*TestClass.java

CSCI 5828 Foundations of Software Engineering

PRESENTATION - 1

ECLIPSE IDE

By –
Praveen Kumar Devaraj (prde1873)
Sharath Vontari (shvo9824)

ECLIPSE IDE

INTRODUCTION

- What is IDE - **Integrated Development Environment** - Tool with features required for software development.
- Eclipse is one such open source IDE used for developing softwares.
- Can be used to develop software in C, C++,java, ABAP, COBOL, Python, Ruby etc.
- Provides an open platform for application development tools run on a wide range of operating systems.
- Can be used in any phase of development like Design, Development, Testing etc.
- Provides features for seamless integration of software modules developed individually.
- First release of the eclipse project was made in 2001 and has been growing since then.
- Eclipse created by OTI and IBM teams responsible for IDE products
  - IBM VisualAge/Smalltalk (Smalltalk IDE)
  - IBM VisualAge/Java (Java IDE)
  - IBM VisualAge/Micro Edition (Java IDE)

Fig 1

What is Eclipse

- Eclipse is a universal platform for integrating development tools.

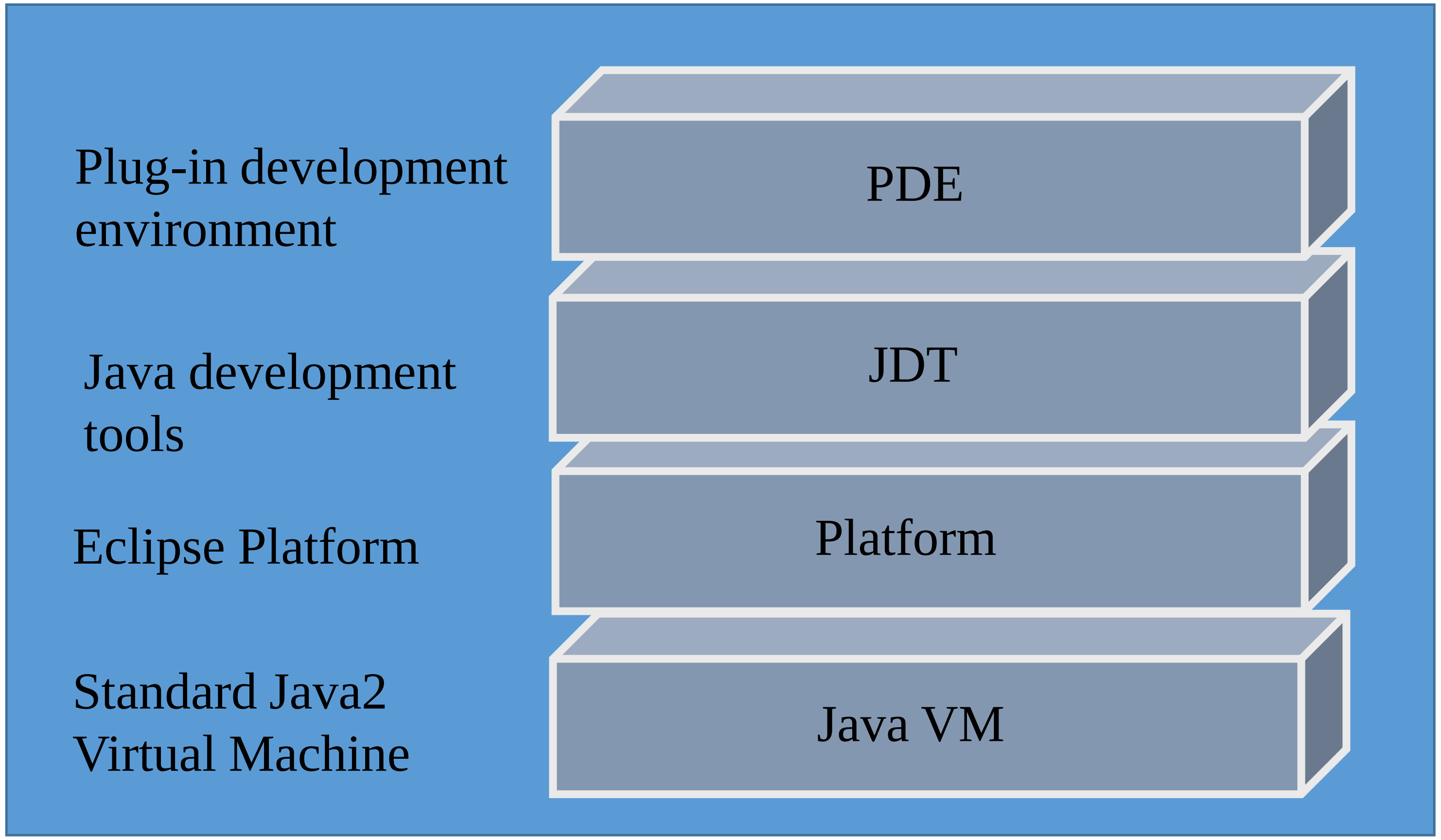
Eclipse Architecture

Plug-in - smallest unit of Eclipse function
Example – CDT, ObjectAid etc.

JDT - provides the tool plug-ins that implement a Java IDE supporting the development of any Java application.

Eclipse Platform - common base

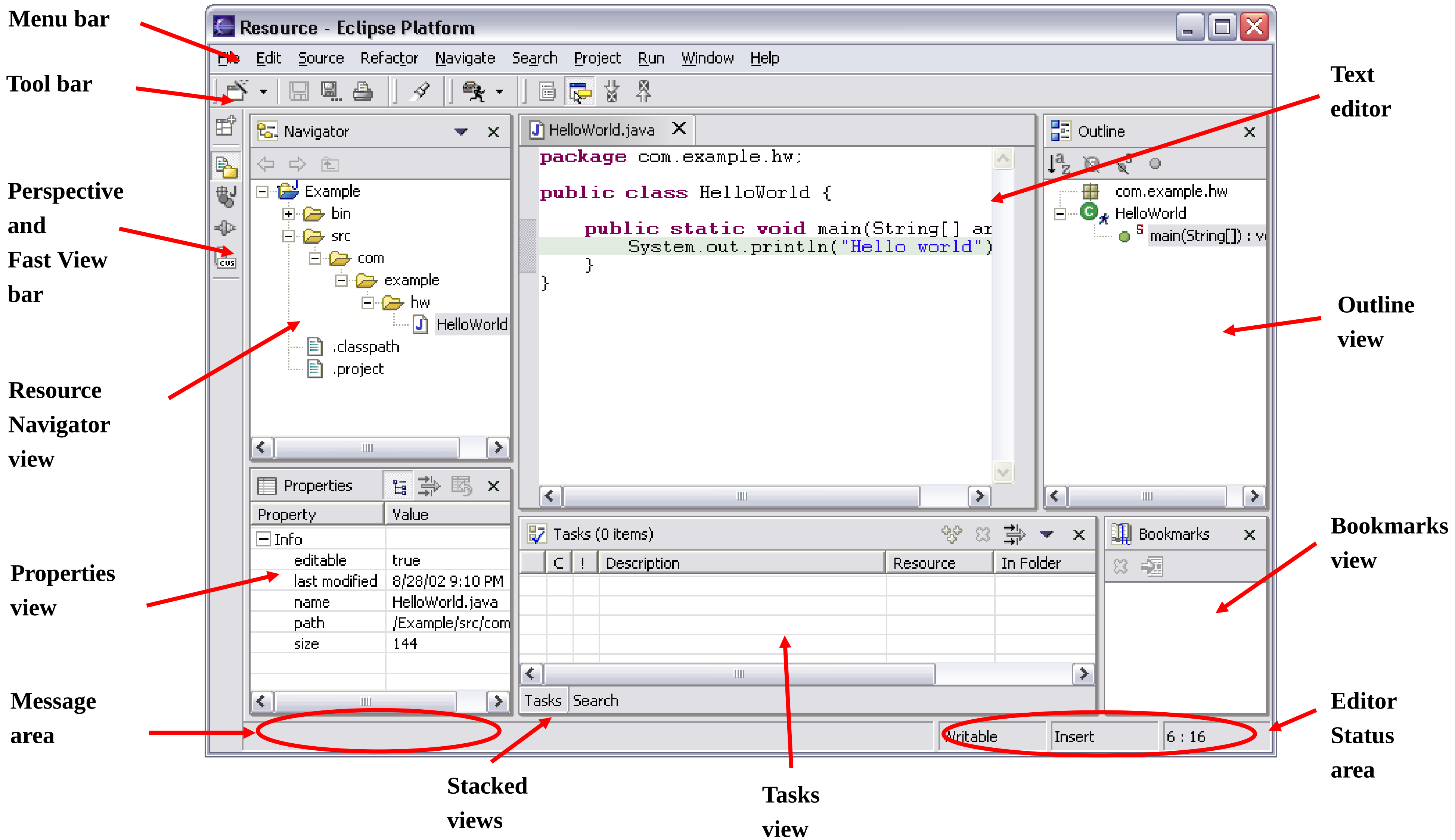
JVM - an abstract computing machine that enables a computer to run a Java program



Plug-in development environment
PDE
Java development tools
JDT
Eclipse Platform
Platform
Standard Java2 Virtual Machine
Java VM

Eclipse User-Interface

User Interface of Eclipse can be changed according to the user preferences. This is a typical UI provided.



```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

Why Eclipse

Several reasons to use Eclipse -

- Open Source – Free of Cost.
- Supports most of the programming languages with appropriate plugins.
- Single tool that can be used for design, development, testing etc.
- User Friendly (Intuitive User Interface).
- Can be integrated with several Version management (Ex: Perforce etc) and build softwares(Ex: ANT).
- Available on most of the platforms.

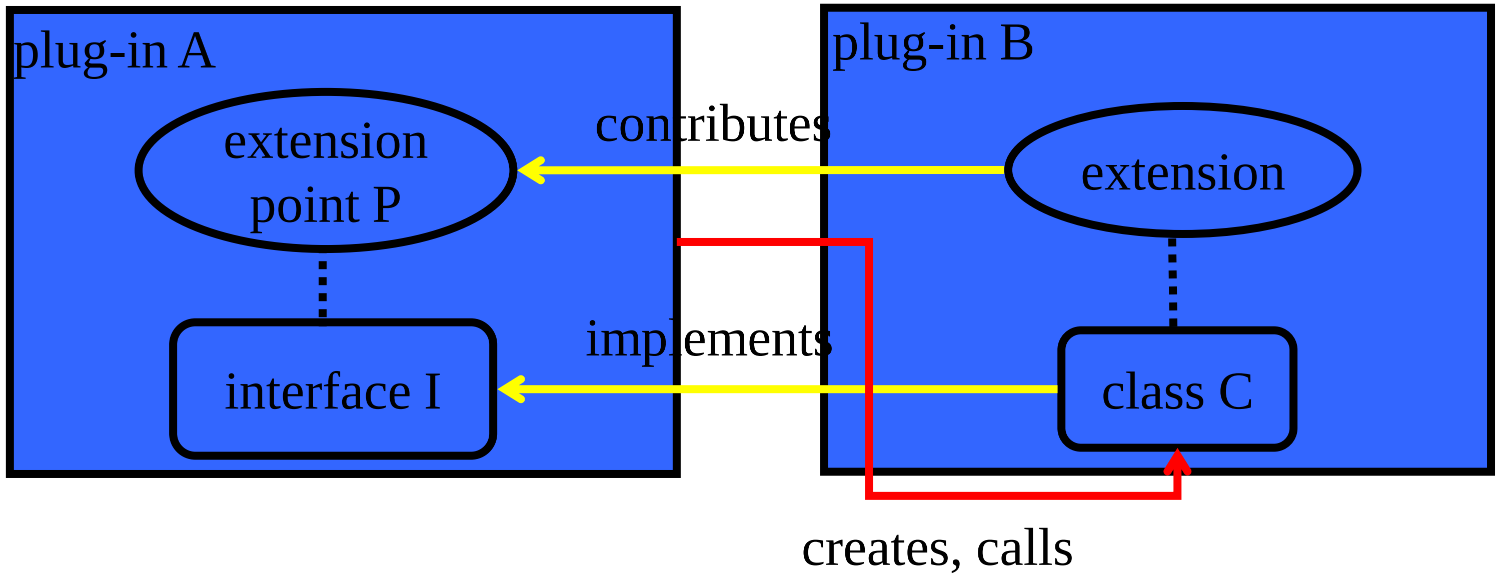
Getting Eclipse

- Download the latest version from here.
- You may need to install Java SDK1.6 or JRE if you haven’t from here.
- After installation the path variable is to be set to the bin folder of java installation.
- Unpack the package and run eclipse.exe in windows or eclipse in Linux.

Plugins

- **Plug-in -** smallest unit of Eclipse function

Typical arrangement in Plugin Architecture



- Plug-in A
  - Declares extension point P
  - Declares interface I to go with P
- Plug-in B
  - Implements interface I with its own class C
  - Contributes class C to extension point P
- Plug-in A instantiates C and calls its I methods

- **Extension point -** named entity for collecting “contributions” Example: extension point for workbench preference UI
- Specified in the manifest.xml file of every plugin

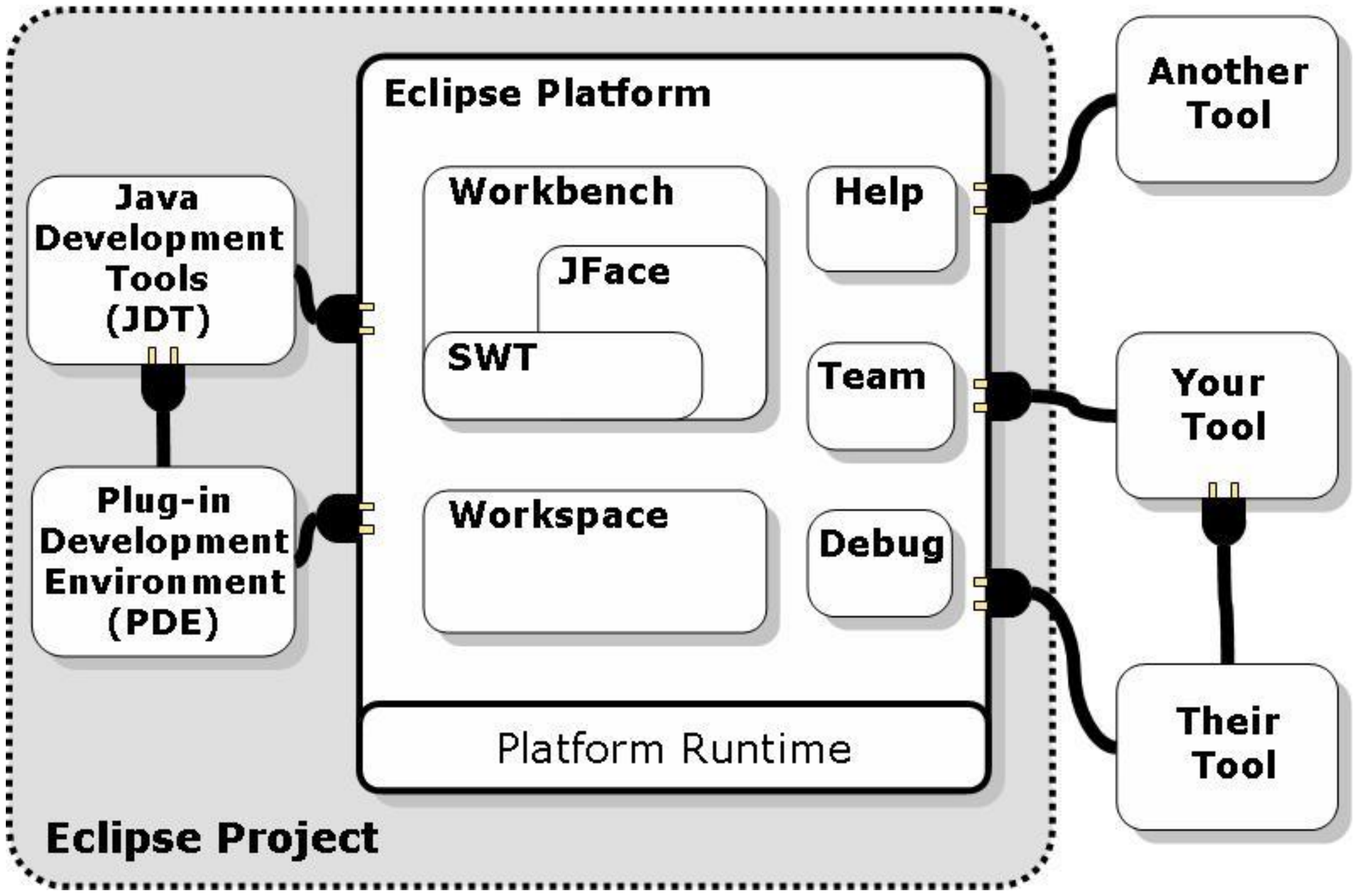
Plugin Installation

- Plugins are mostly in java and can be installed either directly from the repository using the url or by downloading the package and importing the same.
- To learn how to install the plugin directly from the repository check here

Eclipse Runtime

- The Eclipse runtime defines the plug-ins (org.eclipse.osgi and org.eclipse.core.runtime) on which all other plug-ins depend and is responsible for defining a structure for plug-ins and the implementation details of the same.
- The runtime is also responsible for finding and executing the main Eclipse application and for maintaining a registry of plug-ins, their extensions, and extension points.
- The runtime also provides utilities, such as logging, debug trace options, adapters, a preference store, and a concurrency infrastructure.

Eclipse Platform Architecture Overview



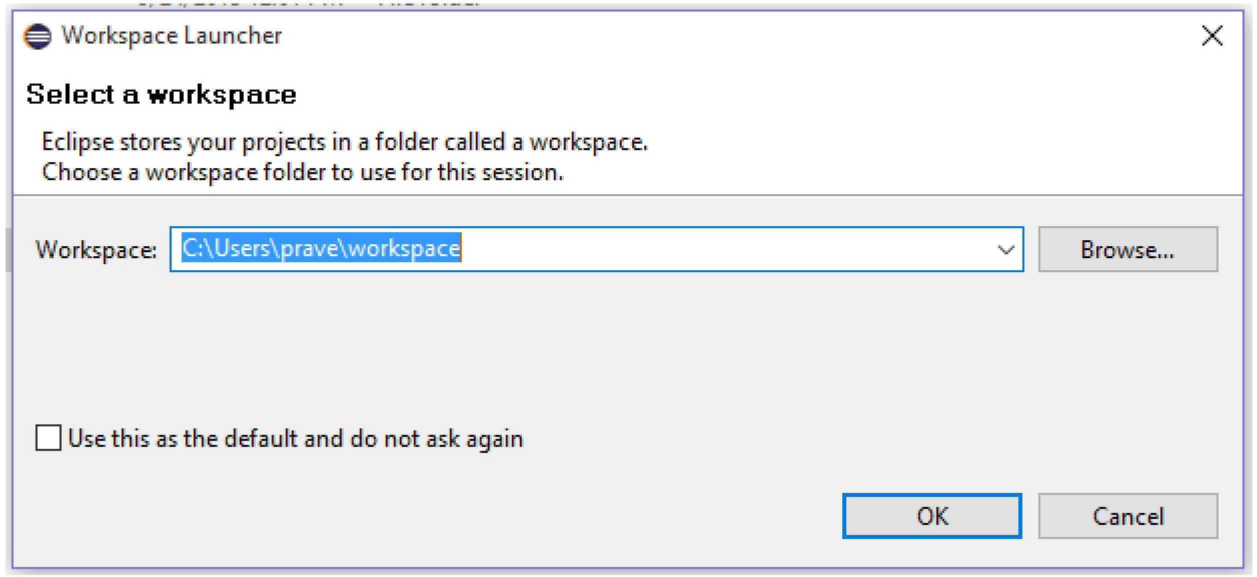
SWT – Standard widget Toolkit - generic low-level graphics and widget set.

JFace -UI frameworks for common UI tasks

Tools here refer to plugins.

GETTING STARTED WITH ECLIPSE DEVELOPMENT

- Create a Workspace for saving all the data handled and modified in eclipse.
- On start up a pop up appears requesting the path for the workspace as shown below. Enter and Click OK.



Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

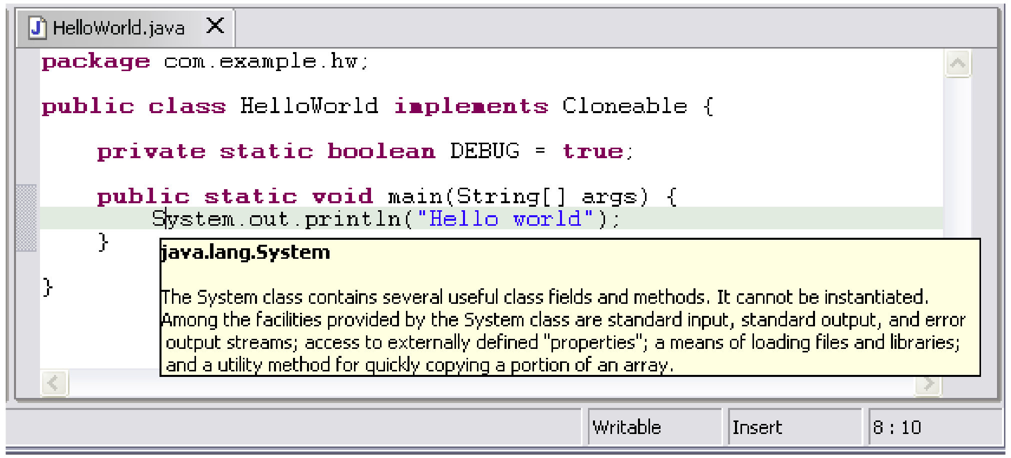
Workspace: C:\Users\prave\workspace

☐ Use this as the default and do not ask again

- Workspace is the directory used to store the source code and other data on which the user is working on in Eclipse.
- This dialog appears only if the java path variable is configured correctly else it does not launch the IDE. To configure the path variable check here.
- Eclipse can be used for design, development, testing etc. Lets consider them individually later.

Features of Eclipse Java Editor

- Hovering over identifier shows Javadoc spec (Fig1).
- Method completion and suggestions to keywords (Fig2).
- Spell Check
- Java editor creates stub methods
- Local method history
- Code formatter
- Source code for binary libraries
- Built-in refactoring etc



```
package com.example.hw;

public class HelloWorld implements Cloneable {

    private static boolean DEBUG = true;

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

}
```

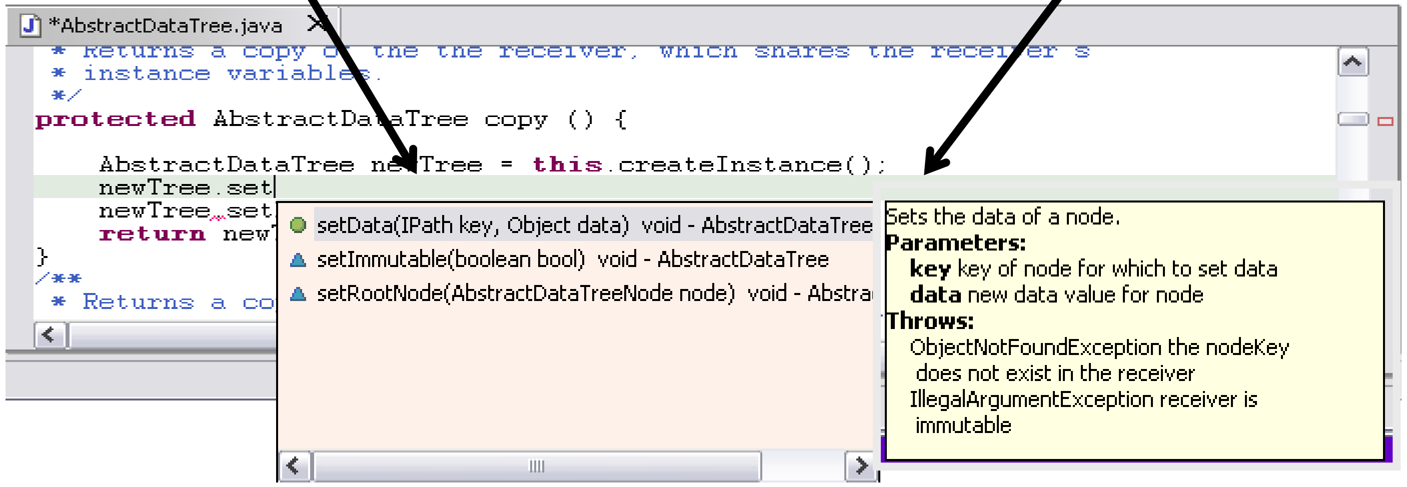
java.lang.System

The System class contains several useful class fields and methods. It cannot be instantiated. Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined "properties"; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

Fig 1

List of plausible methods

Doc for method



```
 * Returns a copy of the the receiver, which shares the receiver's
 * instance variables.
 */
protected AbstractDataTree copy () {

    AbstractDataTree newTree = this.createInstance();
    newTree.set
    newTree.set
    return new
}
/**
 * Returns a co
```

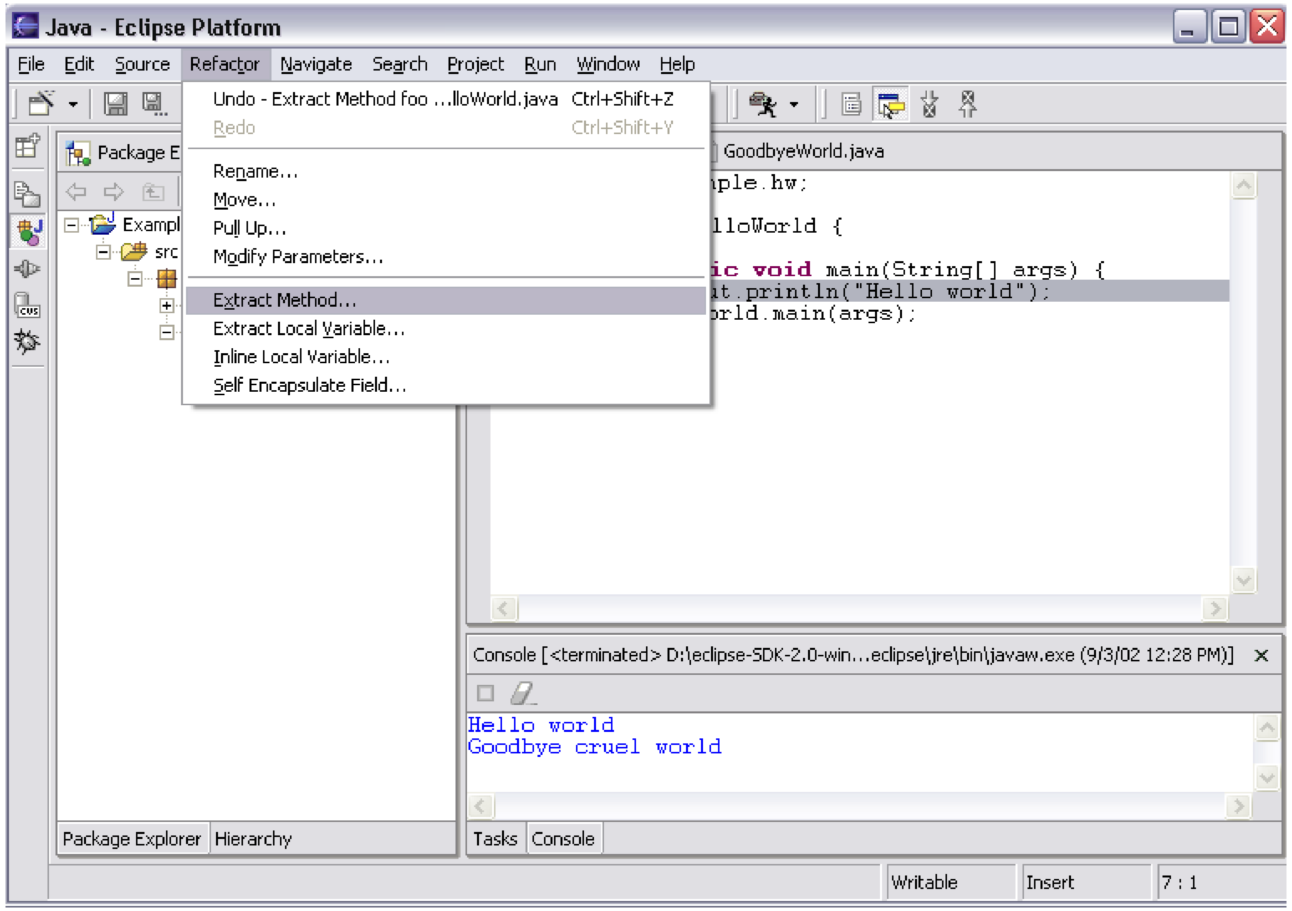
- setData(IPath key, Object data) void - AbstractDataTree
- setImmutable(boolean bool) void - AbstractDataTree
- setRootNode(AbstractDataTreeNode node) void - Abstra

Sets the data of a node.
Parameters:
key key of node for which to set data
data new data value for node
Throws:
ObjectNotFoundException the nodeKey does not exist in the receiver
IllegalArgumentException receiver is immutable

Fig 2

Code Refactoring in Eclipse

- JDT has actions for refactoring Java code
- Refactoring actions rewrite source code
  - Within a single Java source file
  - Across multiple interrelated Java source files
- Refactoring actions preserve program semantics
  - Does not alter what program does
  - Just affects the way it does it
- Refactoring actions rewrite source code
  - Within a single Java source file
  - Across multiple interrelated Java source files
- Refactoring actions include
  - Move
  - Rename
  - Organize Imports
  - Extract Methods etc



```
ple.hw;

lloWorld {

ic void main(String[] args) {
t.println("Hello world");
orld.main(args);
```

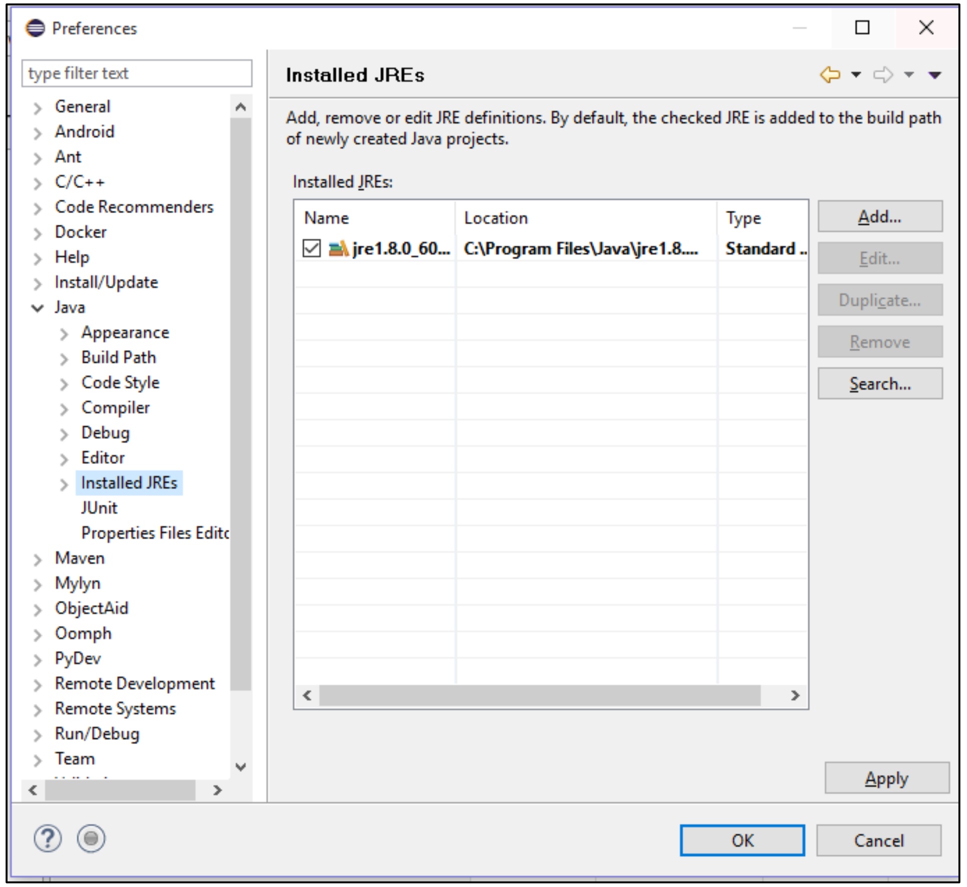
Console [<terminated> D:\eclipse-SDK-2.0-win...eclipse\jre\bin\javaw.exe (9/3/02 12:28 PM)]

```
Hello world
Goodbye cruel world
```

ECLIPSE IN DEVELOPMENT

- Eclipse can be used to develop software modules in various languages. Languages that are supported in Eclipse include C, C++,java, ABAP, COBOL, Python, Ruby, Fortran, Birt etc.
- Let us consider Java for demonstrating the Eclipse development environment.

- Before getting started verify the Java runtime Environment.
- Select Windows in Menu Bar > preferences >Java >Installed JRE. Confirm that a JRE has been detected.



Installed JREs

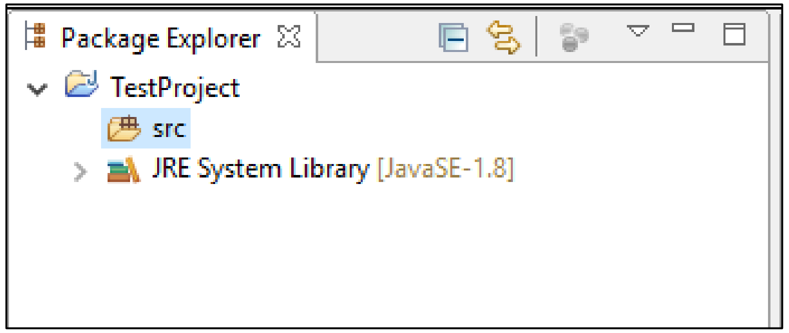
Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

Name	Location	Type
☑ jre1.8.0_60...	C:\Program Files\Java\jre1.8....	Standard ..

- Launch Eclipse and select the menu item **File > New > Project….** to open the **New Project** wizard

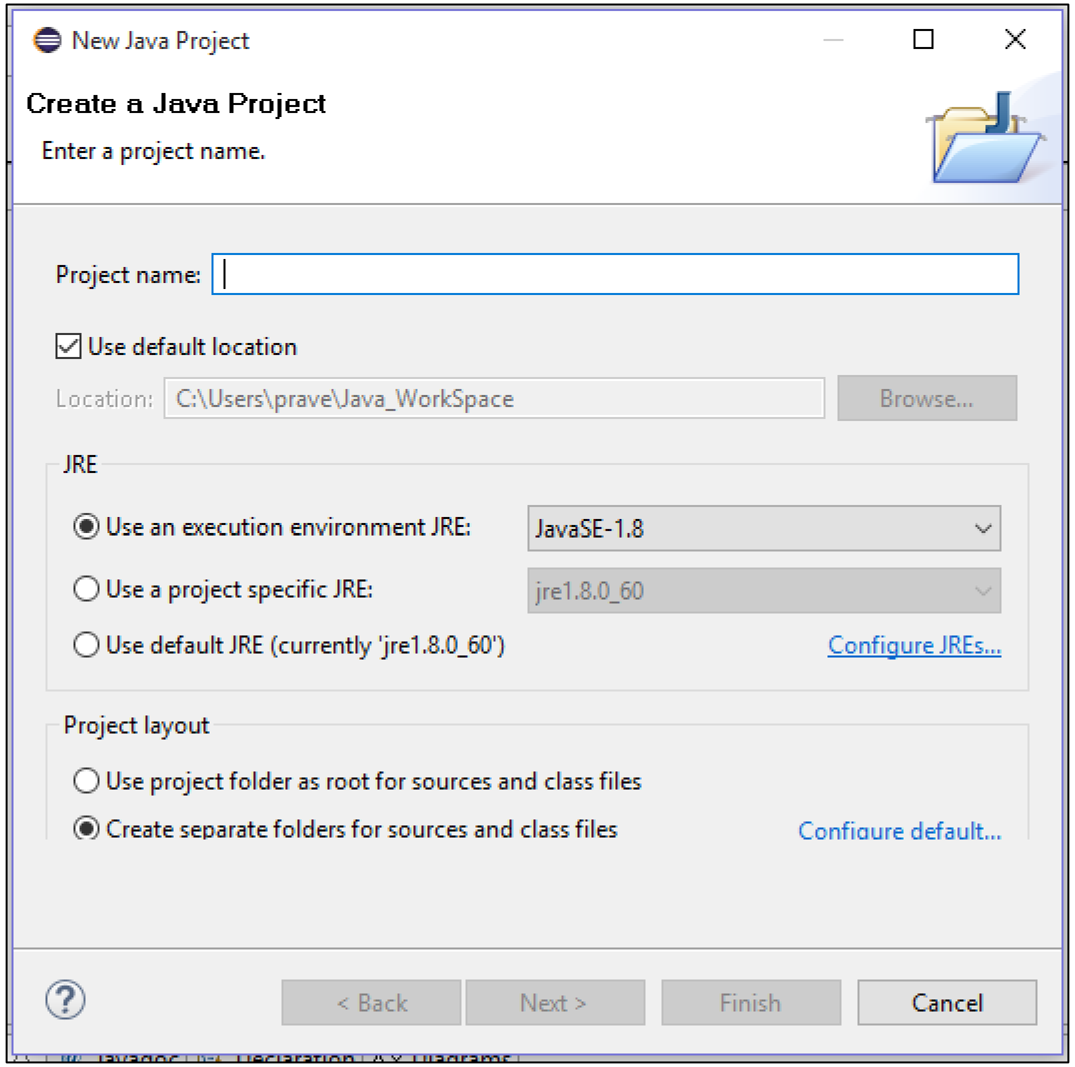
- Enter the project Name and select the version of JRE to be used and click on Finish
- After creating the project this can be seen in the project explored as shown in fig2.



TestProject
src
JRE System Library [JavaSE-1.8]

Fig 2

- In the project explorer only a source directory and an instance of JRE is created.



New Java Project

Create a Java Project

Enter a project name.

Project name:

[✓] Use default location

Location: C:\Users\prave\Java_WorkSpace

JRE

[● Use an execution environment JRE:] JavaSE-1.8

○ Use a project specific JRE: jre1.8.0_60

○ Use default JRE (currently 'jre1.8.0_60')

Project layout

○ Use project folder as root for sources and class files

[● Create separate folders for sources and class files]

Fig 1

- After creating the java project create a class in the source folder. Go to source package > New > Class.
- A dialog box will pop up to help create the class file. Enter the class name, package name and the stubs to be created. Click Finish.(shown in fig 2).
- The class file is created with empty stubs in it.

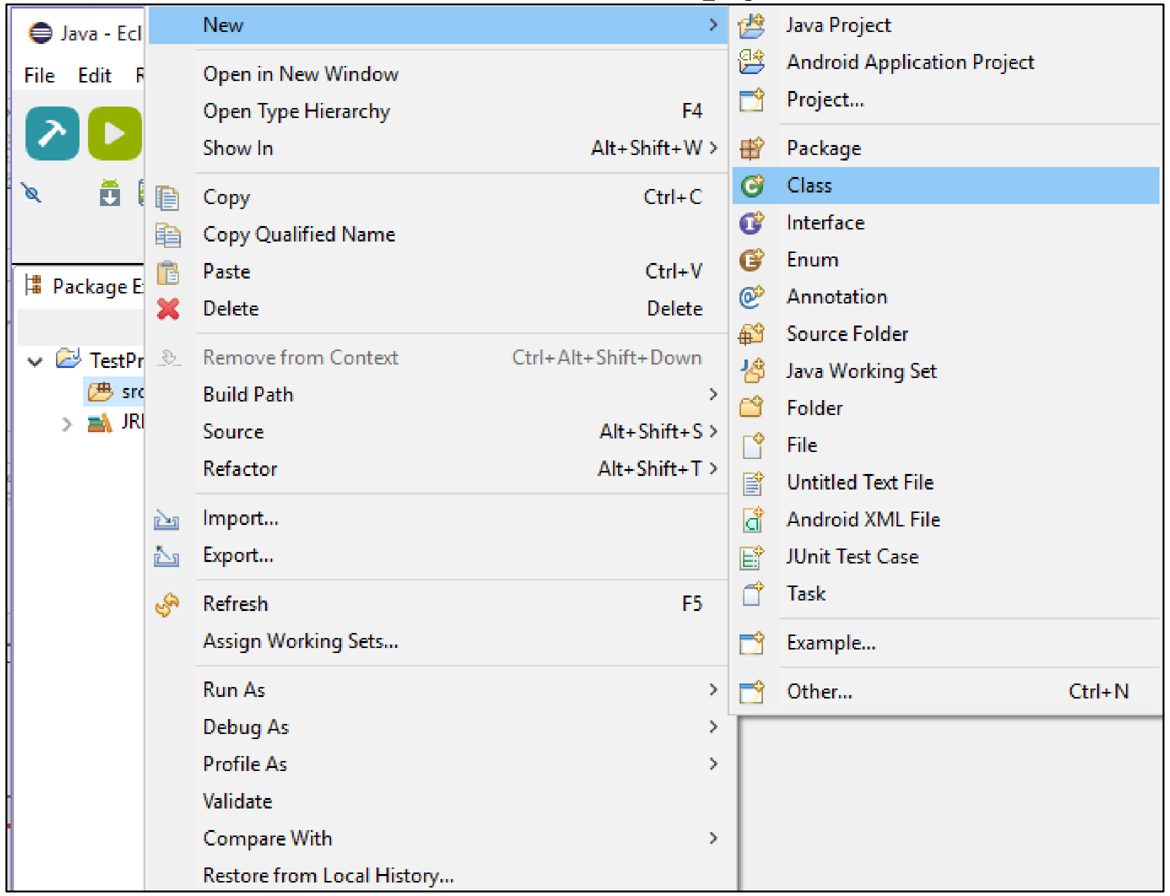
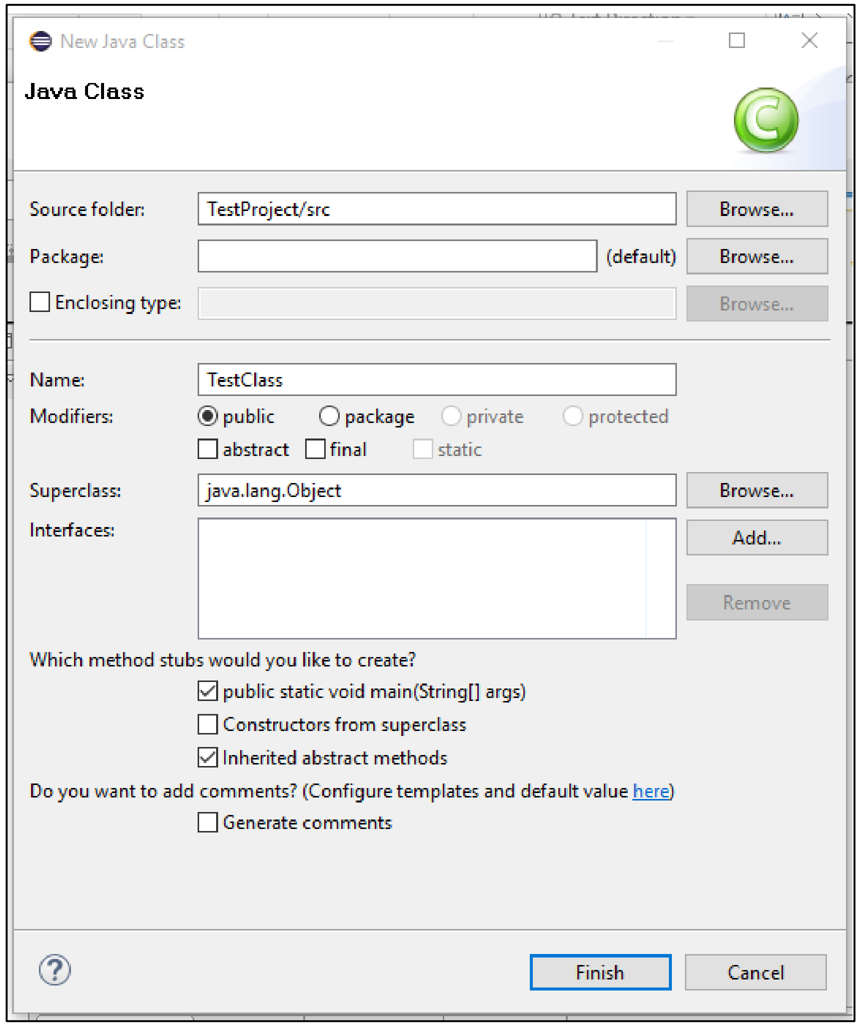


Fig 1



New Java Class

Java Class

Source folder: TestProject/src

Package: (default)

☐ Enclosing type:

Name: TestClass

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

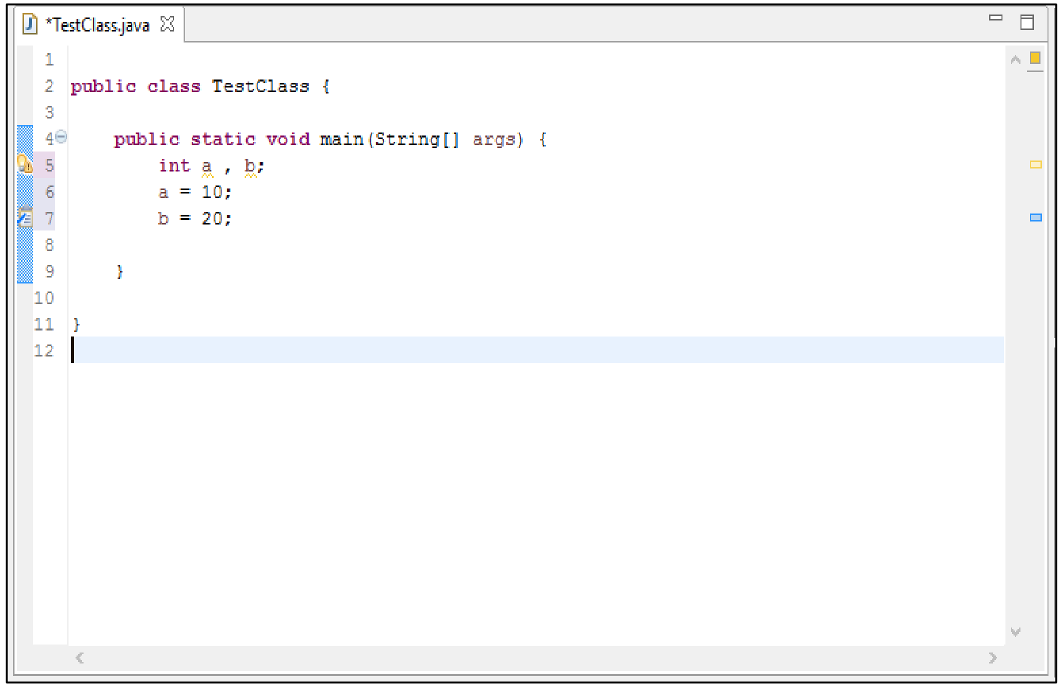
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Fig 2

- Include the source code in the file using the editor as show in fig 1. Once the source code is completed build the project by right clicking on the package and selecting build project as shown in Fig 2.



*TestClass.java

```
public class TestClass {

    public static void main(String[] args) {
        int a , b;
        a = 10;
        b = 20;

    }

}
```

Fig 1

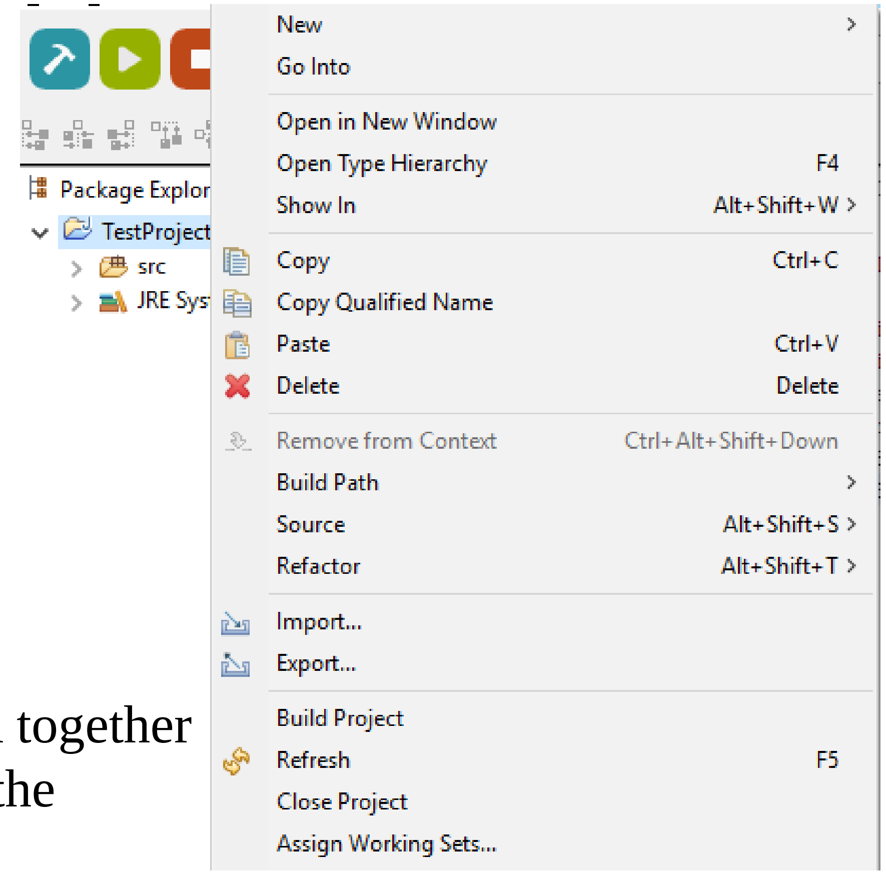
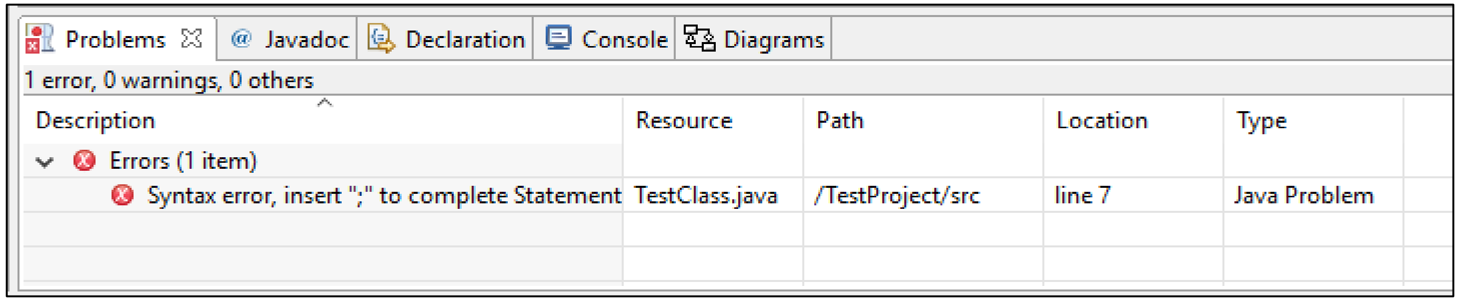


Fig 2

- Build compiles all of the source files to object files, then links them together into one executable. When there is any syntax errors it is shown in the problems view of eclipse as shown below in fig 3.



Problems | Javadoc | Declaration | Console | Diagrams

1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
Syntax error, insert ";" to complete Statement	TestClass.java	/TestProject/src	line 7	Java Problem

Fig 3

- Another option in Eclipse is the build automatically option where every time the code is saved the IDE compiles source code. To check this option Click on Project option in Menu bar and select build automatically as shown in fig 1.
- We can also reuse the existing code by importing the same into the workspace. For this Right Click on the package and select import. This will redirect you through a dialog box where the kind of file imported and the path of the file is specified as shown in fig 2 and fig 3.

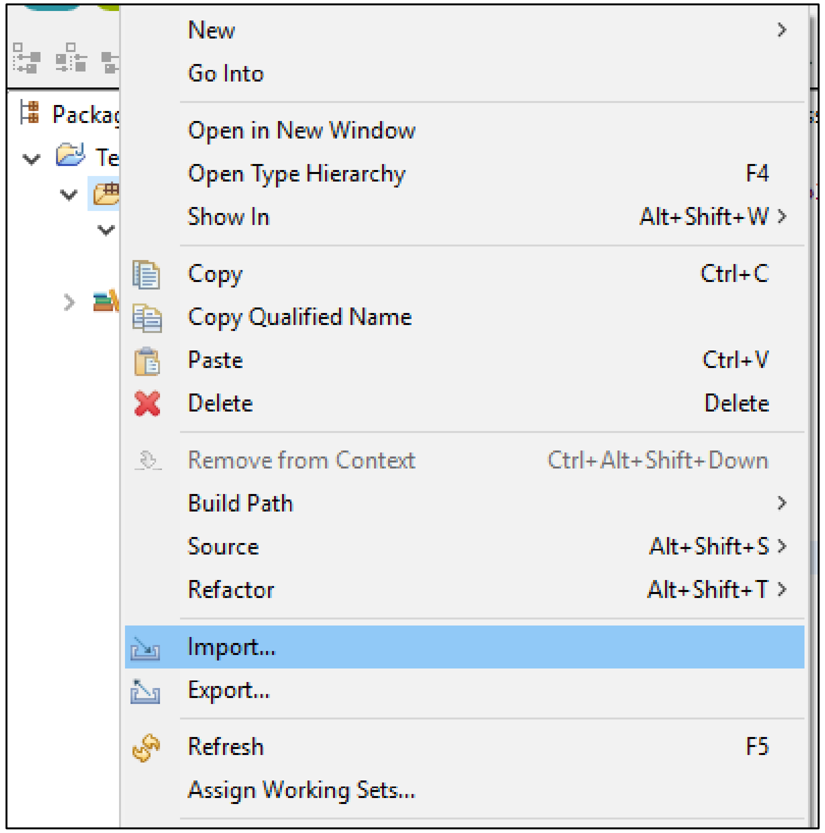


Fig 2

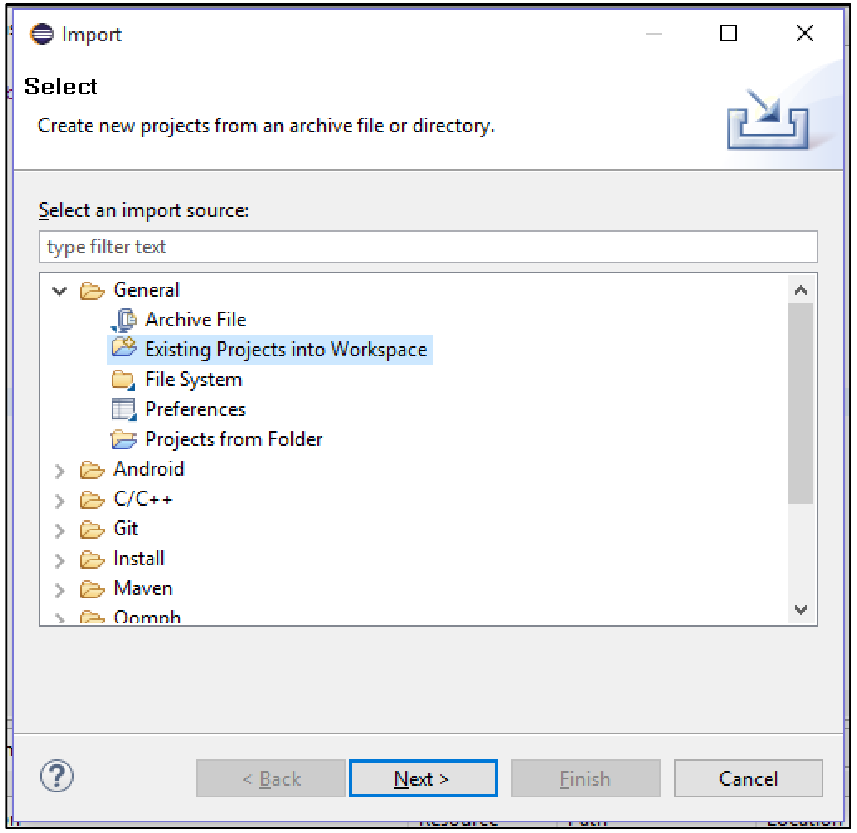


Fig 3

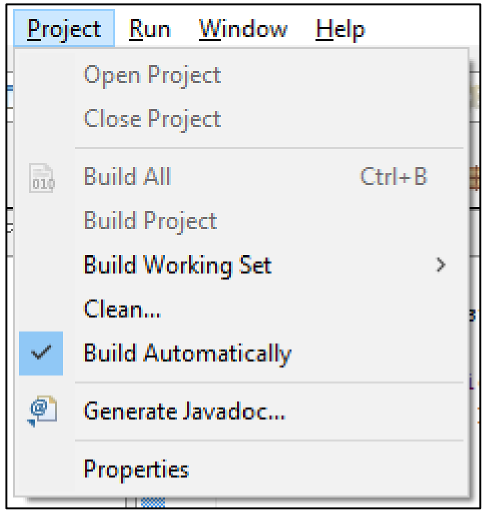


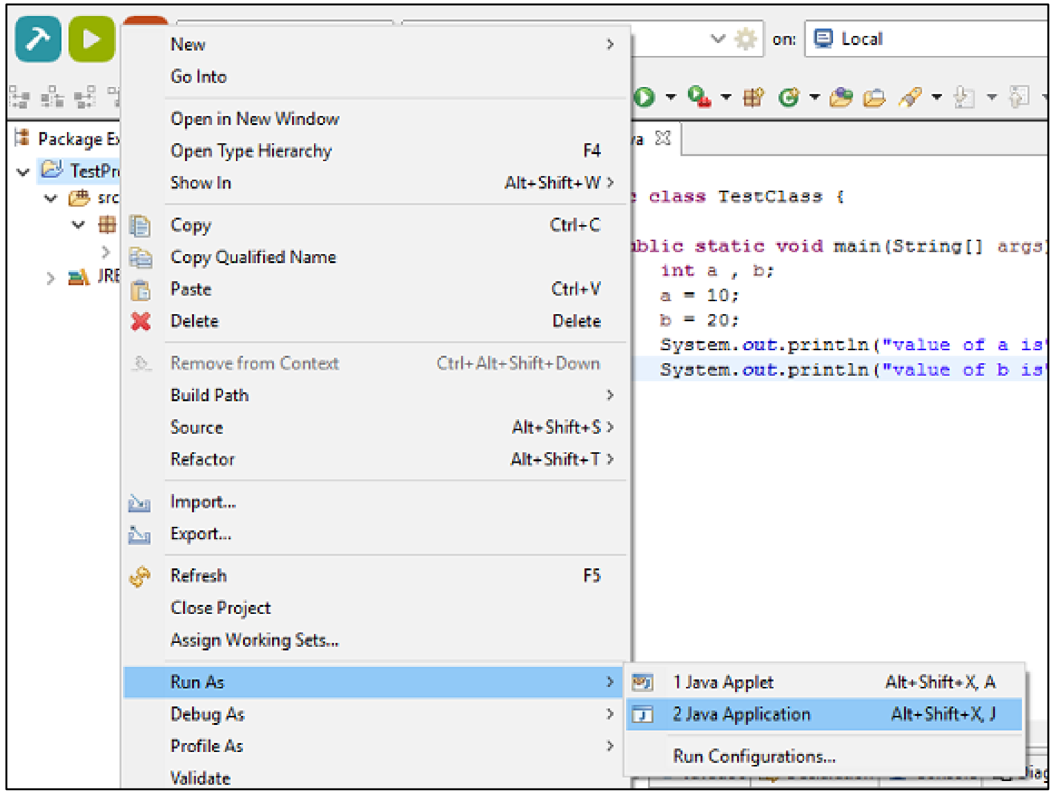
Fig 1

- To run the project Right click on the project > Run as > Run as Java Application as shown in fig 1. The source code is executed and the results is displayed in the console view of eclipse as shown in fig 2.

- Idea case if the program executes correctly the output is displayed in the console view. But when it fails, Eclipse offers a GUI based debugger which helps debugging the programs line by line.

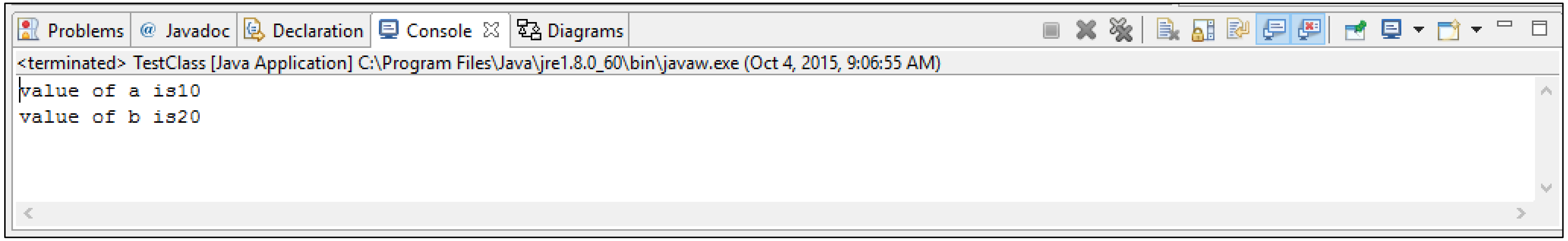
- Open Debug perspective to proceed with debugging.

  Go to Perspective > open Perspective and select debug option.



```
class TestClass {
blic static void main(String[] args
    int a , b;
    a = 10;
    b = 20;
    System.out.println("value of a is
    System.out.println("value of b is
```

Fig 1



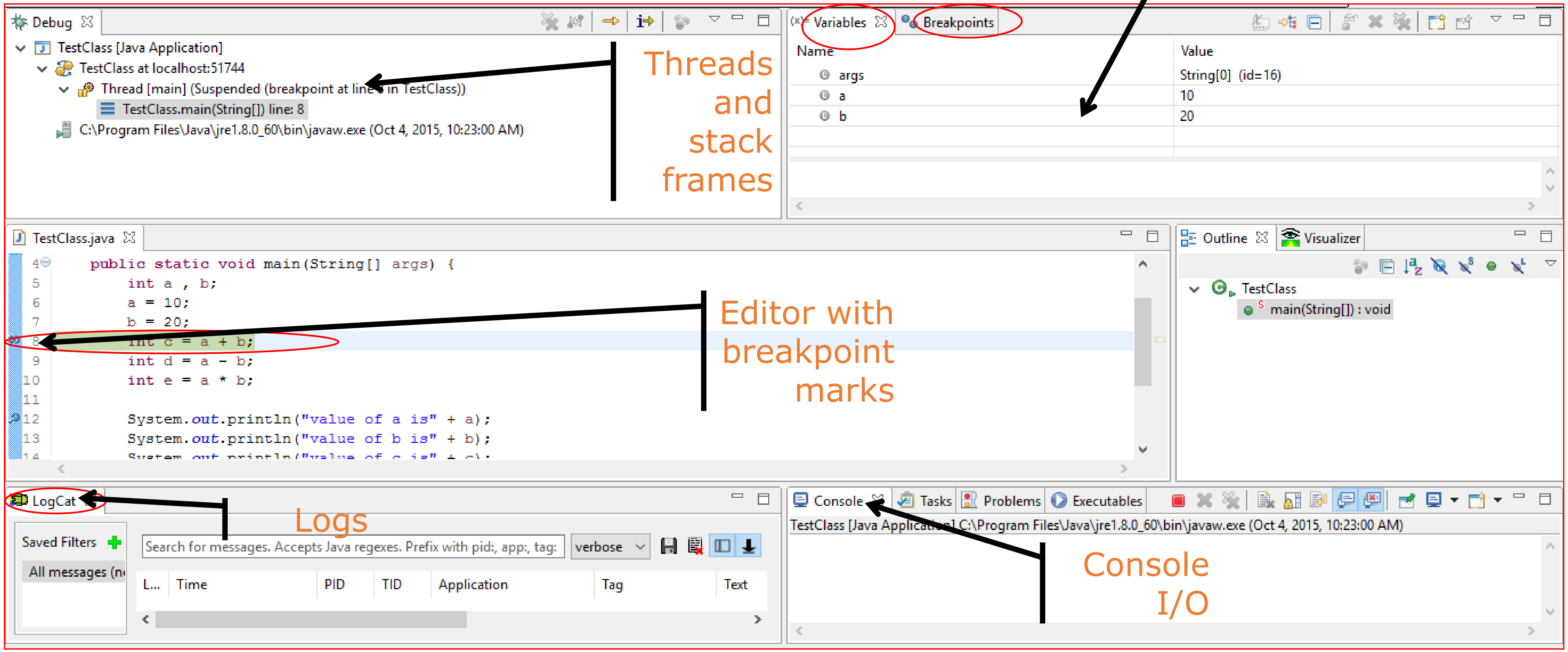
```
<terminated> TestClass [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Oct 4, 2015, 9:06:55 AM)
value of a is10
value of b is20
```

Fig 2

Java Compiler in Eclipse

- Eclipse Java compiler
  - JCK-compliant Java compiler
  - Helpful error messages
  - Generates runnable code even in presence of errors
  - Fully-automatic incremental recompilation
  - High performance
  - Scales to large projects
- Multiple other uses besides the obvious
  - Syntax and spell checking
  - Analyze structure inside Java source file
  - Name resolution
  - Content assist
  - Refactoring
  - Searches

- Set the break points on the lines which are to be debugged. Click on debug option a dialog box appears as a check for switching perspectives. The Debug perspective is shown below.



```
TestClass [Java Application]
  TestClass at localhost:51744
    Thread [main] (Suspended (breakpoint at line 8 in TestClass))
      TestClass.main(String[]) line: 8
  C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Oct 4, 2015, 10:23:00 AM)
```

Name	Value
args	String[0] (id=16)
a	10
b	20

```
 4    public static void main(String[] args) {
 5        int a , b;
 6        a = 10;
 7        b = 20;
 8        int c = a + b;
 9        int d = a - b;
10        int e = a * b;
11
12        System.out.println("value of a is" + a);
13        System.out.println("value of b is" + b);
14        System.out.println("value of c is" + c);
```

TestClass
main(String[]) : void

L...	Time	PID	TID	Application	Tag	Text

```
TestClass [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Oct 4, 2015, 10:23:00 AM)
```

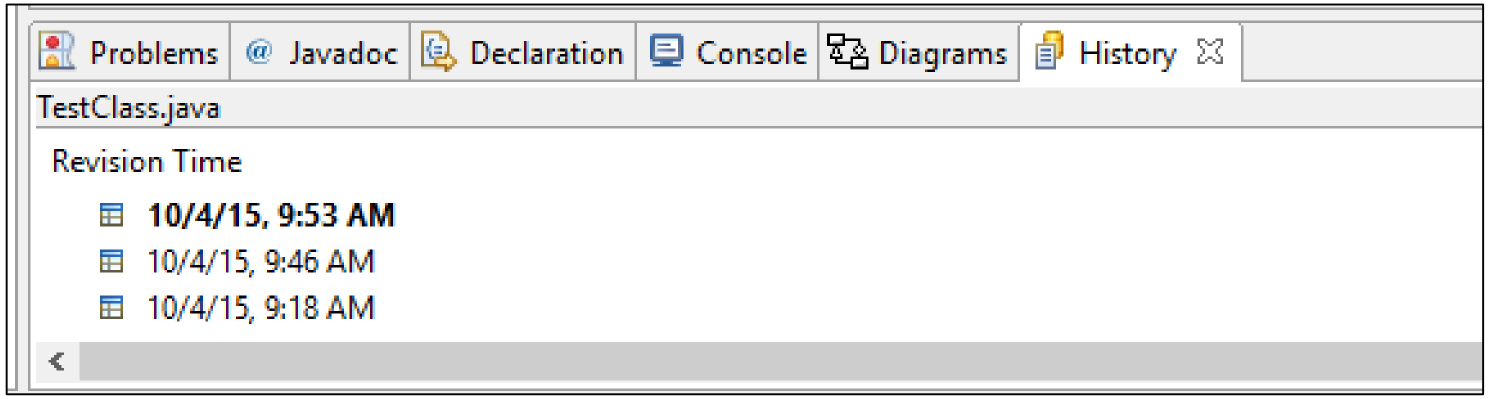
- In this perspective we can monitor the variables and breakpoints in Variables and breakpoints tab. Instruction currently executed is highlighted in green and logs can be monitored from logcat view. Step by step instructions can be executed using F5, F6 and F7 keys or run the whole code by pressing F8.

Eclipse java Debugger

- Run Java programs
  - In separate target JVM (user selectable)
  - Console provides stdout, stdin, stderr for input, output and errors.
  - Scrapbook pages for executing Java code snippets
- Debug Java programs
  - Full source code debugging
  - Any JPDA-compliant JVM
- Debugger features include
  - Method and exception breakpoints
  - Conditional breakpoints
  - Watchpoints
  - Step over, into, return; run to line
  - Inspect and modify fields and local variables
  - Evaluate snippets in context of method
  - Hot swap (if target JVM supports)

Eclipse Code Comparison

- Eclipse offers a feature to compare the content of a file at different instances of time. This enables us to identify the changes made after the last stable version of the code.
- To compare the code Right click on the file and select Compare with and check local history as shown in Fig 1.
- This shows the files at different time stamps in the history view as shown in fig 2.



TestClass.java

Revision Time

10/4/15, 9:53 AM
10/4/15, 9:46 AM
10/4/15, 9:18 AM

Fig 2

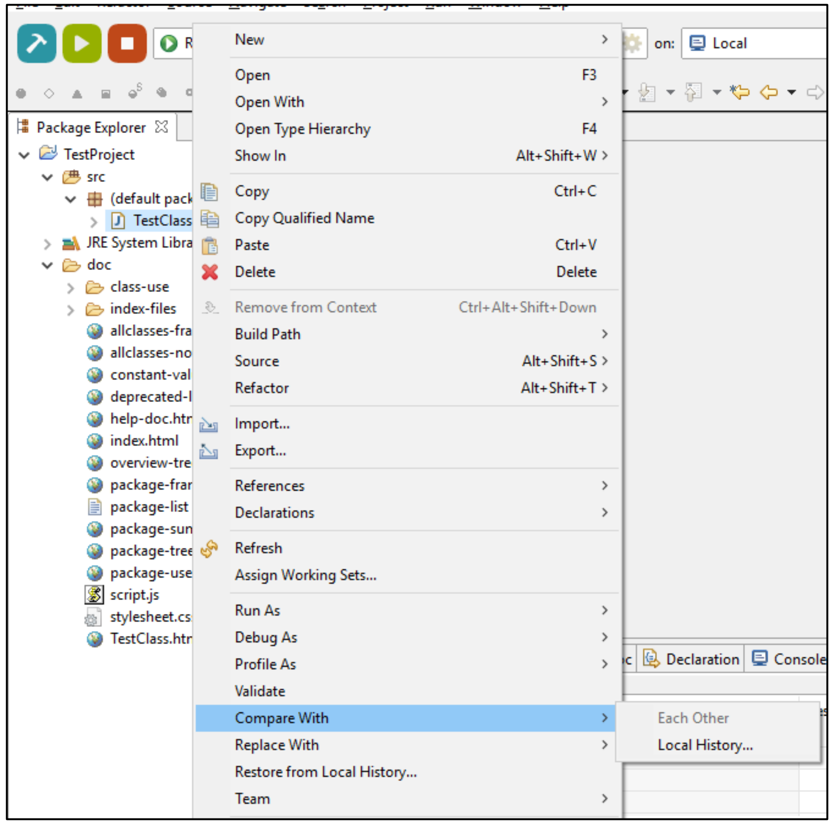
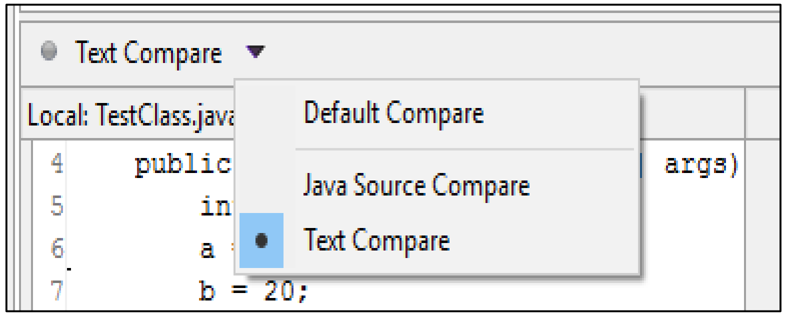


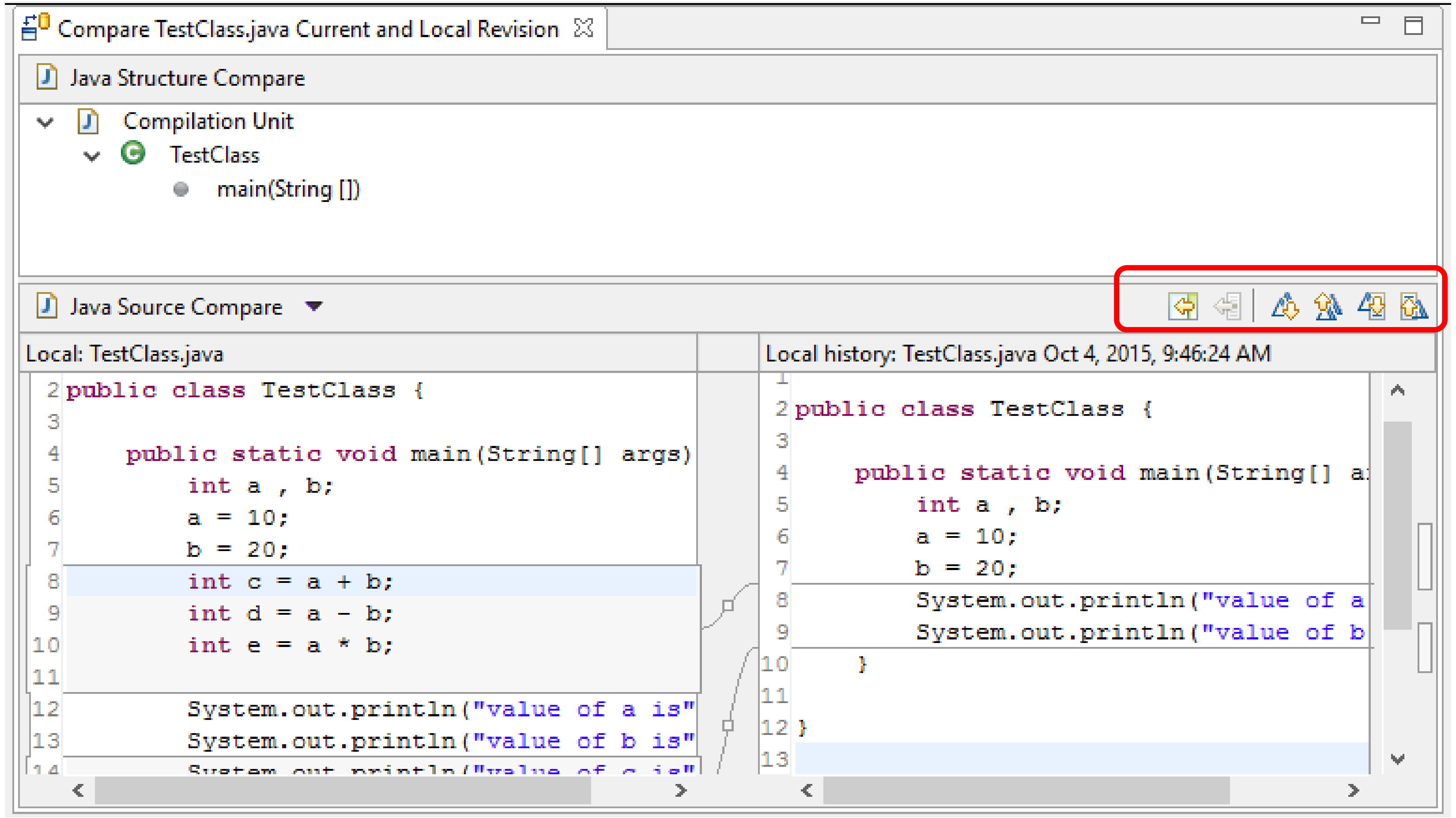
Fig 1

- Once the appropriate entry is selected the code difference is displayed as shown in the fig below. Changes can be viewed and compared using the options provided by the editor to spot the changes (highlighted in red in fig 1).
- Changes can be compared either using the java source compare or general text compare. This can be chosen with the options available. (fig2)



```
4       public                      args)
5           in
6           a
7           b = 20;
```

Fig 2


Compare TestClass.java Current and Local Revision
Java Structure Compare
Compilation Unit
TestClass
main(String [])
Java Source Compare
Local: TestClass.java
Local history: TestClass.java Oct 4, 2015, 9:46:24 AM

```
public class TestClass {

    public static void main(String[] args)
        int a , b;
        a = 10;
        b = 20;
        int c = a + b;
        int d = a - b;
        int e = a * b;

        System.out.println("value of a is"
        System.out.println("value of b is"
```

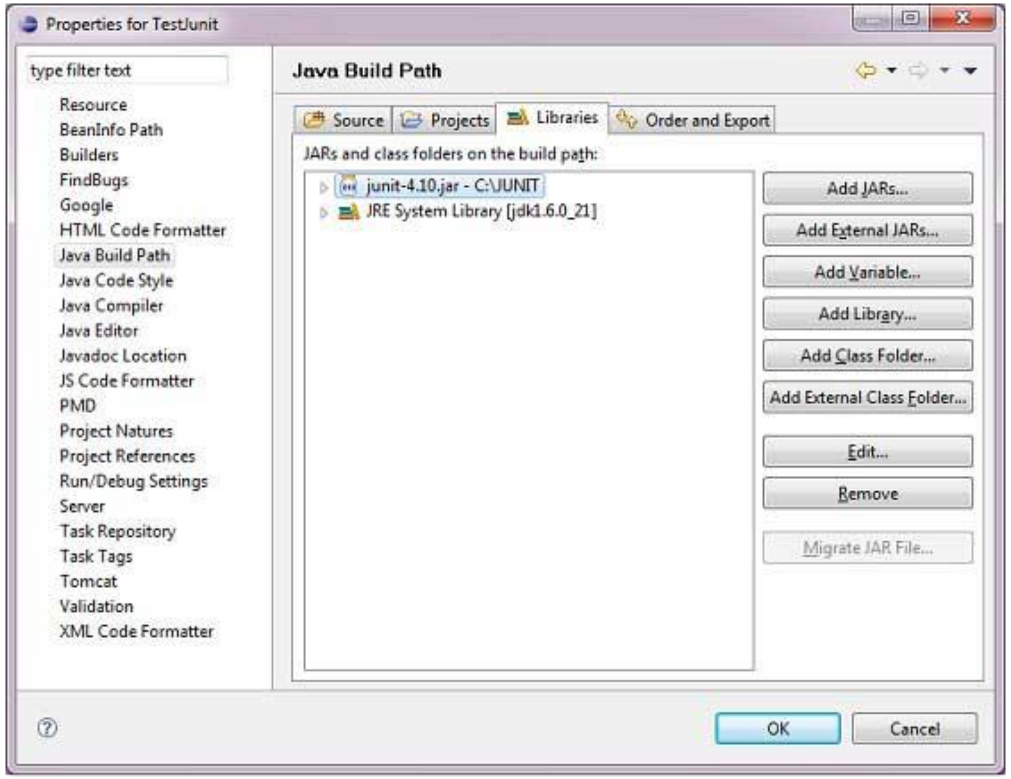
```
public class TestClass {

    public static void main(String[] a
        int a , b;
        a = 10;
        b = 20;
        System.out.println("value of a
        System.out.println("value of b
    }


}
```

Fig 2

ECLIPSE IN TESTING

- A software test is a piece of software which executes another pierce of software and validates if that code results in the correct state (state testing) or executes the correct sequence of events (behavior testing).
- Software unit tests allow the developer to verify that a piece of program logic is correct.
- There are several kinds of testing that has been integrated on Eclipse. For demonstration let us consider Junit testing here.
- A JUnit test is a method contained in a class which is only used for testing. This is called a Test class.

- To start with junit development in eclipse download Junit archive and save it in a directory.
- Open eclipse -> right click on project and click on property > Build Path > Configure Build Path and add the junit-4.10.jar in the using Add External Jar button.
- Junit plugin is required to which is a part of JDT. To install
- JDT check here



JARs and class folders on the build path:
junit-4.10.jar - C:\JUNIT
JRE System Library [jdk1.6.0_21]

- To use JUnit you must create a separate .java file in your project that will test one of your existing classes. In the Package Explorer area on the left side of the Eclipse window, right-click the class you want to test and click New → JUnit Test Case.(shown in fig 1)
- A dialog box will pop up to help you create your test case. Make sure that the option at the top is set to use JUnit 4, not JUnit 3. Click Next.(shown in fig 2)

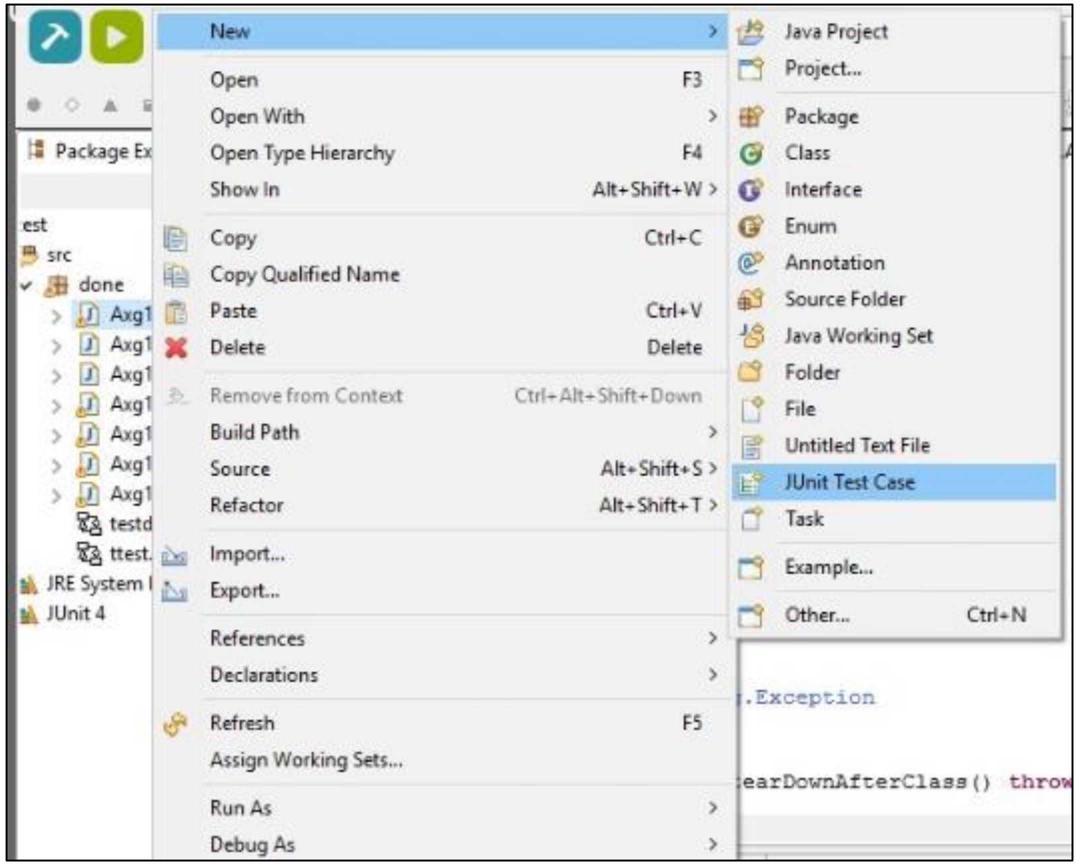
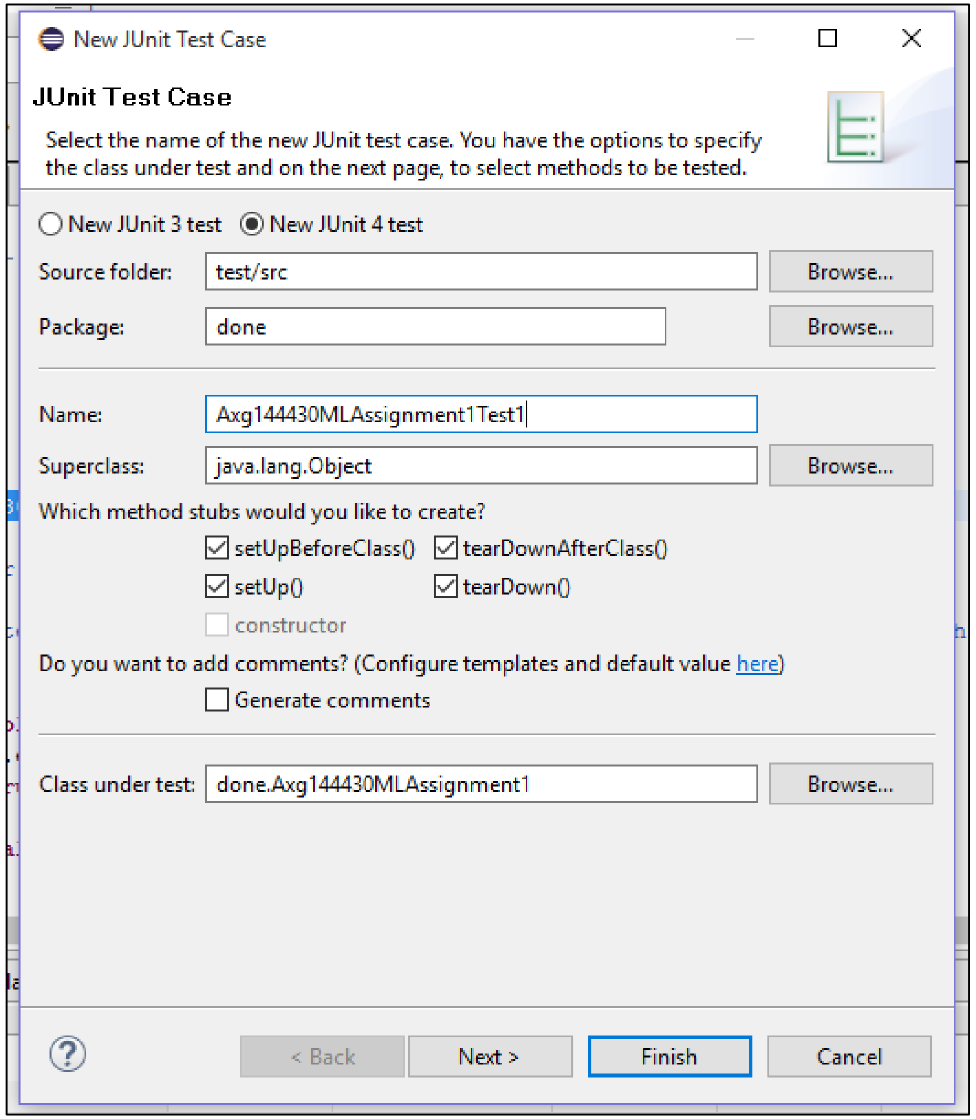


Fig 1



New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

( ) New JUnit 3 test [✓ New JUnit 4 test]

Source folder: test/src

Package: done

Name: Axg144430MLAssignment1Test1

Superclass: java.lang.Object

Which method stubs would you like to create?

☑ setUpBeforeClass() ☑ tearDownAfterClass()

☑ setUp() ☑ tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value here)

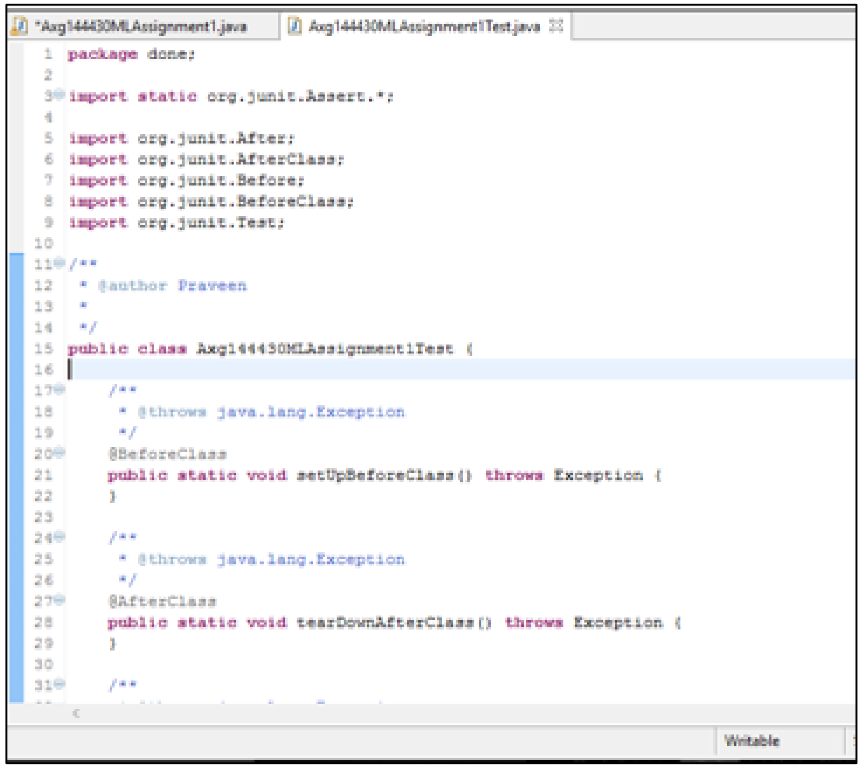
☐ Generate comments

Class under test: done.Axg144430MLAssignment1

Fig 2

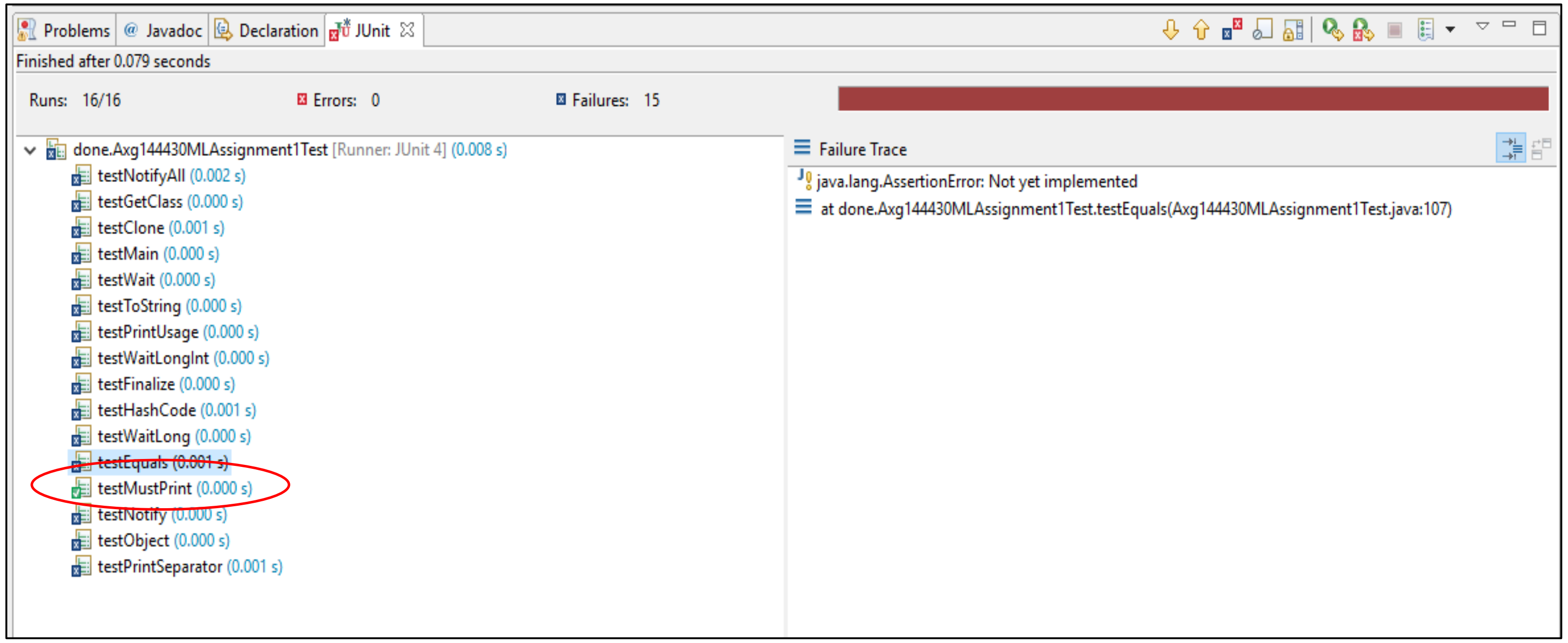
- This creates a test file with built in skeleton of the test cases of the methods in the class and the setup and the teardown methods.(shown in fig 1)

- Write the test cases for individual modules or methods and run the test cases. To run the test cases right click on project -> run as ->junit test.
- Once the test case is executed it shows the results as shown in fig(2) in the Junit view of Eclipse.



```
package done;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * @author Praveen
 *
 */
public class Axg144430MLAssignment1Test {

    /**
     * @throws java.lang.Exception
     */
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    /**
     * @throws java.lang.Exception
     */
    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    /**
```

Fig 2



```
Finished after 0.079 seconds
Runs: 16/16          Errors: 0          Failures: 15

done.Axg144430MLAssignment1Test [Runner: JUnit 4] (0.008 s)
    testNotifyAll (0.002 s)
    testGetClass (0.000 s)
    testClone (0.001 s)
    testMain (0.000 s)
    testWait (0.000 s)
    testToString (0.000 s)
    testPrintUsage (0.000 s)
    testWaitLongInt (0.000 s)
    testFinalize (0.000 s)
    testHashCode (0.001 s)
    testWaitLong (0.000 s)
    testEquals (0.001 s)
    testMustPrint (0.000 s)
    testNotify (0.000 s)
    testObject (0.000 s)
    testPrintSeparator (0.001 s)

java.lang.AssertionError: Not yet implemented
at done.Axg144430MLAssignment1Test.testEquals(Axg144430MLAssignment1Test.java:107)
```

Fig 1

The results are of 3 kinds

- Errors : Test Cases with errors
- Failures : Test cases with wrong logic
- Success : Test cases that passed the test.

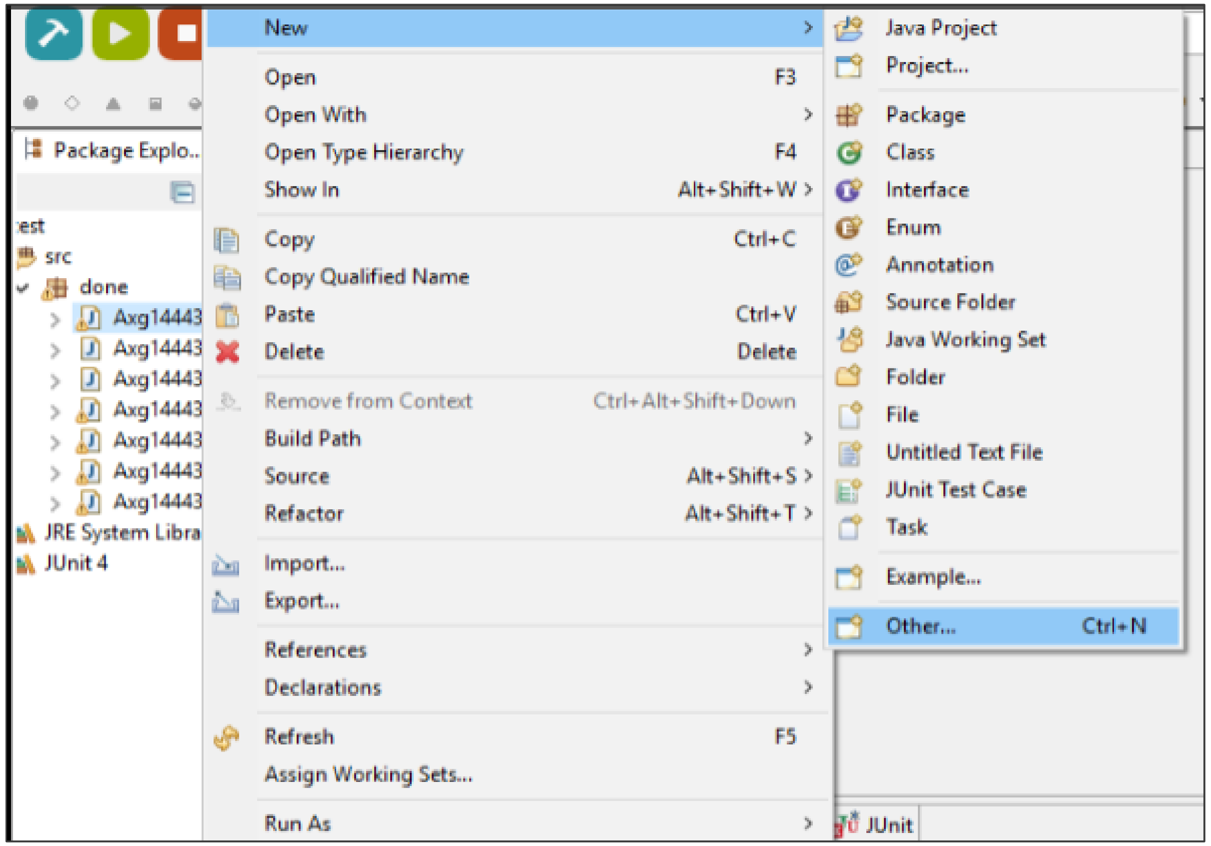
ECLIPSE IN DESIGN

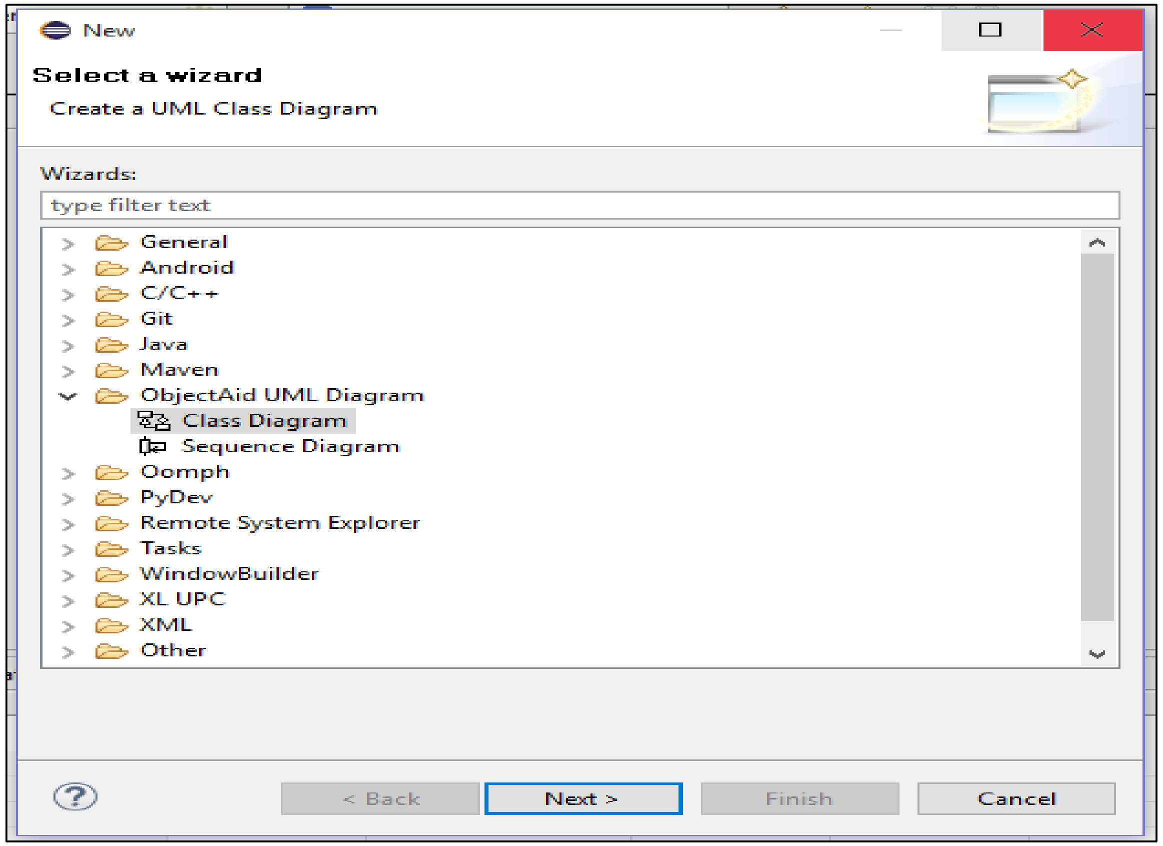
- Eclipse can be used for developing the design documentation of a project like Class diagram, Sequence diagram etc.
- Eclipse has a plugin called Object Aid which generated the class diagram and sequence diagram from the source code.
- ObjectAid can be installed from here with the installation instructions here.
- The ObjectAid UML Explorer is an agile and lightweight code visualization tool for the Eclipse IDE.

- It shows your Java source code and libraries in live UML class and sequence diagrams that automatically update as your code changes.
- It uses the UML notation to show a graphical representation of existing code that is as accurate and up-to-date as the text editor, while being very easy to use.
- Using ObjectAid all diagrams in your Eclipse workspace are updated with refactoring changes as appropriate. If necessary, they are checked out of your version control system.

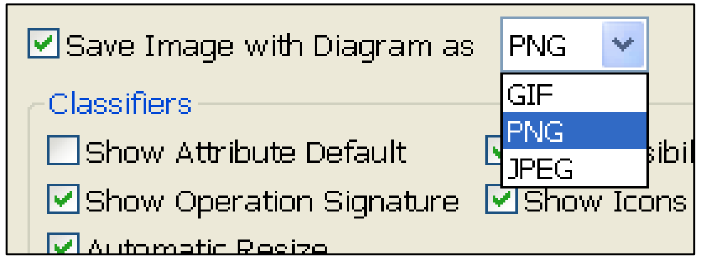
ObjectAid : A Plugin for Eclipse

- Objectaid is a free tool for Eclipse which can be used to automatically create class diagrams.
- Lets consider the procedure for creating UML Class diagrams.
- After installing the plugin, from the package explorer or File menu , create a new ObjectAid class diagram
- Click on the File package -> New -> Other and select Class Diagram**.**





- While creating the diagram , you can specify what all Classifiers and Relationships among classes you need in the diagram to have . You can also specify the Attributes and Operations of a class that you want to be shown in the diagram.
- You can mark Save Image in case you need the diagram in an Image file for some presentation etc.



☑ Save Image with Diagram as PNG

GIF / PNG / JPEG

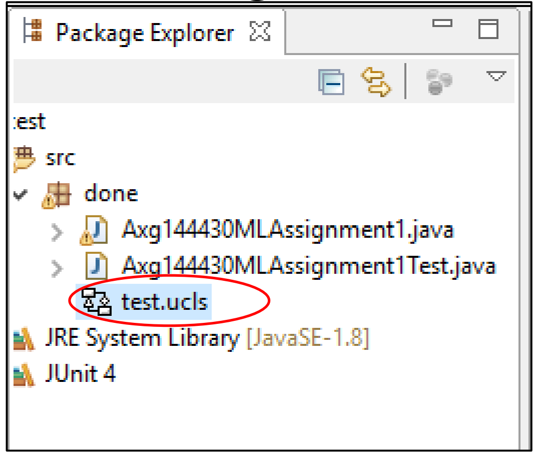
Classifiers

☐ Show Attribute Default ☑ Show Visibil

☑ Show Operation Signature ☑ Show Icons

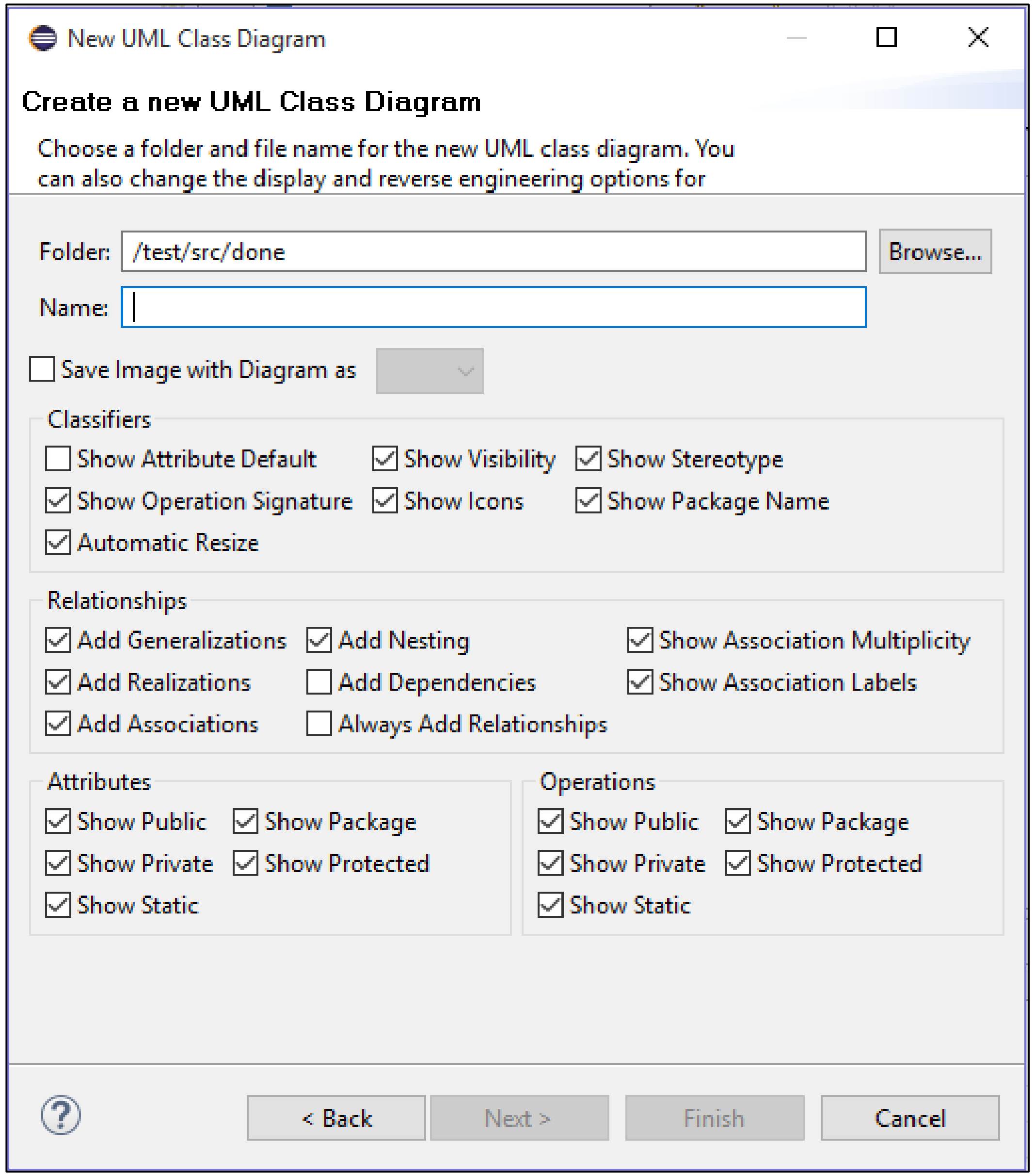
☑ Automatic Resize

- Once created , the Class diagram icon will show in package explorer.



Package Explorer

```
est
  src
  ∨ done
    > Axg144430MLAssignment1.java
    > Axg144430MLAssignment1Test.java
      test.ucls
  JRE System Library [JavaSE-1.8]
  JUnit 4
```



New UML Class Diagram

Create a new UML Class Diagram

Choose a folder and file name for the new UML class diagram. You can also change the display and reverse engineering options for

Folder: /test/src/done Browse...

Name:

☐ Save Image with Diagram as

Classifiers

☐ Show Attribute Default ☑ Show Visibility ☑ Show Stereotype

☑ Show Operation Signature ☑ Show Icons ☑ Show Package Name

☑ Automatic Resize

Relationships

☑ Add Generalizations ☑ Add Nesting ☑ Show Association Multiplicity

☑ Add Realizations ☐ Add Dependencies ☑ Show Association Labels

☑ Add Associations ☐ Always Add Relationships

Attributes

☑ Show Public ☑ Show Package

☑ Show Private ☑ Show Protected

☑ Show Static

Operations

☑ Show Public ☑ Show Package

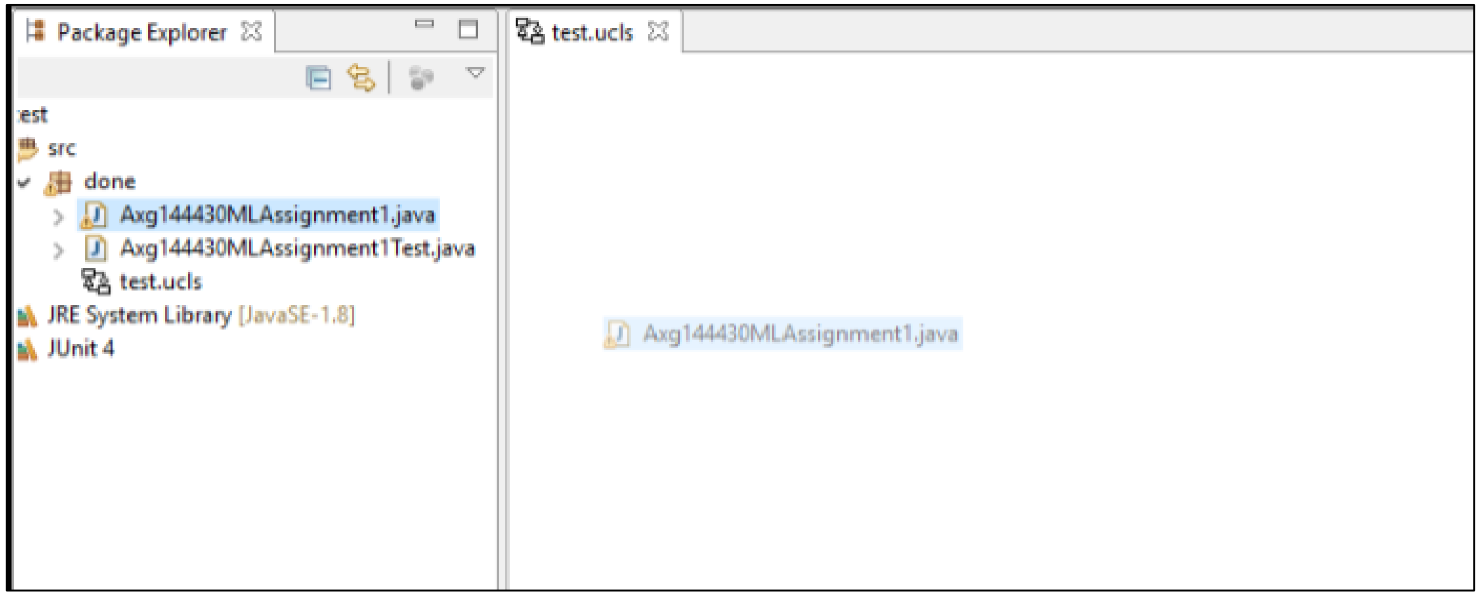
☑ Show Private ☑ Show Protected

☑ Show Static

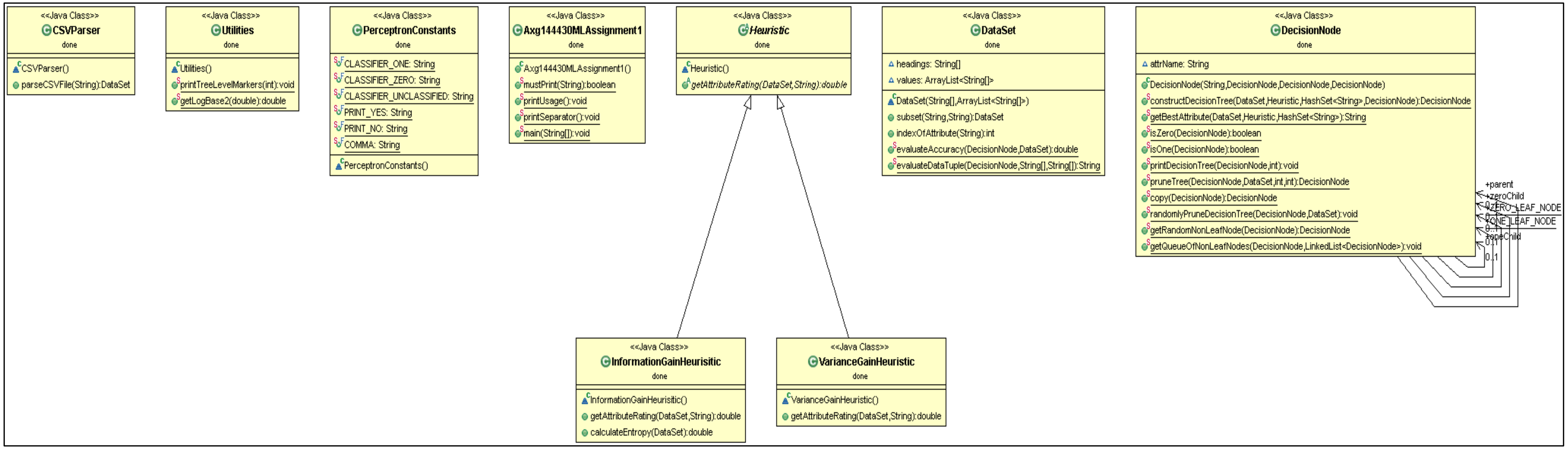
< Back Next > Finish Cancel

- You can open and drag into it the classes whose diagram you want to create.

- After you drag , the diagram will appear , listing the attributes and operations were allowed while creating it. Example shown in figure below.

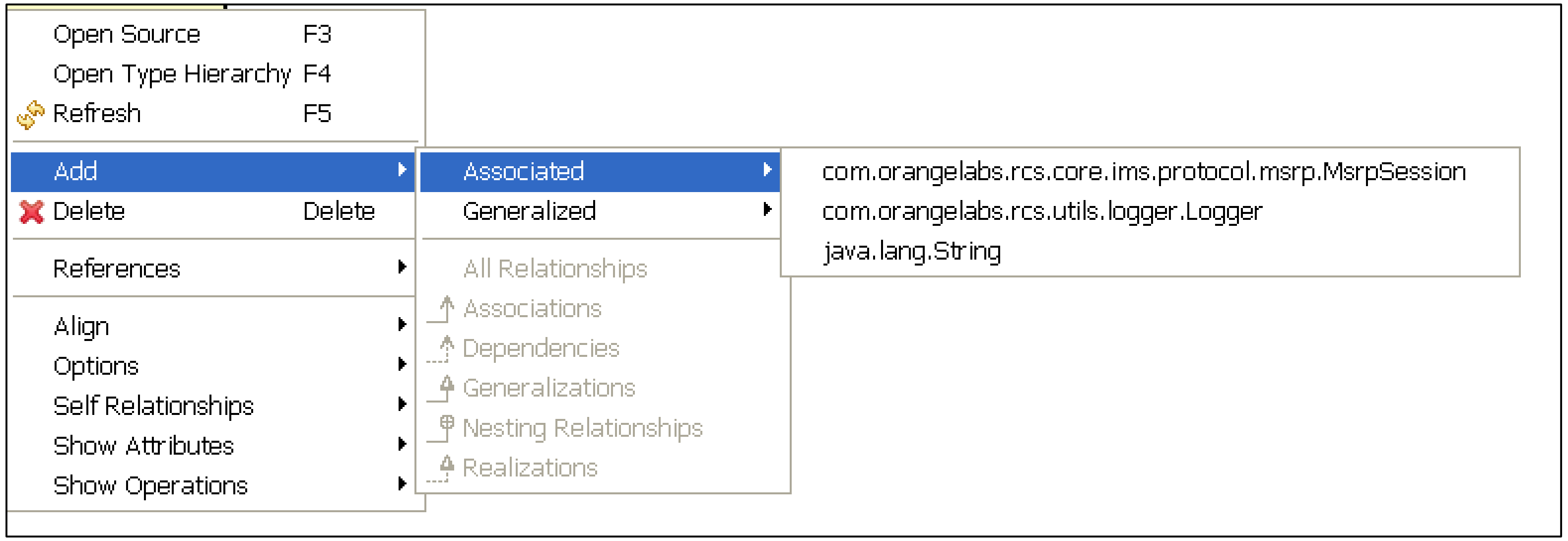


Package Explorer
test.ucls
est
src
done
Axg144430MLAssignment1.java
Axg144430MLAssignment1Test.java
test.ucls
JRE System Library [JavaSE-1.8]
JUnit 4
Axg144430MLAssignment1.java



<<Java Class>>
CSVParser
done
CSVParser()
parseCSVFile(String):DataSet
<<Java Class>>
Utilities
done
Utilities()
printTreeLevelMarkers(int):void
getLogBase2(double):double
<<Java Class>>
PerceptronConstants
done
CLASSIFIER_ONE: String
CLASSIFIER_ZERO: String
CLASSIFIER_UNCLASSIFIED: String
PRINT_YES: String
PRINT_NO: String
COMMA: String
PerceptronConstants()
<<Java Class>>
Axg144430MLAssignment1
done
Axg144430MLAssignment1()
mustPrint(String):boolean
printUsage():void
printSeparator():void
main(String[]):void
<<Java Class>>
Heuristic
done
Heuristic()
getAttributeRating(DataSet,String):double
<<Java Class>>
DataSet
done
headings: String[]
values: ArrayList<String[]>
DataSet(String[],ArrayList<String[]>)
subset(String,String):DataSet
indexOfAttribute(String):int
evaluateAccuracy(DecisionNode,DataSet):double
evaluateDataTuple(DecisionNode,String[],String[]):String
<<Java Class>>
DecisionNode
done
attrName: String
DecisionNode(String,DecisionNode,DecisionNode,DecisionNode)
constructDecisionTree(DataSet,Heuristic,HashSet<String>,DecisionNode):DecisionNode
getBestAttribute(DataSet,Heuristic,HashSet<String>):String
isZero(DecisionNode):boolean
isOne(DecisionNode):boolean
printDecisionTree(DecisionNode,int):void
pruneTree(DecisionNode,DataSet,int,int):DecisionNode
copy(DecisionNode):DecisionNode
randomlyPruneDecisionTree(DecisionNode,DataSet):void
getRandomNonLeafNode(DecisionNode):DecisionNode
getQueueOfNonLeafNodes(DecisionNode,LinkedList<DecisionNode>):void
+parent
+zeroChild
+ZERO_LEAF_NODE
+ONE_LEAF_NODE
+oneChild
0..1
0..1
0..1
0..1
0..1
<<Java Class>>
InformationGainHeurisitic
done
InformationGainHeurisitic()
getAttributeRating(DataSet,String):double
calculateEntropy(DataSet):double
<<Java Class>>
VarianceGainHeuristic
done
VarianceGainHeuristic()
getAttributeRating(DataSet,String):double

- Right click on a class to open the menu which has useful operations through which you can extend the diagram by adding Associates and Relationships. By clicking an operation or attribute , you can go to its source code .



- If you want to change the type of attributes and operations you allowed while creating the diagram , you can also change it from this Menu.(shown in fig 2)

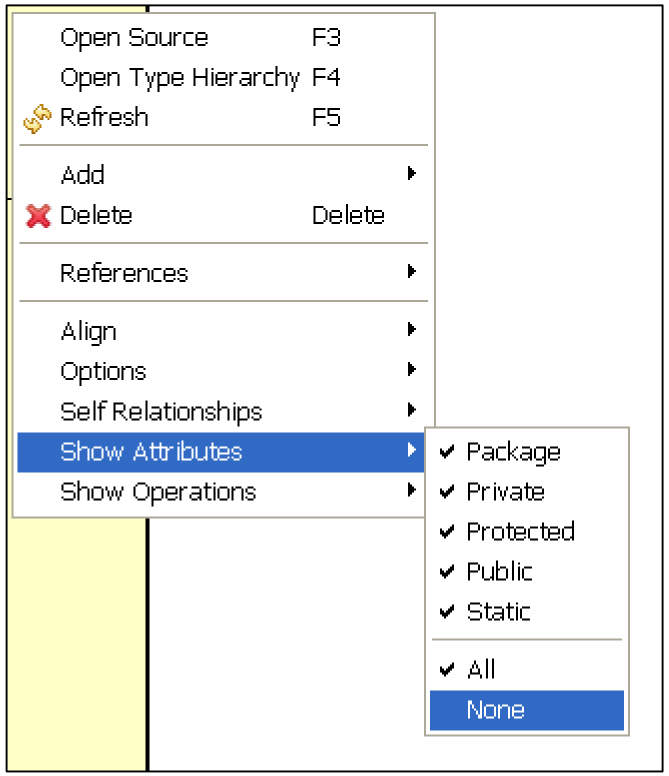


Fig 2

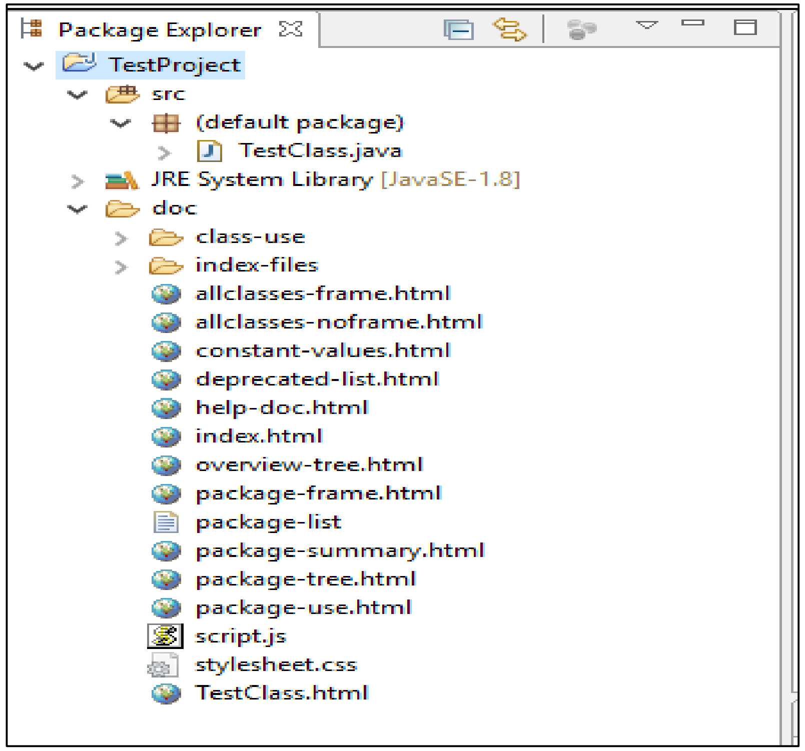
ObjectAid Advantages

Several unique features of ObjectAid includes:

- Generates the diagram with simple drag and drop.
- Any update in the code in Eclipse, UML diagram is updated as well; there is no need to reverse engineering of the source code.
- Refactoring updates on the diagram as well as the source code. Renaming a field or moving a class, diagram simply reflects the changes without going out of sync.
- All diagrams in Eclipse workspace are updated with refactoring changes as appropriate. If necessary, they are checked out of version control system.(Integration of Version control)
- Diagrams are fully integrated into the Eclipse IDE. Java classes can be dragged from any other view onto the diagram, and diagram-related information is shown in other views wherever applicable.
- The ObjectAid UML Explorer achieves all this while staying light, fast and easy to use
- More options are available to edit the diagram by adding and removing certain features and relationships in the diagram.
- For more information check this.

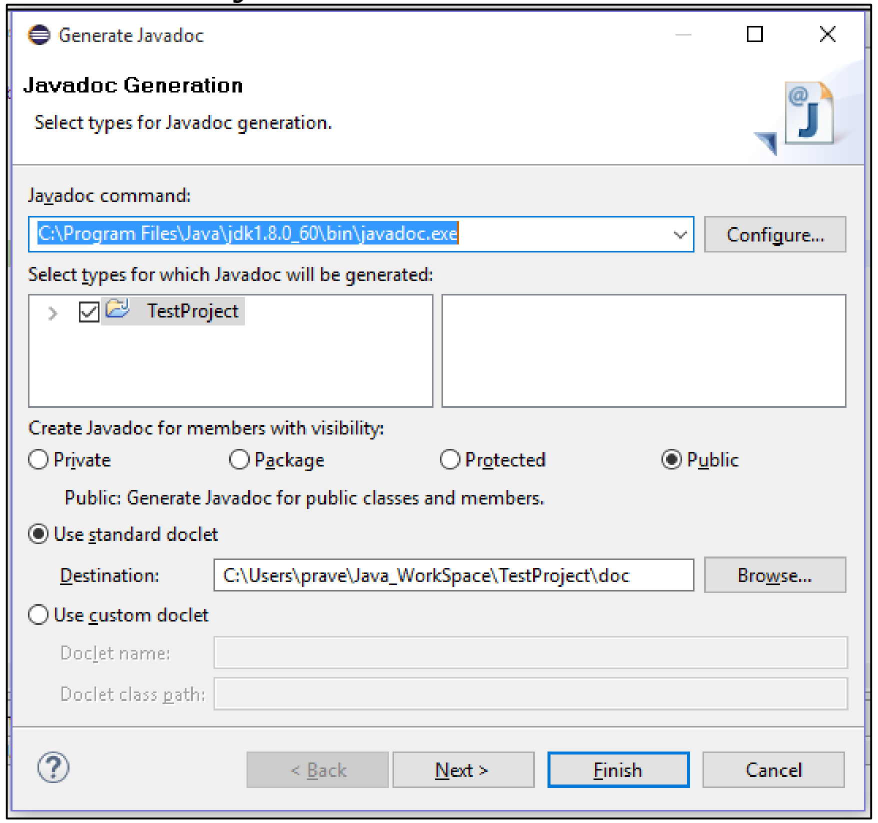
JAVADOC IN ECLIPSE

- Javadoc is a documentation generator created by Sun Microsystems for the Java language for generating API documentation in HTML format from Java source code.
- We can generate the javadoc for a java project from Eclipse. Go to project > generate java doc.
- At first step, define settings for:
  - Select path for the javadoc.exe tool from the JDK
  - Classes and methods for which to generate Javadoc based on their visibility.
  - Location of the Javadoc as shown in fig 1.
- Generated Javadoc can be viewed in the project explorer(fig 2).


Package Explorer

```
TestProject
  src
    (default package)
      TestClass.java
  JRE System Library [JavaSE-1.8]
  doc
    class-use
    index-files
    allclasses-frame.html
    allclasses-noframe.html
    constant-values.html
    deprecated-list.html
    help-doc.html
    index.html
    overview-tree.html
    package-frame.html
    package-list
    package-summary.html
    package-tree.html
    package-use.html
    script.js
    stylesheet.css
    TestClass.html
```

Fig 2


Generate Javadoc

Javadoc Generation

Select types for Javadoc generation.

Javadoc command: C:\Program Files\Java\jdk1.8.0_60\bin\javadoc.exe

Select types for which Javadoc will be generated: [✓] TestProject

Create Javadoc for members with visibility: Private Package Protected [✓ Public]

Public: Generate Javadoc for public classes and members.

[✓] Use standard doclet

Destination: C:\Users\prave\Java_WorkSpace\TestProject\doc

Use custom doclet

Doclet name:

Doclet class path:

Fig 1

ECLIPSE IN SOURCE CONTROL

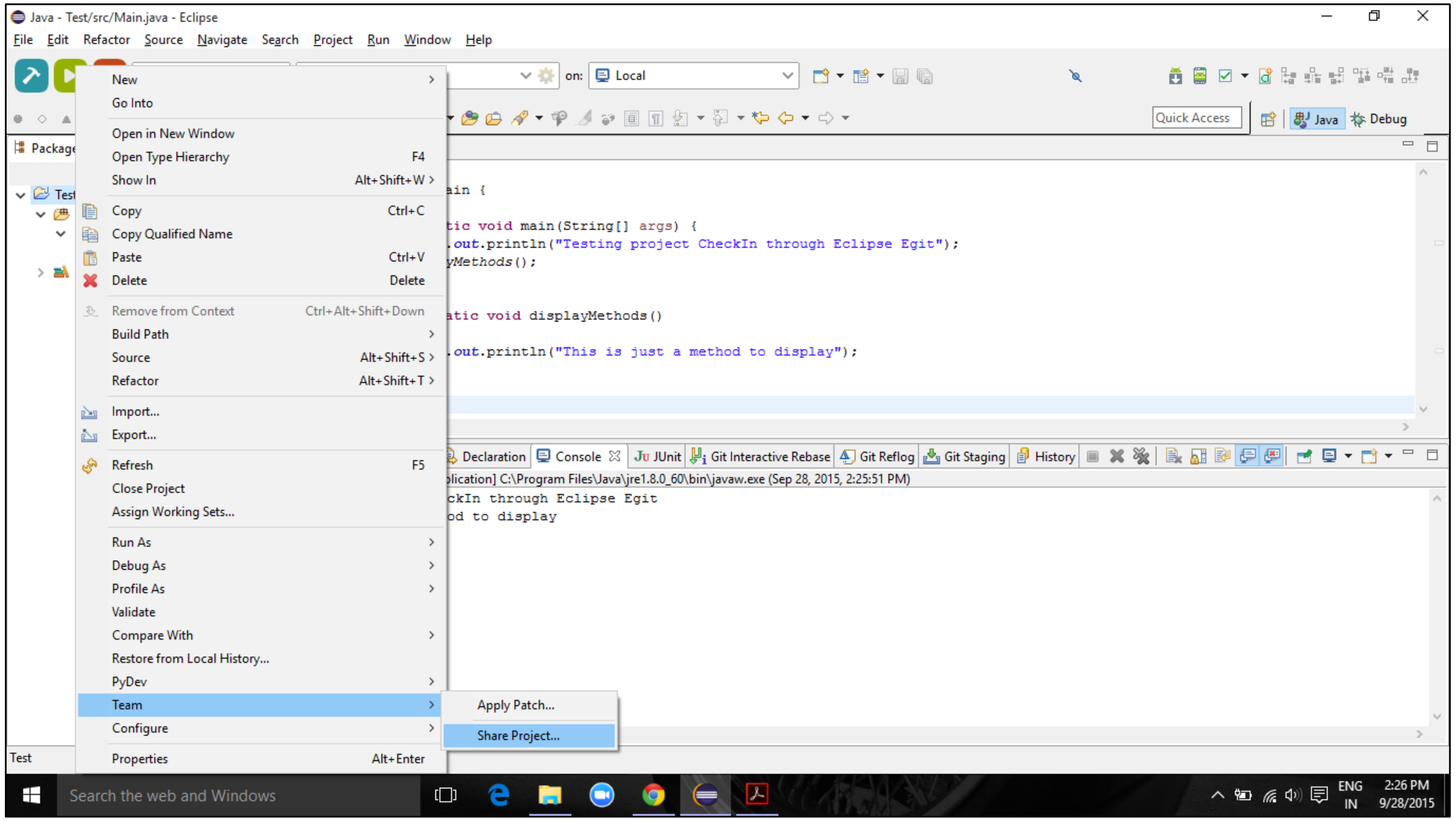
- Source Control or Version Control is the management of source code of softwares and other files.
- These can be used to track the changes and revert back if necessary without affecting the other modules. Each changes made are identified by a unique Identifier.
- Examples include : Perforce, Git, ClearCase, Vault etc.

- ECLIPSE has been integrated with a plugin called **Egit** to access Git repositories.
- For instructions to install the Egit plugin into Eclipse click here.
- Git is a distributed SCM(), which means every developer has a full copy of all history of every revision of the code, making queries against the history very fast and versatile.
- This is one of the major features of Eclipse because of which it is used in industrial developments as well. It can be integrated with many public and private Source Control tools.
- Examples : Perforce – P4Eclipse, Git – Egit etc.

EGIT - Plugin for Git in Eclipse

Lets consider an example of Egit for the Source Control in Eclipse.

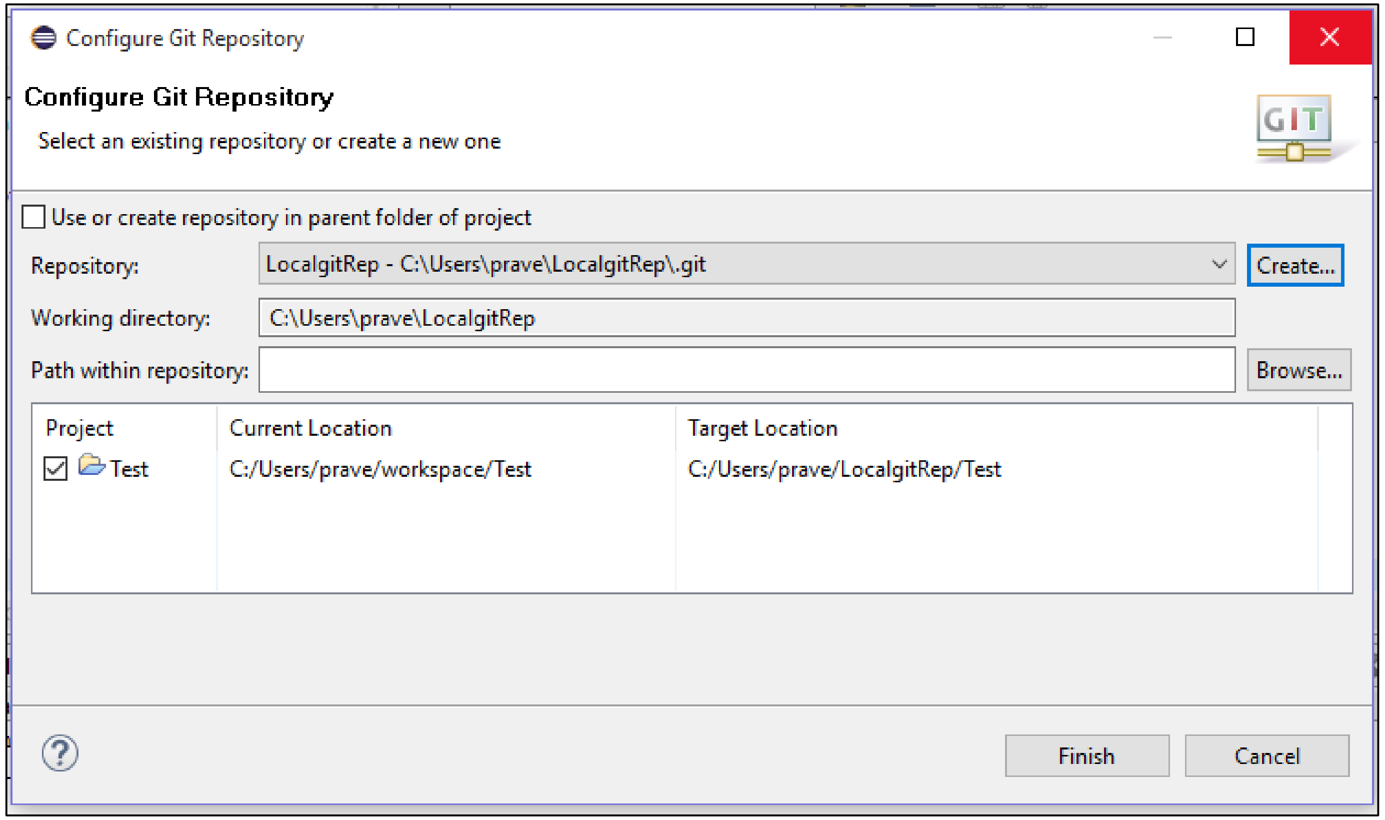
- Install Egit plugin on Eclipse from here.
- Create a sample project in any language and save it.
- For Ex : a project named Test is created here.
- Right Click on the project and select Team -> Share Project.
- Share the project on Git and this allows you to create a local repository for the project if it is not created previously.



```
ain {

tic void main(String[] args) {
.out.println("Testing project CheckIn through Eclipse Egit");
yMethods();

atic void displayMethods()

.out.println("This is just a method to display");
```

```
plication] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Sep 28, 2015, 2:25:51 PM)
ckIn through Eclipse Egit
od to display
```

- To create a repository click on the Create button in the pop-up and specify a name. A local repository is created with the name specified and initializes Git in the repository. This is equivalent to executing “git init” command in the repository.



Configure Git Repository

Select an existing repository or create a new one

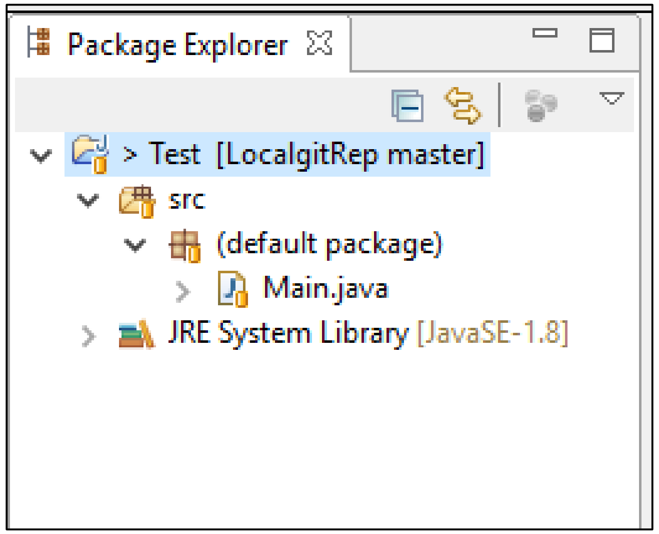
☐ Use or create repository in parent folder of project

Repository: LocalgitRep - C:\Users\prave\LocalgitRep\.git [Create...]

Working directory: C:\Users\prave\LocalgitRep

Path within repository: [Browse...]

Project	Current Location	Target Location
✓ Test	C:/Users/prave/workspace/Test	C:/Users/prave/LocalgitRep/Test



```
> Test  [LocalgitRep master]
  src
    (default package)
      Main.java
  JRE System Library [JavaSE-1.8]
```

Fig 2

- Once the repository is created and initialized the structure in the project explorer displays the repository name and the branch name in git if any. In the example fig 2 the project is in LocalgitRep directory and on master branch.

- To add the changes made into the local repository we use the commit option. This pushes all the data in the staging area into the local workspace.

- To commit right click on the project and select team and check commit option as shown in fig 1.

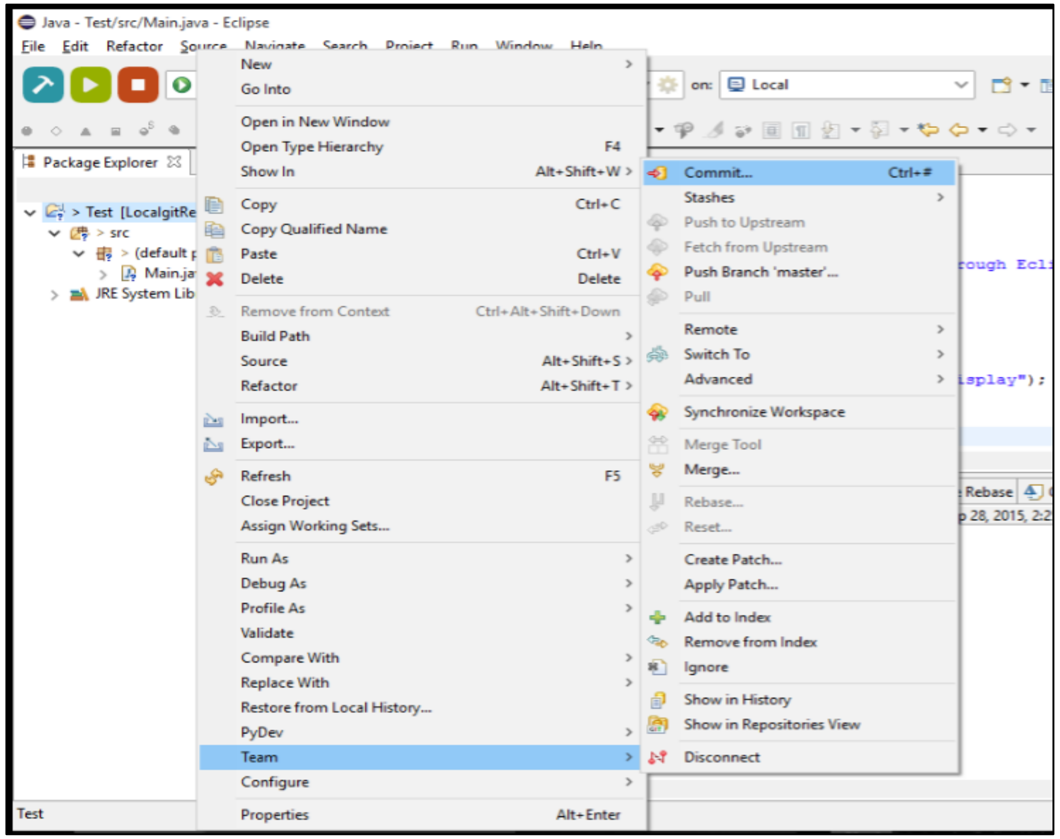
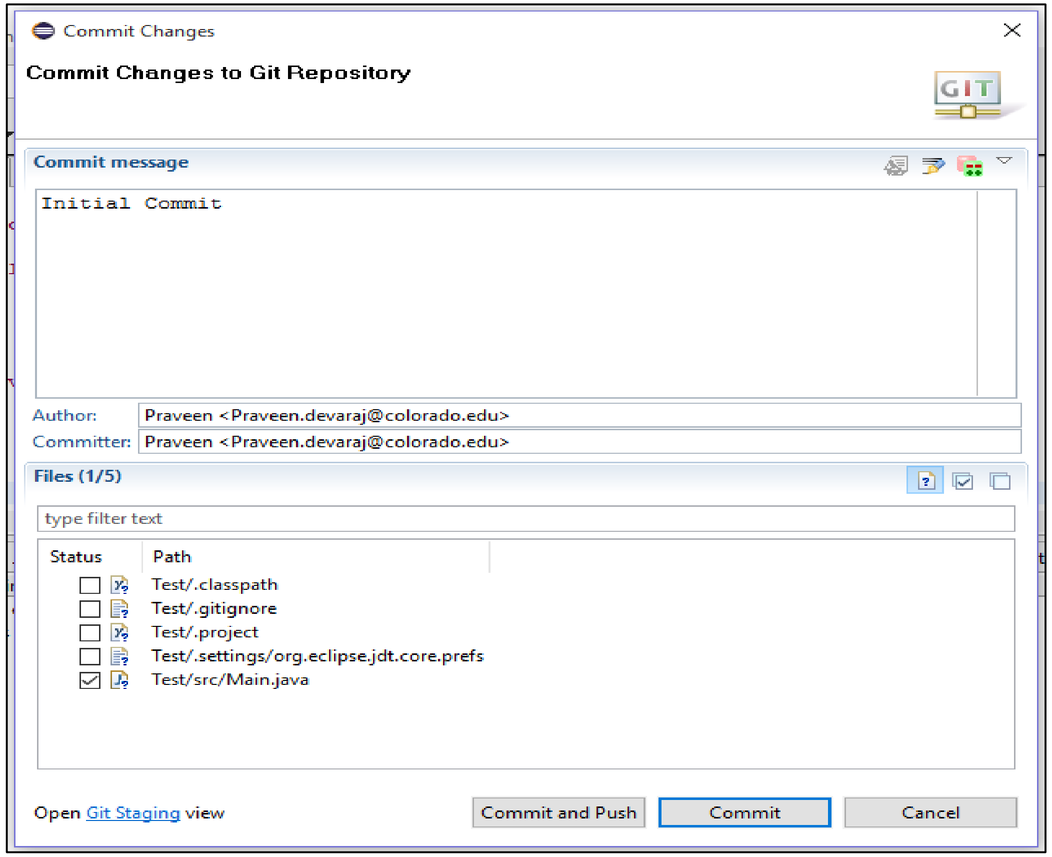


Fig 1



Commit Changes

Commit Changes to Git Repository

Commit message

```
Initial Commit
```

Author: Praveen <Praveen.devaraj@colorado.edu>

Committer: Praveen <Praveen.devaraj@colorado.edu>

Files (1/5)

type filter text

Status	Path
☐	Test/.classpath
☐	Test/.gitignore
☐	Test/.project
☐	Test/.settings/org.eclipse.jdt.core.prefs
☑	Test/src/Main.java

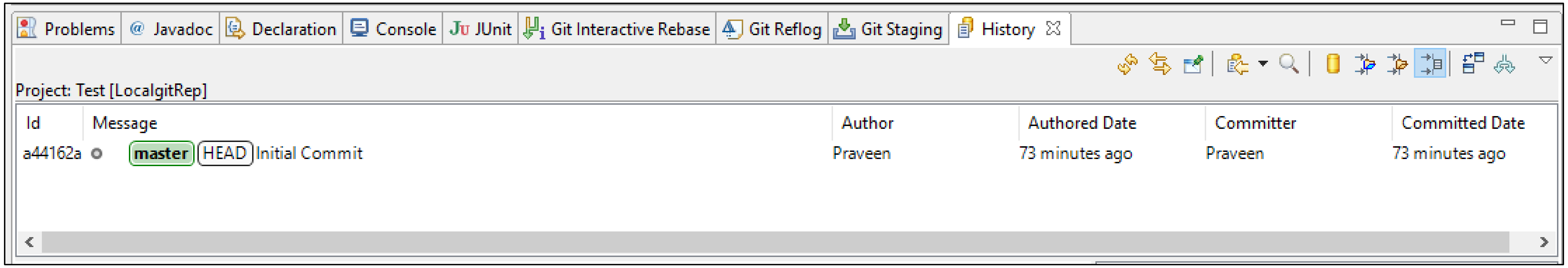
Open Git Staging view

Commit and Push | Commit | Cancel

Fig 2

- After checking the commit option a pop up is displayed where we fill the commit message and select the files individually which are to be committed. (shown in fig 2)

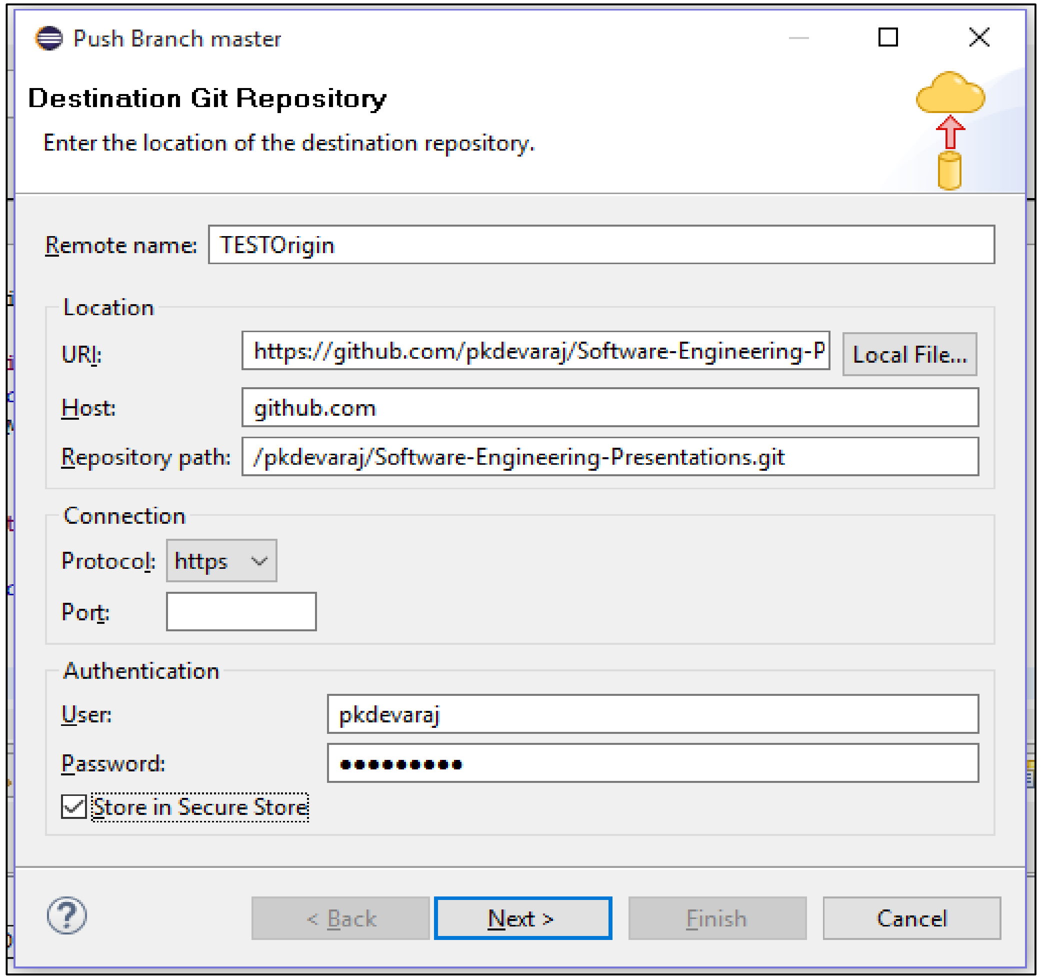
- After the changes are committed we can check the branch and the commit changes in the history view of eclipse. To view this right click on the project and select show In history.
- History View shows the author , branch information, commit messages and their Ids, Authored Date and the date of each commit. Ex : Figure 1, this has only one commit on Master branch.



Project: Test [LocalgitRep]

Id	Message	Author	Authored Date	Committer	Committed Date
a44162a	master HEAD Initial Commit	Praveen	73 minutes ago	Praveen	73 minutes ago

- All the changes committed are now saved in the local repository and not in the server.
- To push the changes from the local repository to the remote repository we use the push option available when we right click on project and select Team.
- A pop up appears when clicked on Push which requests for the url of the repository and the user credentials to authenticate the user. (show in Next Slide)
- Successful push sends the changes to the remote repository and can be viewed on github account of the user.


Push Branch master

Destination Git Repository

Enter the location of the destination repository.

Remote name: TESTOrigin

Location

URI: https://github.com/pkdevaraj/Software-Engineering-P

Host: github.com

Repository path: /pkdevaraj/Software-Engineering-Presentations.git

Connection

Protocol: https

Port:

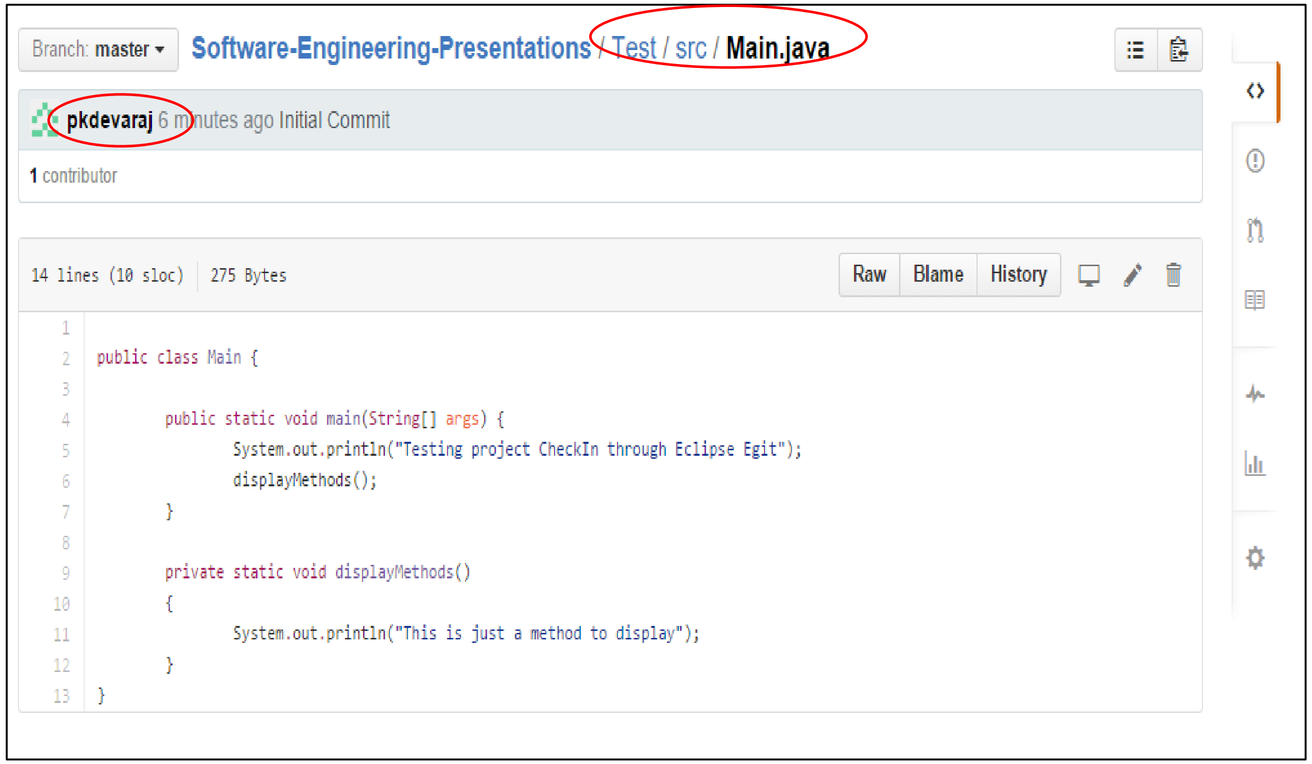
Authentication

User: pkdevaraj

Password: ●●●●●●●●●●

☑ Store in Secure Store

Fig 1 (left) : Pop up window for push command.
Fig 2 (below) : View of file in Github remote repository


Branch: master ▾ Software-Engineering-Presentations / Test / src / Main.java

pkdevaraj 6 minutes ago Initial Commit

1 contributor

14 lines (10 sloc) 275 Bytes

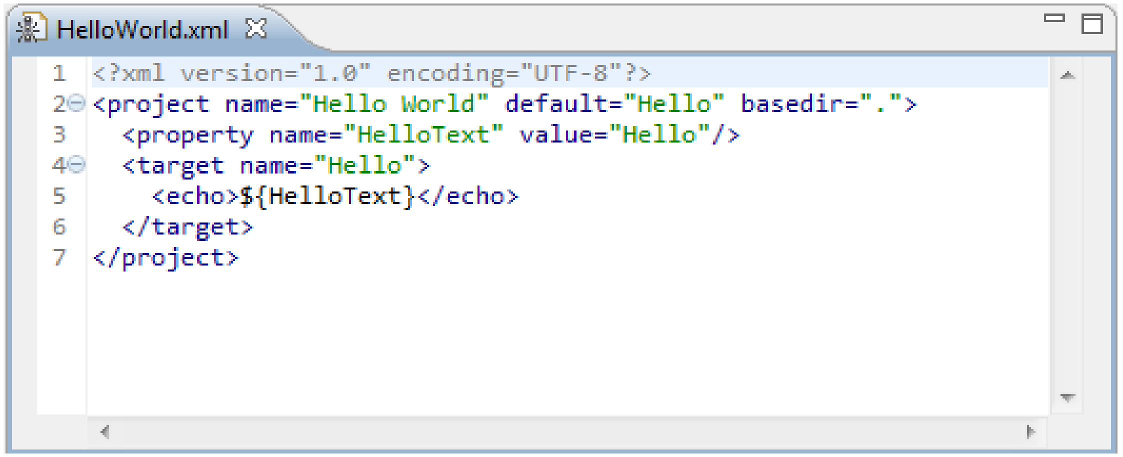
```

public class Main {

	public static void main(String[] args) {
		System.out.println("Testing project CheckIn through Eclipse Egit");
		displayMethods();
	}

	private static void displayMethods()
	{
		System.out.println("This is just a method to display");
	}
}
```

- Eclipse offers a GUI based access to Git and is integrated to the development environment which makes it easier to use.
- Similarly Eclipse offers a wide range of plugins for many Source Control SCMs.
- More information on Egit can be found here.

ECLIPSE AND ANT BUILD

- ANT build is a software tool for automating build processes developed in java.
- The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications.
- Ant can also be used effectively to build non Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.
- The most immediately noticeable difference between Ant and Make is that Ant uses XML to describe the build process and its dependencies, whereas Make uses Makefile format.
- Ant buildfiles are just text files, to create an Ant buildfile in Eclipse: File > New > File (Enter a name for the file) Click Finish.
- By default the Ant Editor has an association with build.xml file or File associations can be manually set using Window > Preference > General > Editors > File associations and add an association between the created xml file and the ant editor.
- Now, as long as the file has a .xml extension, Eclipse will consider it to be a possible Ant buildfile, and will enable Ant-related actions when it is selected. Until a file has Ant buildfile content, you will need to open it using Open With > Ant Editor.

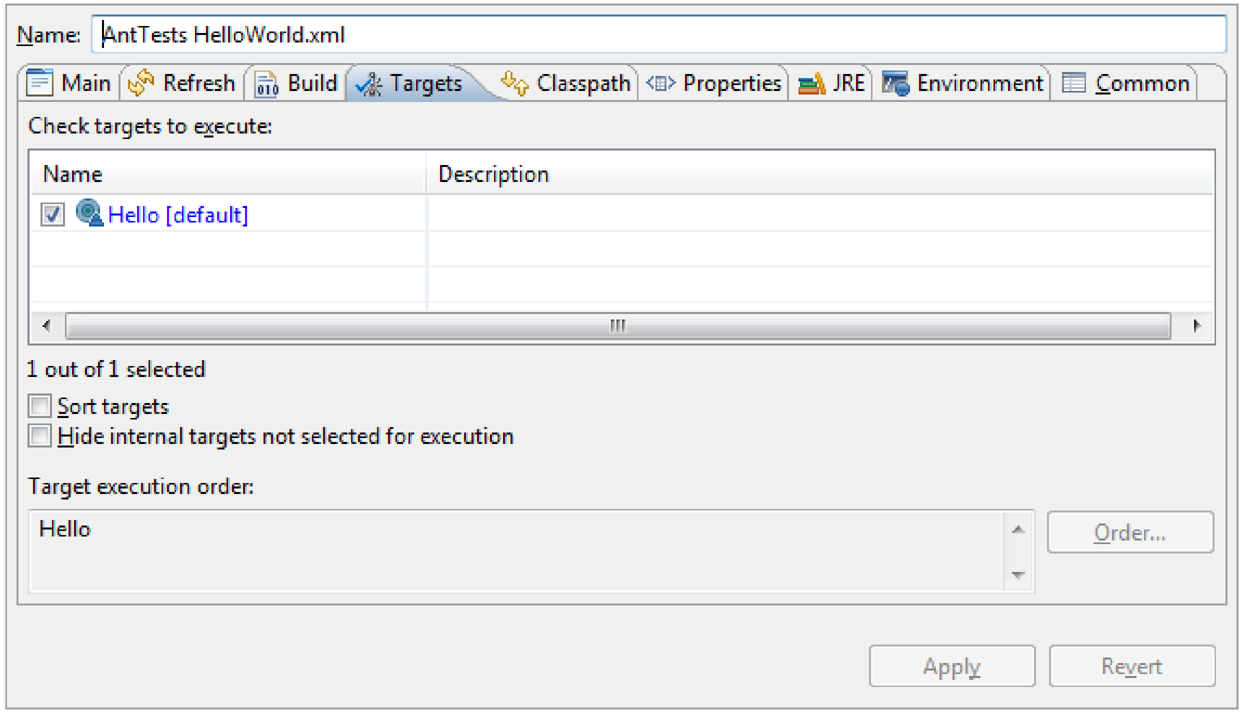
- Open the xml file created with the Ant Editor and add the contents as shown in Fig 1.
- Notice the syntax coloring for property values.
- Save the buildfile contents.
- Select the file and run as AntBuild from its context menu. A dialog appears as shown in Fig 2.
- This dialog allows the configuration of many aspects of the way an Ant buildfile is run, but for now concentrate on the Targets tab which allows the selection of which Ant targets to run and their order. Select both targets and leave the order as the default. Click on Run and the output is displayed in the console view.



HelloWorld.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Hello World" default="Hello" basedir=".">
  <property name="HelloText" value="Hello"/>
  <target name="Hello">
    <echo>${HelloText}</echo>
  </target>
</project>
```

Fig 1: Contents in the xml file



Name: AntTests HelloWorld.xml

Check targets to execute:

Name	Description
☑ Hello [default]	

1 out of 1 selected

☐ Sort targets
☐ Hide internal targets not selected for execution

Target execution order:

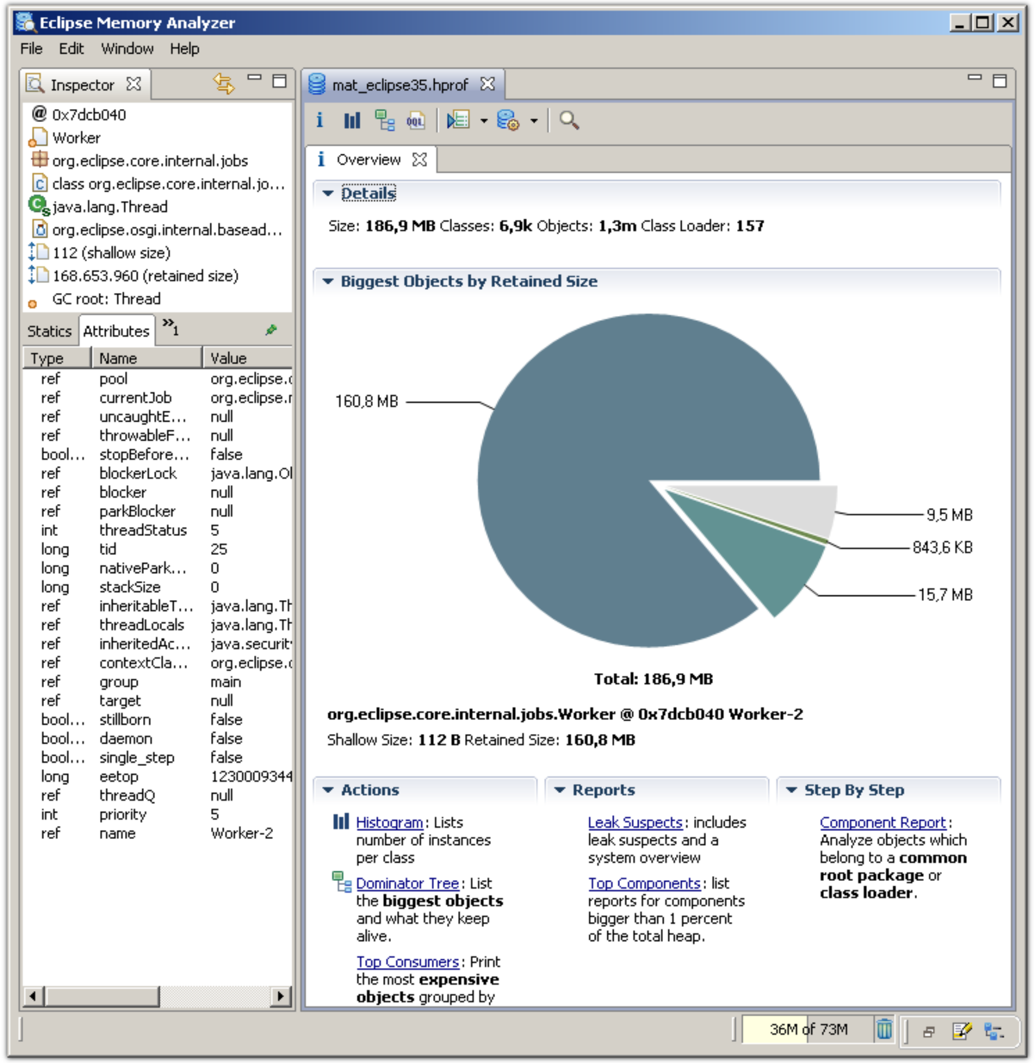
Hello

Fig 2: Dialog box to run the ant build file

Memory Analyzer in Eclipse

- The Eclipse Memory Analyzer is a fast and feature-rich Java heap analyzer that helps you find memory leaks and reduce memory consumption.

- Use the Memory Analyzer to analyze productive heap dumps with hundreds of millions of objects, quickly calculate the retained sizes of objects, see who is preventing the Garbage Collector from collecting objects, run a report to automatically extract leak suspects.
- Eclipse represents the memory utilized at that point in time by a process in a graphical representation as shown in Fig.
- For more information check this.



0x7dcb040
Worker
org.eclipse.core.internal.jobs
class org.eclipse.core.internal.jo...
java.lang.Thread
org.eclipse.osgi.internal.basead...
112 (shallow size)
168.653.960 (retained size)
GC root: Thread

Type	Name	Value
ref	pool	org.eclipse.c
ref	currentJob	org.eclipse.r
ref	uncaughtE...	null
ref	throwableF...	null
bool...	stopBefore...	false
ref	blockerLock	java.lang.Ol
ref	blocker	null
ref	parkBlocker	null
int	threadStatus	5
long	tid	25
long	nativePark...	0
long	stackSize	0
ref	inheritableT...	java.lang.Th
ref	threadLocals	java.lang.Th
ref	inheritedAc...	java.securit
ref	contextCla...	org.eclipse.c
ref	group	main
ref	target	null
bool...	stillborn	false
bool...	daemon	false
bool...	single_step	false
long	eetop	1230009344
ref	threadQ	null
int	priority	5
ref	name	Worker-2

Size: **186,9 MB** Classes: **6,9k** Objects: **1,3m** Class Loader: **157**

Total: 186,9 MB

org.eclipse.core.internal.jobs.Worker @ 0x7dcb040 Worker-2
Shallow Size: **112 B** Retained Size: **160,8 MB**

Histogram: Lists number of instances per class

Dominator Tree: List the **biggest objects** and what they keep alive.

Top Consumers: Print the most **expensive objects** grouped by

Leak Suspects: includes leak suspects and a system overview

Top Components: list reports for components bigger than 1 percent of the total heap.

Component Report: Analyze objects which belong to a **common root package** or **class loader**.

LTTng Plug-in for Eclipse

- LTTng - Linux Trace Toolkit, next generation.
- This is a highly efficient tracing tool for Linux that can be used to track down kernel and application performance issues as well as troubleshoot problems involving multiple concurrent processes and threads.
- It consists of a set of kernel modules, daemons - to collect the raw tracing data - and a set of tools to control, visualize and analyze the generated data. It also provides support for user space application instrumentation.
- The LTTng Eclipse plug-in has a number of features to allow efficient handling of very large traces
  - Support for arbitrarily large traces (larger than available memory)
  - Support for correlating multiple time-ordered traces
  - Support for zooming down to the nanosecond on any part of a trace or set of traces
  - Views synchronization of currently selected event
  - Efficient searching and filtering of events
  - Support for trace bookmarks
- For more information about LTTng, refer to the project site.

Callgraph plug-in for Eclipse

- A graphical explorer for call and class hierarchies. Designed to help understand complex relations in larger scale applications.
- This plug-in allows you to profile C/C++ projects directly within the Eclipse IDE, providing various runtime details such as:
  - The relationship between function calls
  - Number of times each function was called
  - Time taken by each instance of a function (relative to the program's execution time)
  - Time taken by all instances of a function (relative to program's execution time)
- Eclipse Callgraph uses SystemTap to perform function traces. For more information about SystemTap, refer to the here.
- This leverages the internal platform Call Hierarchy and Search mechanisms.
- For more information check this.

References:

- http://www.eclipse.org
- http://help.eclipse.org/mars/index.jsp
- http://www.csee.umbc.edu/courses/undergraduate/341/fall08/Lectures/Eclipse
- http://homepage.cs.uiowa.edu/~sriram/21/fall08
- http://objectaid.com
- http://www.eclipse.org/egit
- http://ant.apache.org
- https://marketplace.eclipse.org/content/callgraph-viewer#sthash.CVhMjnkx.dpuf