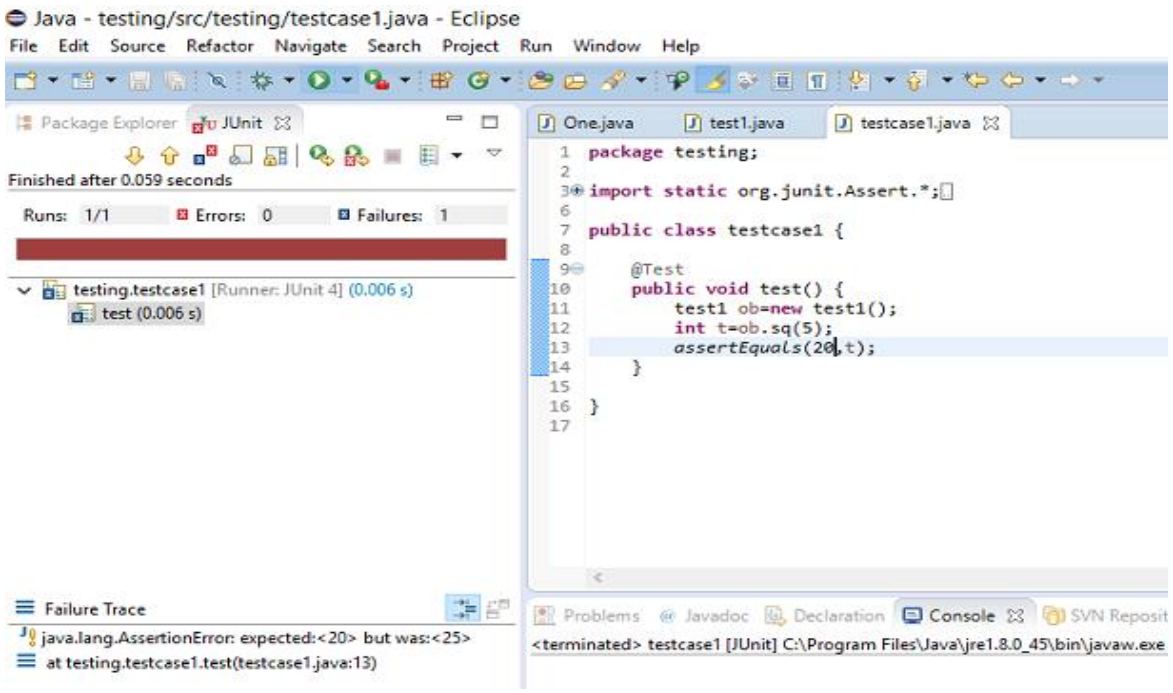
Click on Add External Jars and select the jar files ([junit.jar]() and [hamcrest-core.jar]()) and press OK, the jars now have been imported.

III. Junit test class creation.

Create a project and a class which is to be tested, now right-click on project New>others>Java>JUnit>Junit Test Cases. It will ask to create a new class.

IV. Testing

Write some code in Class file and test some redefined functions

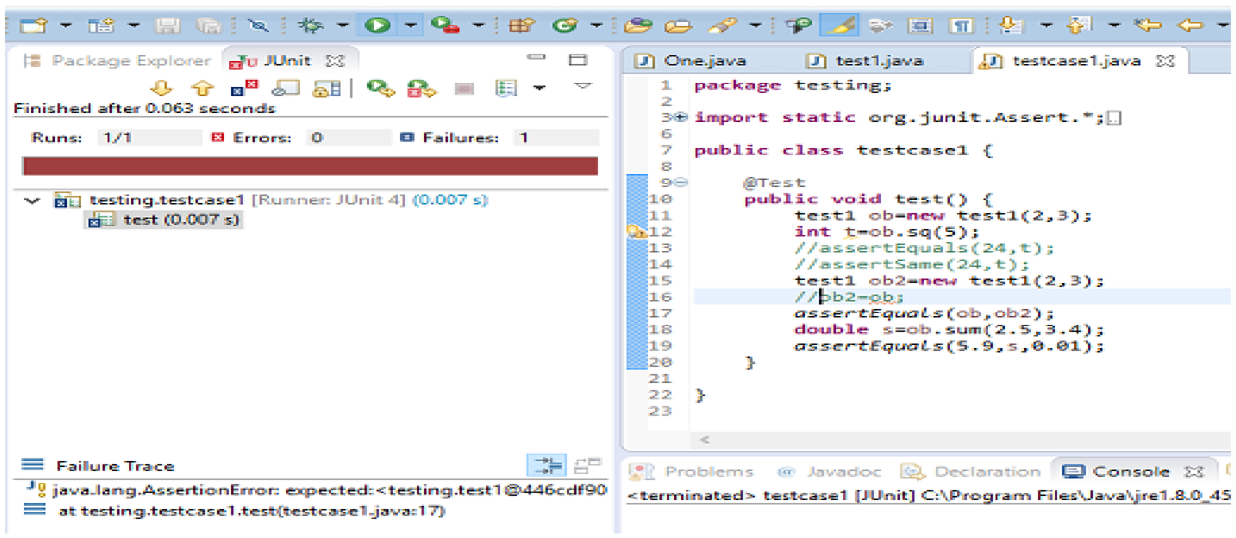

Java - testing/src/testing/testcase1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
One.java
test1.java
testcase1.java
Problems
Javadoc
Declaration
Console
SVN Reposit

```
Finished after 0.059 seconds
Runs: 1/1    Errors: 0    Failures: 1
testing.testcase1 [Runner: JUnit 4] (0.006 s)
    test (0.006 s)
```

```java
package testing;

import static org.junit.Assert.*;

public class testcase1 {

    @Test
    public void test() {
        test1 ob=new test1();
        int t=ob.sq(5);
        assertEquals(20,t);
    }

}
```

```
Failure Trace
java.lang.AssertionError: expected:<20> but was:<25>
at testing.testcase1.test(testcase1.java:13)
```

```
<terminated> testcase1 [JUnit] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe
```



```
Finished after 0.063 seconds
Runs: 1/1    Errors: 0    Failures: 1
testing.testcase1 [Runner: JUnit 4] (0.007 s)
    test (0.007 s)
```

```java
package testing;

import static org.junit.Assert.*;

public class testcase1 {

    @Test
    public void test() {
        test1 ob=new test1(2,3);
        int t=ob.sq(5);
        //assertEquals(24,t);
        //assertSame(24,t);
        test1 ob2=new test1(2,3);
        //ob2=ob;
        assertEquals(ob,ob2);
        double s=ob.sum(2.5,3.4);
        assertEquals(5.9,s,0.01);
    }

}
```

```
Failure Trace
java.lang.AssertionError: expected:<testing.test1@446cdf90
at testing.testcase1.test(testcase1.java:17)
```

```
<terminated> testcase1 [JUnit] C:\Program Files\Java\jre1.8.0_45
```

LAB- 6

Aim: To manually detect design smells from linked list code and refactor it.

Description: There are 7 design smells as follows:

Rigidity

Rigidity is the tendency of software to be difficult to change. A single change causes a cascade of subsequent changes in dependent modules. The more modules that must be changed, the more rigid the design. Many developers recognize this as they chase, what should be a simple change, through the system touching ever more modules and taking far more time than the original estimate. When asked why their estimate was so poor, they repeat the traditional software development lament: "It was a lot more complicated than I thought."

Fragility

Fragility is the tendency of software to break in many places when a single change is made. Often the problems occur in places that have no obvious relation to the area that was changed. As the fragility of a module increases the likelihood a change will introduce unexpected problems approaches certainty.

Immobility

A design is immobile when it contains parts that could be useful in other systems, but the effort and risk of separating them from the original system are too great.

Viscosity

When faced with a change, developers often have several ways to implement the desired change. Some ways preserve the design, others do not (they are called hacks). A system has a high viscosity if the design-preserving changes are more difficult to use than the hacks. It's easy to do the wrong thing and difficult to do the right thing. Apart from viscosity of the software itself, there is also viscosity of the system as a whole. If running unit tests and compilation takes a lot of time, it is likely for a developer to bypass procedures and implement a hack without running all automated tests.

Needless repetition

In short copy and pasting code throughout the system. Usually this happens when necessary abstractions have not been made, either because of a lack of time or a lack of experience. What remains is procedural software. Code may not have been literally copy and pasted but business rules are not implemented in a clear and distinct way.

Opacity

Opacity is the tendency of a module to be difficult to understand. Code can be written in a clear and expressive manner, or it can be written in an opaque and convoluted way. Code that evolves over time tends to become more difficult to understand over time.

Needless complexity

This may very well be the most important smell of bad design. Developers, in a passionate attempt to avoid the other 6 smells, introduce all sorts of abstractions and preparations for potential changes in the future. Good software design is lightweight, flexible, easy to read and understand and above all easy to change so you don't have to keep into account all potential changes in the future.

Procedure:

1. *Write a code to create and reverse a linked list.*

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node
{
int data;
node *link;
};
void display(node *s)
{
while(s!=NULL)
{
cout<<s->data<<" ";
s=s->link;
}
}
struct node* middleNode(struct node *s)
{
struct node *a,*b;
a=b=s;
for(;b->link!=NULL&&b->link->link!=NULL;)
{
a=a->link;
b=b->link->link;
}
return (a);
}
```

```
node *reverselist(node *start)
{
node *p,*q,*r;
p=NULL,q=start,r=start;
while(r!=NULL)
{
r=q->link;
q->link=p;
p=q;
q=r;
}
return p;
}
int palindrome(node *start)
{
node *v=middleNode(start);
node *u=reverselist(v->link);
while(u!=NULL)
{
if(start->data!=u->data)
{
return 0;
}
u=u->link;
}
if(u==NULL)
return 1;
}
int main()
{
node *start=(struct node *)malloc(sizeof(struct node));
int n;
cout<<"enter the number of elements\n";
cin>>n;
cout<<"enter elements\n";
node *s=start;
int i,info;
for(i=0;i<n;i++)
{
cin>>info;
s->data=info;
s->link=(struct node *)malloc(sizeof(struct node));
if(i!=n-1)
s=s->link;
}
s->link=NULL;
int ans=palindrome(start);
if(ans==1)
cout<<"a palindrome\n";
else
cout<<"not a palindrome\n";
}
```

2. Remove different design smells manually.

The above code contains smells of

- ✓ Needless complexity as display function has been kept for future use but is not needed yet, 2. It lacks comments therefore, lacks understandability.
- ✓ The functions are not appended with word List therefore lack Understandability
- ✓ Some methods are long therefore, extract method refactoring has to be done.

```
#include<iostream>
#include<stdlib.h>
using namespace std;
//structure of node
struct node
{
   int data;
   node *link;
};
//creation of list
node * createList(struct node *start,int n)
{
   node *s=start;
      int i,info;
   for(i=0;i<n;i++)
   {
       cin>>info;
       s->data=info;
       s->link=(struct node *)malloc(sizeof(struct node));
       if(i!=n-1)
        s=s->link;
   }
   s->link=NULL;
   return start;
```

```
}
void displayList(node *s)
{
    while(s!=NULL)
    {
        cout<<s->data<<" ";
        s=s->link;
    }
}
struct node* middleNodeList(struct node *s)
{
struct node *a,*b;
a=b=s;
for(;b->link!=NULL&&b->link->link!=NULL;)
{
a=a->link;
b=b->link->link;
}
return (a);
}

node * reverseList(node *start)
{
    node *p,*q,*r;
    p=NULL,q=start,r=start;
    while(r!=NULL)
    {
     r=q->link;
     q->link=p;
     p=q;
     q=r;
    }
    return p;
}
int palindromeList(node *start)
{
    node *v=middleNodeList(start);
    node *u=reverseList(v->link);
    while(u!=NULL)
    {
        if(start->data!=u->data)
        {
            return 0;
        }
        u=u->link;
    }
    if(u==NULL)
        return 1;
}

int main()
{
    node *start=(struct node *)malloc(sizeof(struct node));
    int n;
    //entering the elements
    cout<<"enter the number of elements\n";
    cin>>n;
    cout<<"enter elements\n";
    //creating list
    start=createList(start,n);
    //calling palindrome
    int ans=palindromeList(start);
    if(ans==1)
        cout<<"a palindrome\n";
    else
        cout<<"not a palindrome\n";
}
```

LAB- 7

Aim: *Automate a set of given tests using Test automation tool- JDeodorant.*

Description: *JDeodorant is an Eclipse plug-in that identifies design problems in software, known as bad smells, and resolves them by applying appropriate refactorings.*

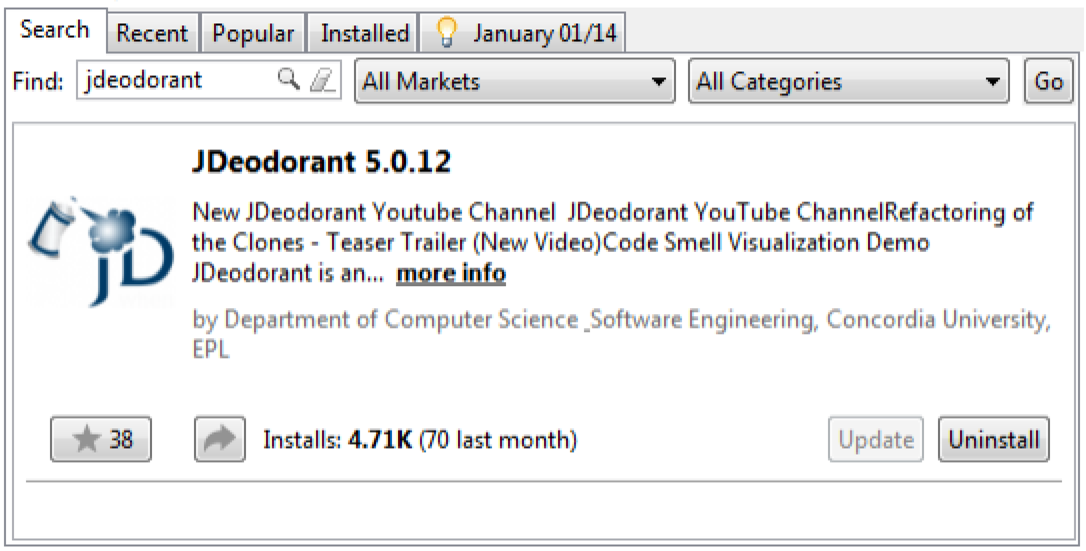
JDeodorant employs a variety of novel methods and techniques in order to identify code smells and suggest the appropriate refactorings that resolve them.

For the moment, the tool identifies five kinds of bad smells, namely ***Feature Envy***, ***Type Checking***, ***Long Method***, ***God Class*** *and* ***Duplicated Code***.

- ✓ *Feature Envy problems are resolved by appropriate Move Method refactorings.*
- ✓ *Type Checking problems are resolved by appropriate Replace Conditional with Polymorphism and Replace Type code with State/Strategy refactorings.*
- ✓ *Long Method problems are resolved by appropriate Extract Method refactorings.*
- ✓ *God Class problems are resolved by appropriate Extract Class refactorings.*
- ✓ *Duplicated Code problems are resolved by appropriate Extract Clone refactorings.*

Installation

- ✓ *The installation of JDeodorant is very easy! You have two options: a) Eclipse Marketplace Client: Go to Help -> Eclipse Marketplace... , search for JDeodorant in the Find: box, and click Install.After the installation Bad Smells menu item should appear on Eclipse menu bar..*
- ✓ *The identification of code smells can be performed on a whole Java Project, a Package Fragment Root of a project, a Package Fragment of a project, a Compilation Unit, a Type (along with its nested types), and a Method of a Type (only in the case of Long Method code smell) by selecting the appropriate element on the Package Explorer.*



Find: jdeodorant
All Markets
All Categories

JDeodorant 5.0.12

New JDeodorant Youtube Channel JDeodorant YouTube ChannelRefactoring of the Clones - Teaser Trailer (New Video)Code Smell Visualization Demo JDeodorant is an... **more info**

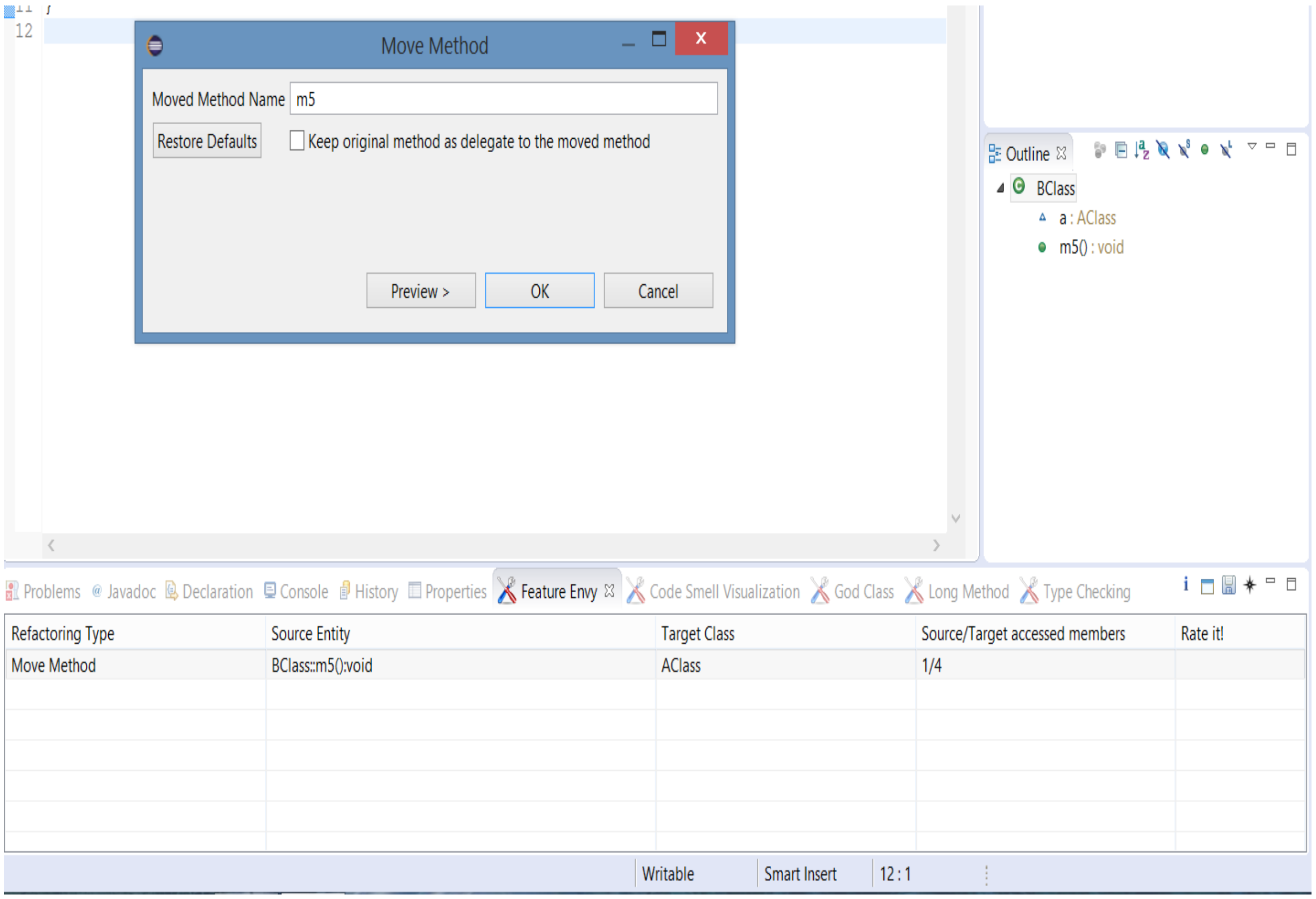
by Department of Computer Science _Software Engineering, Concordia University, EPL

38 Installs: **4.71K** (70 last month)

Feature Envy

- ✓ *Open the Package Explorer View (Window -> Show View -> Package Explorer) and the Feature Envy View (Bad Smells -> Feature Envy).*
- ✓ *Import the Java project to be analyzed for Feature Envy bad smells and select it on the Package Explorer View.*
- ✓ *From the Feature Envy view click on the "Identify Bad Smells" button to run the detection process.*
- ✓ *In the table of the Feature Envy view you can see all refactoring opportunities identified for the selected project sorted by the value of the Entity Placement metric in ascending order.*
- ✓ *Each row of the table contains information about:*
    - ✓ *the Refactoring Type of the suggestion (Move Method).*
    - ✓ *the Source Entity (source class along with the method which is suggested to be moved).*
    - ✓ *the Target Class.*
    - ✓ *the value of the Entity Placement metric after the application of the corresponding refactoring.*
- ✓ *There is an additional row that shows the value of the Entity Placement metric of the current system.*
- ✓ *Double-clicking a row of the table will open both target and source classes in separate editors of Eclipse. The method to be moved will be highlighted.*
- ✓ *In case there is a dependency between two refactoring suggestions, a tooltip indicates which refactoring should be applied first by hovering over the dependent suggestion.*
- ✓ *To apply a refactoring select a row from the table and click on the "Apply Refactoring" button.*
- ✓ *The refactoring wizard allows to change the default name of the method suggested to be moved to the target class. The validity of the new name is automatically checked.*
- ✓ *After applying or undoing the application of refactoring, you have to press the "Identify Bad Smells" button again to refresh the table.*

```
    public class AClass {
    void m1()
    {
    System.out.println("Hello1");
    }
    void m2()
    {
    System.out.println("Hello2");
    }
    void m3()
    {
    System.out.println("Hello3");
    }
    void m4()
    {
    System.out.println("Hello4");
    }
}
public class BClass {

    AClass a=new AClass();
    public void m5() {
            a.m1();
            a.m2();
            a.m3();
            a.m4();
    }
}
public class Main {

    public static void main(String args[])
    {
            BClass b=new BClass();
            b.m5();
    }
}
```



Refactoring Type	Source Entity	Target Class	Source/Target accessed members	Rate it!
Move Method	BClass::m5():void	AClass	1/4	

Type Checking

- ✓ *Open the Package Explorer View (Window -> Show View -> Package Explorer) and the Type Checking View (Bad Smells -> Type Checking).*
- ✓ *Import the Java project to be analyzed for Type Checking bad smells and select it on the Package Explorer View.*
- ✓ *From the Type Checking view click on the "Identify Bad Smells" button to run the detection process.*
- ✓ *In the table of the Type Checking view you can see all refactoring opportunities identified for the selected project grouped according to their relevance.*
- ✓ *Each row of the table contains information about:*
- ✓ *the Refactoring Type of the suggestion (Replace Conditional with Polymorhism or Replace Type Code with State/Strategy).*
- ✓ *the Type Checking Method (the method that contains the type checking code fragment).*
- ✓ *the Abstract Method Name (this is the name of the polymorphic method that will be created if the corresponding refactoring is applied. The default name of the polymorphic method is the name of the type checking method).*

- ✓ *the number of System-Level Occurrences (this is the number of relevant suggestions corresponding to the same inheritance hierarchy at system level).*
- ✓ *the number of Class-Level Occurrences (this is the number of relevant suggestions corresponding to the same inheritance hierarchy at class level).*
- ✓ *the Average number of statements per case.*

- ✓ *Double-clicking a row of the table will open both the class where type checking method belongs to and the subclasses (if they exist, in case of Replace Conditional with Polymorphism refactoring) in separate editors of Eclipse.*
- ✓ *The code fragment that contains the type checking will be highlighted.*
- ✓ *The default name of the polymorphic method can be changed by clicking on the "Rename Method" button. The validity of the new name is automatically checked.*
- ✓ *To apply a refactoring select a row from the table and click on the "Apply Refactoring" button.*
- ✓ *After applying or undoing the application of refactoring, you have to press the "Identify Bad Smells" button again to refresh the table.*

Long Method

- ✓ *In order to enable the analysis of methods belonging to library/API classes you should follow the steps shown in this guide.*
- ✓ *Open the Package Explorer View (Window -> Show View -> Package Explorer) and the Long Method View (Bad Smells -> Long Method).*
- ✓ *Import the Java project to be analyzed for Long Method bad smells and select it on the Package Explorer View.*
- ✓ *From the Long Method view click on the "Identify Bad Smells" button to run the detection process.*
- ✓ *In the tree-like table of the Long Method view you can see all refactoring opportunities identified for the selected project.*
- ✓ *Each leaf row of the tree-like table contains information about*:
  - ✓ *the Refactoring Type of the suggestion (Extract Method).*
  - ✓ *the Source Method (the method from which the slice will be extracted).*
  - ✓ *the Variable Criterion (the variable for which the slice has been generated).*
  - ✓ *the Block-Based Region (the region within the method body in which the slice expands).*
  - ✓ *the ratio of Duplicated/Extracted (the percentage of the extracted slice statements that will be duplicated in both the original and the extracted methods).*
- ✓ *Double-clicking a leaf row of the tree-like table will open the class where source method belongs to in a separate editor of Eclipse.*

- ✓ *The statements belonging to the slice will be highlighted. By hovering over the highlighted statements the set of Defined and Used variables are displayed.*
- ✓ *To apply a refactoring select a leaf row from the tree-like table and click on the "Apply Refactoring" button.*
- ✓ *The refactoring wizard allows to specify the name of the method to be extracted (the default name for the extracted method is the name of the variable criterion). The validity of the new name is automatically checked.*
- ✓ *After applying or undoing the application of refactoring, you have to press the "Identify Bad Smells" button again to refresh the table.*

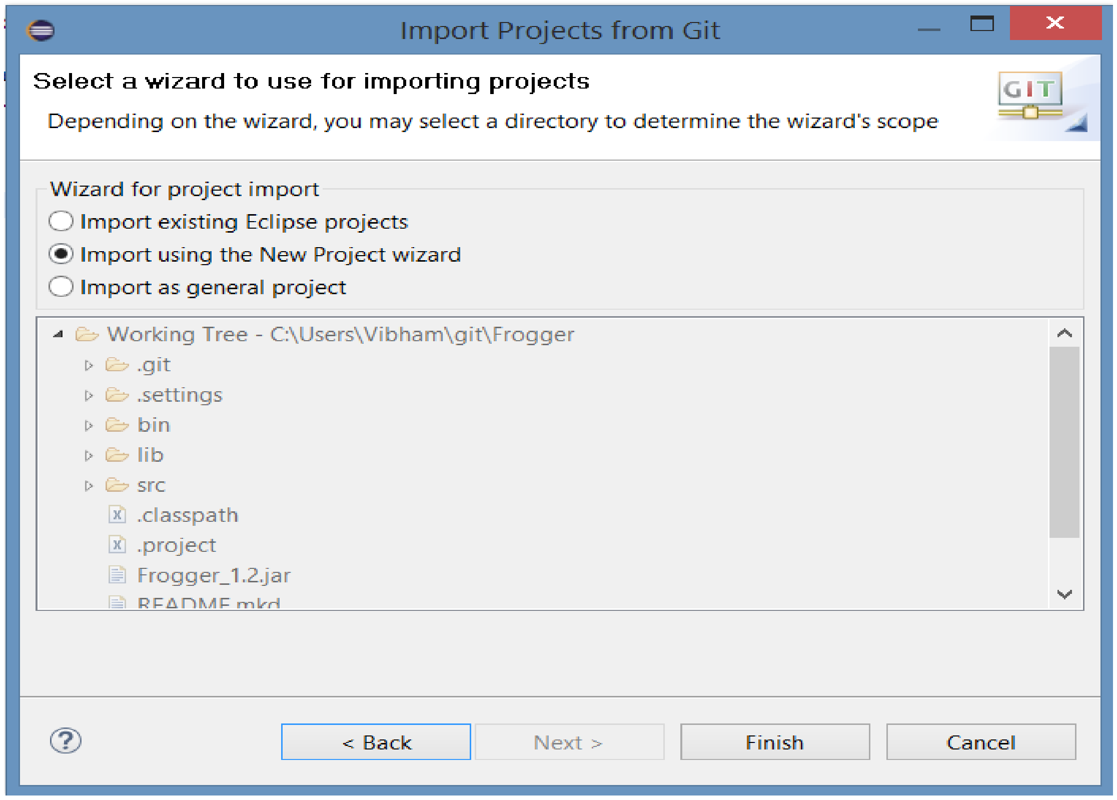
God Class

- ✓ *Open the Package Explorer View (Window -> Show View -> Package Explorer) and the God Class View (Bad Smells -> God Class).*
- ✓ *Import the Java project to be analyzed for God Class bad smells and select it on the Package Explorer View.*
- ✓ *From the God Class view click on the "Identify Bad Smells" button to run the detection process.*
- ✓ *In the tree-like table of the God Class view you can see all refactoring opportunities identified for the selected project.*
- ✓ *Each top-level node in the tree-like table corresponds to a class of the analyzed project. Second-level nodes represent the 'general concepts' found for the given class. The general concepts do not have any common class members.*
- ✓ *Each leaf row of the tree-like (third-level nodes) table contains information about:*
  - *i* *the Refactoring Type of the suggestion (Extract Class).*
  - *ii* *the Source Class (the name of the class for which decompositions are suggested).*
  - *iii* *the Extractable Concept (corresponds to a set of fields and methods from the Source class that can be extracted to a new class in order to decompose the Source class).*
  - *iv* *the value of the Entity Placement metric after the application of the corresponding refactoring.*
- ✓ *Double-clicking a leaf row of the tree-like table will open the class suggested to be decomposed in a separate editor of Eclipse.*
- ✓ *The fields and methods which are suggested to be extracted to a new class will be highlighted.*
- ✓ *To apply a refactoring select a row from the table and click on the "Apply Refactoring" button. This will open a refactoring wizard that allows to specify the name of the new extracted class.*

- ✓ *After applying or undoing the application of refactoring, you have to press the "Identify Bad Smells" button again to refresh the table.*

Removing smells from frogger code from github:

- ✓ *search frogger project from github and copy the url*
- ✓ *Go to window > perspective > open perspective > other >git > clone> paste the url*
- ✓ *Switch again to java perspective.*
- ✓ *Right click on project explorer window> import >select projects from git > select existing local repository > select frogger >check import using the new project wizard.*



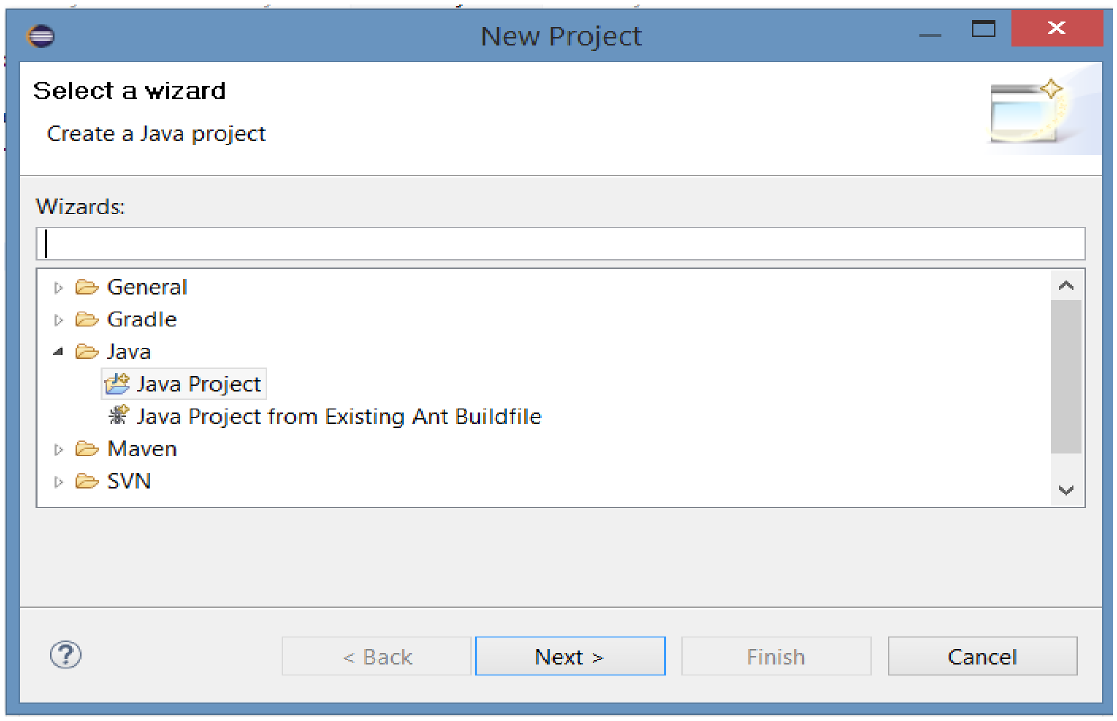
Select a wizard to use for importing projects

Depending on the wizard, you may select a directory to determine the wizard's scope

Wizard for project import
- Import existing Eclipse projects
- [✓ Import using the New Project wizard]
- Import as general project

```
Working Tree - C:\Users\Vibham\git\Frogger
  .git
  .settings
  bin
  lib
  src
  .classpath
  .project
  Frogger_1.2.jar
  README.mkd
```

click finish> java project


New Project

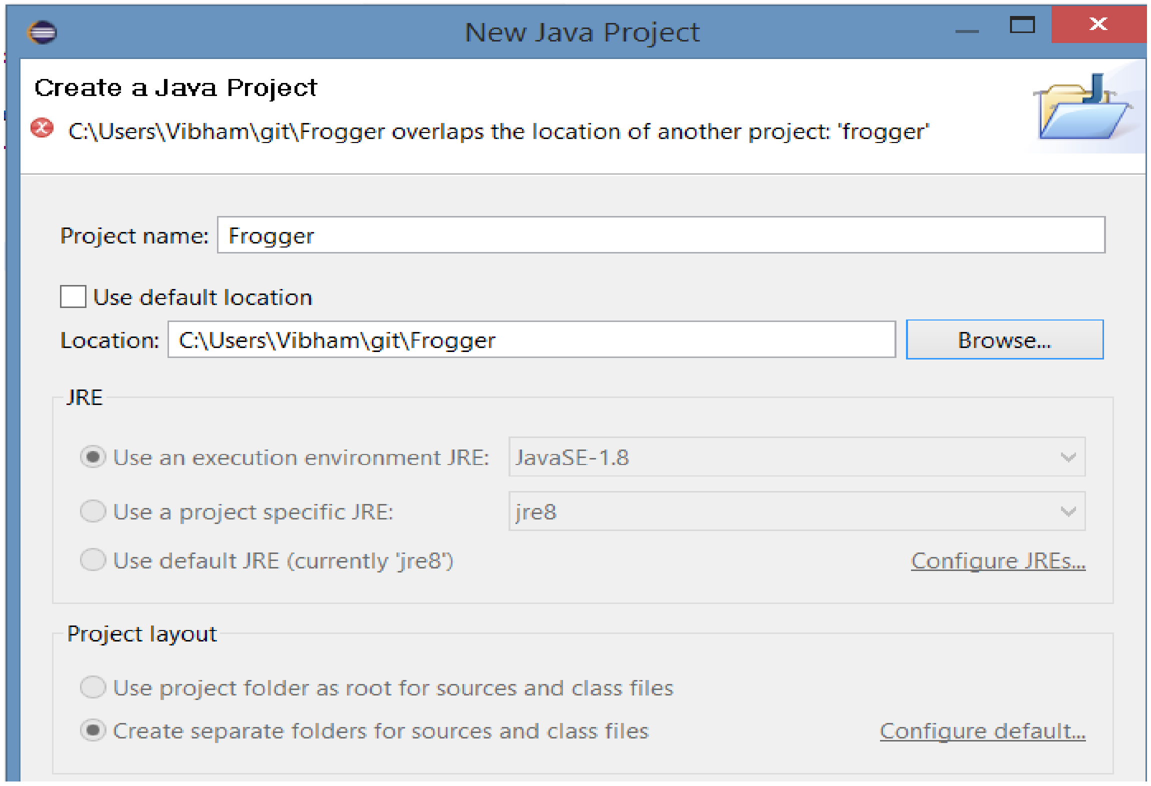
Select a wizard

Create a Java project

Wizards:

- General
- Gradle
- Java
  - Java Project
  - Java Project from Existing Ant Buildfile
- Maven
- SVN

give project name > iuncheck default location > browse to git / frogger>finish


New Java Project

Create a Java Project

C:\Users\Vibham\git\Frogger overlaps the location of another project: 'frogger'

Project name: Frogger

[ ] Use default location

Location: C:\Users\Vibham\git\Frogger

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre8

( ) Use default JRE (currently 'jre8')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

5. Click Bad Smells> select different smells and apply refactoring on frogger project.

LAB- 8

Aim: Introduction to Build Tools –ANT (Another Neat Tool).

Description: ANT stands for Another Neat Tool. It is a Java-based build tool from Apache.

On an average, a developer spends a substantial amount of time doing mundane tasks like build and deployment that include:

- ✓ *Compiling the code*
- ✓ *Packaging the binaries*
- ✓ *Deploying the binaries to the test server*
- ✓ *Testing the changes*
- ✓ *Copying the code from one location to another*

To automate and simplify the above tasks, Apache Ant is useful. It is an Operating System build and deployment tool that can be executed from the command line.

Installing steps:

- ✓ *Download the binaries from http://ant.apache.org*
- ✓ *Unzip the zip file to a convenient location c:\folder. using Winzip, winRAR, 7-zip or similar tools.*
- ✓ *Create a new environment variable called **ANT_HOME** that points to the Ant installation folder, in this case **C:\apache-ant-1.10.1-bin** folder.*
- ✓ *Append the path to the Apache Ant batch file to the PATH environment variable. In our case this would be the **C:\apache-ant-1.10.1-bin\bin** folder.*

Building HelloWorld project:

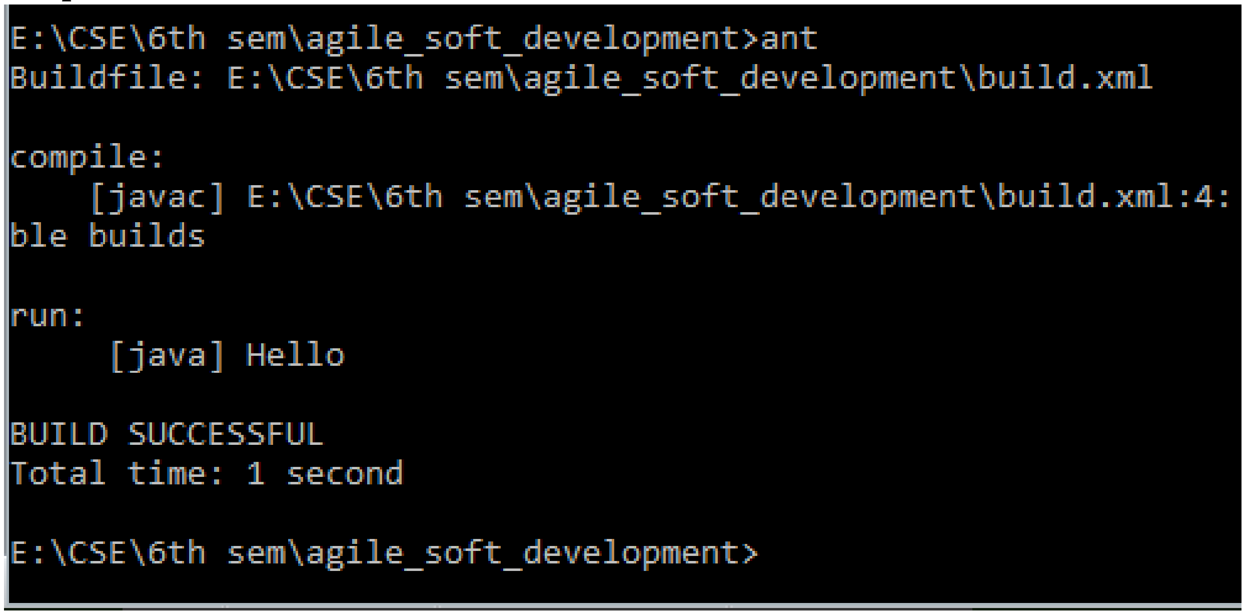
HelloWorld program-

```
public class hello_world
{
        public static void main(String[] args)
        {
                System.out.println("Hello");
        }
}
```

Build.xml file-

```
<?xml version="1.0" encoding="UTF-8" ?>
        <project name="first" default="run">
                <target name="compile">
                        <javac srcdir="." destdir="."/>
                </target>
                <target name="run" depends="compile">
                        <java classname="hello_world" fork="true"/>
                </target>
        </project>
```

Output:

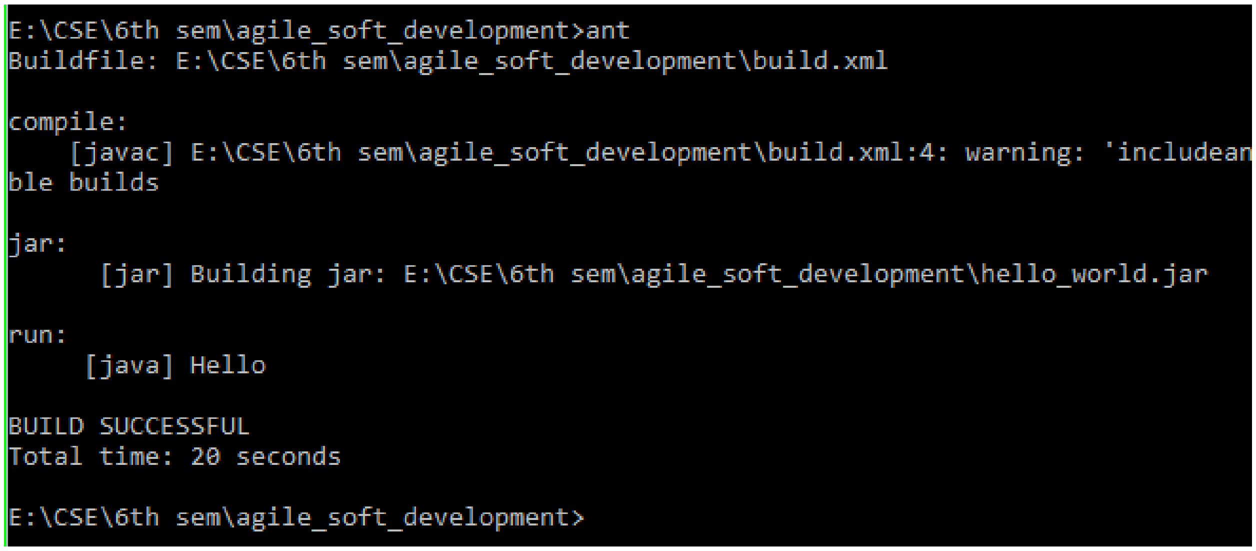


```
E:\CSE\6th sem\agile_soft_development>ant
Buildfile: E:\CSE\6th sem\agile_soft_development\build.xml

compile:
    [javac] E:\CSE\6th sem\agile_soft_development\build.xml:4:
ble builds

run:
     [java] Hello

BUILD SUCCESSFUL
Total time: 1 second

E:\CSE\6th sem\agile_soft_development>
```

Creating Jar file:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <project name="first" default="run">
        <target name="compile">
                <javac srcdir="." destdir="."/>
        </target>
        <target name="jar" depends="compile">
        <jar destfile="hello_world.jar" basedir=".">
                <manifest>
                        <attribute name="Main-Class" value="${main-class}"/>
                </manifest>
        </jar>
        </target>
        <target name="run" depends="jar">
                <java classname="hello_world" fork="true"/>
        </target>
    </project>
```

Output:



```
E:\CSE\6th sem\agile_soft_development>ant
Buildfile: E:\CSE\6th sem\agile_soft_development\build.xml

compile:
    [javac] E:\CSE\6th sem\agile_soft_development\build.xml:4: warning: 'includean
ble builds

jar:
      [jar] Building jar: E:\CSE\6th sem\agile_soft_development\hello_world.jar

run:
     [java] Hello

BUILD SUCCESSFUL
Total time: 20 seconds

E:\CSE\6th sem\agile_soft_development>
```

LAB- 9

Aim: *Execute continuous integration using a tool such as Jenkins*

Description:

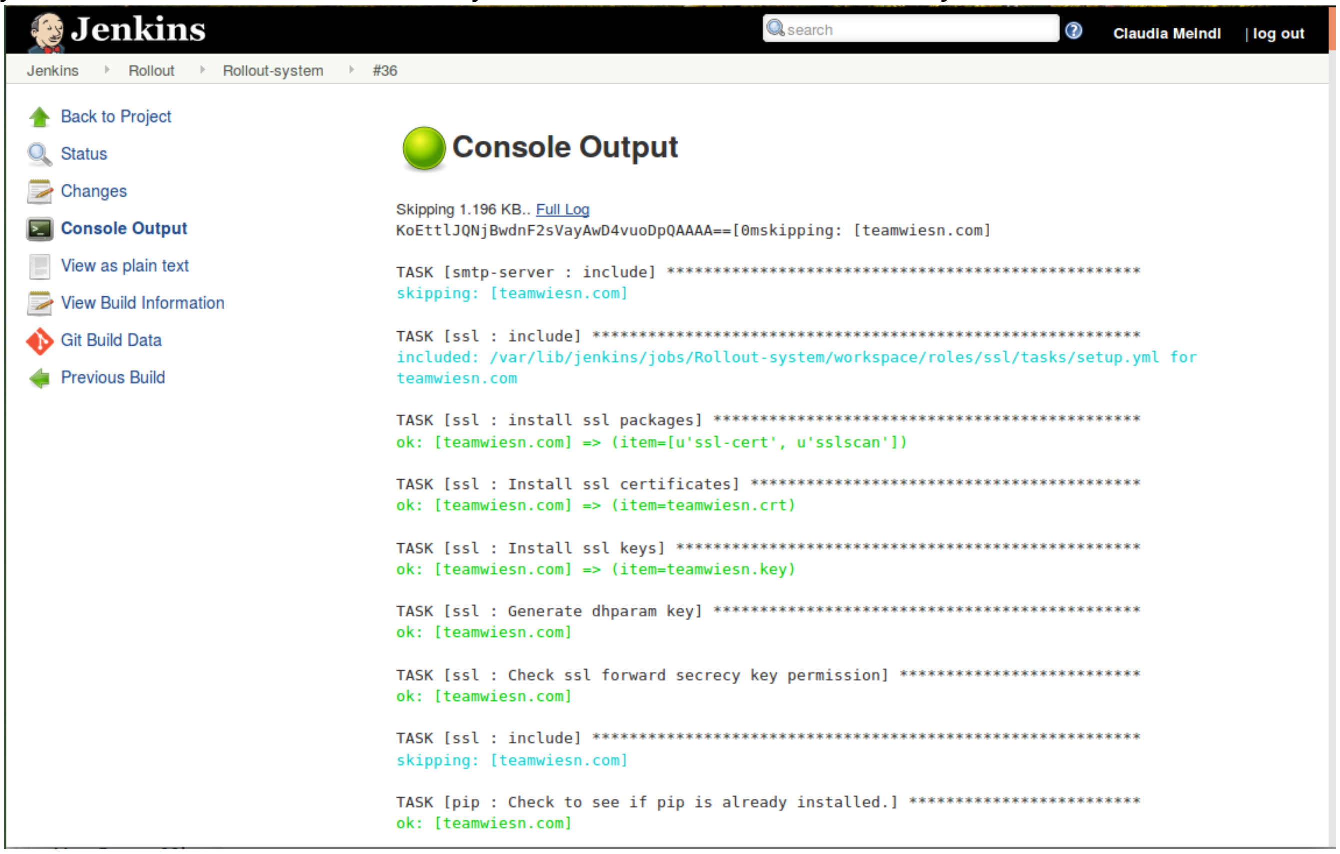
Continuous integration is a process in which all development work is integrated as early as possible. The resulting artifacts are automatically created and tested. This process should identify errors as very early in the process.

Jenkins is one open source tool to perform continuous integration and build automation. The basic functionality of Jenkins is to execute a predefined list of steps. The trigger for this execution can be time or event based. For example, every 20 minutes or after a new commit in a Git repository.

The list of steps can, for example, include:

- ✓ *perform a software build with Apache Maven or Gradle*
- ✓ *Run a shell script*
- ✓ *Archive the build result*
- ✓ *Afterwards start the integration tests*

Jenkins also monitors the execution of the steps and allows to stop the process if one of the steps fails. Jenkins can also send out notification about the build success or failure.



Skipping 1.196 KB.. Full Log

```
KoEttlJQNjBwdnF2sVayAwD4vuoDpQAAAA==[0mskipping: [teamwiesn.com]

TASK [smtp-server : include] ***************************************************
skipping: [teamwiesn.com]

TASK [ssl : include] ***********************************************************
included: /var/lib/jenkins/jobs/Rollout-system/workspace/roles/ssl/tasks/setup.yml for
teamwiesn.com

TASK [ssl : install ssl packages] **********************************************
ok: [teamwiesn.com] => (item=[u'ssl-cert', u'sslscan'])

TASK [ssl : Install ssl certificates] ******************************************
ok: [teamwiesn.com] => (item=teamwiesn.crt)

TASK [ssl : Install ssl keys] **************************************************
ok: [teamwiesn.com] => (item=teamwiesn.key)

TASK [ssl : Generate dhparam key] **********************************************
ok: [teamwiesn.com]

TASK [ssl : Check ssl forward secrecy key permission] **************************
ok: [teamwiesn.com]

TASK [ssl : include] ***********************************************************
skipping: [teamwiesn.com]

TASK [pip : Check to see if pip is already installed.] *************************
ok: [teamwiesn.com]
```