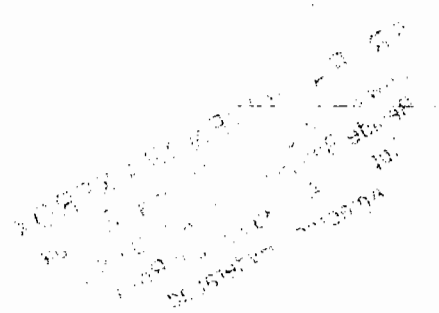
JAVA

REAL TIME TOOLS

1. Junit
2. Log4J
3. MAVEN
4. Svn
5. JASPER Reports

BY

Sekhar. Sir



JUnit

JUnit

&

Mockito

By: Sekhar

Sathya Technologies

Fundamentals of Testing

Testing

Testing is the process of checking the functionality of the application whether it is working as per requirements.

Unit Testing

Unit testing is the testing of single entity (class or method). Unit testing is very essential to every software company to give a quality product to their customers.

Unit testing can be done in two ways:

- Manual Testing
- Automated Testing

Unit Test Case

- A Unit Test Case is a part of code which ensures that another part of code (method) works as expected.
- A formal written test-case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a pre-condition and the expected output should test a post-condition.

Note: There must be at least two test cases for each requirement: one positive test and one negative test.

A unit test is a test of a single isolated component in a repeatable way. A test consists of four phases:

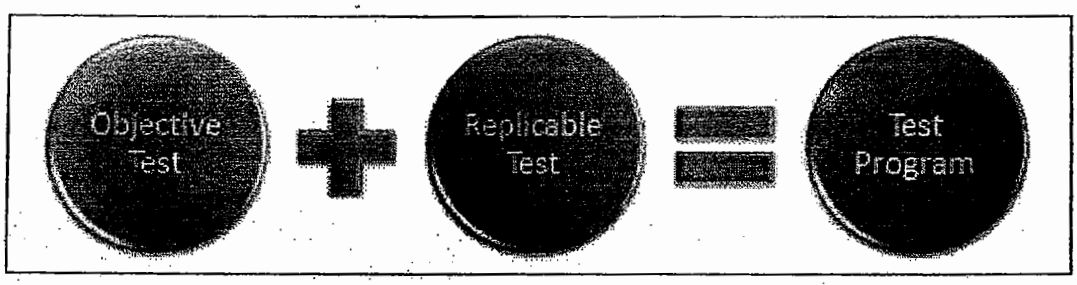
Phase	Description
Prepare	Sets up a baseline for testing and defines the expected results
Execute	Running the test
Validate	Validates the results of the test against previously defined expectations
Reset	Resets the system to the way it was before Prepare

Note: JUnit is a popular framework for creating unit tests for Java. It provides a simple yet effective API for the execution of all four phases of a unit test.

Understanding Unit Testing Frameworks

All unit testing frameworks should observe the following:

- Rule 1: Each unit test must run ***independently*** of all other unit tests.
- Rule 2: Errors must be ***detected*** and reported test by test.
- Rule 3: It must be easy to ***define*** which unit tests will run.



JUnit Framework

- ✓ JUnit is a Java-based unit testing framework. It has been around for a while.
- ✓ Its stability as open source project and its maturity and acceptance by many software groups has also lead to large amount of knowledge sharing in order to maximize the usefulness of JUnit.
- ✓ JUnit promotes the idea of "first testing then coding", which emphasis on setting up the test data for a piece of code which can be tested first and then can be implemented.

JUnit Features

1. JUnit is an open source framework which is used for writing & running tests.
2. Provides Annotation to identify the test methods.
3. Provides Assertions for testing expected results.
4. Provides Test runners for running tests.
5. JUnit tests allow you to write code faster which increasing quality
6. JUnit is elegantly simple. It is less complex & takes less time.
7. JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
8. JUnit tests can be organized into test suites containing test cases and even other test suites.
9. JUnit shows test progress in a bar that is green if test is going fine and it turns red when a test fails.

JUnit Design Goals

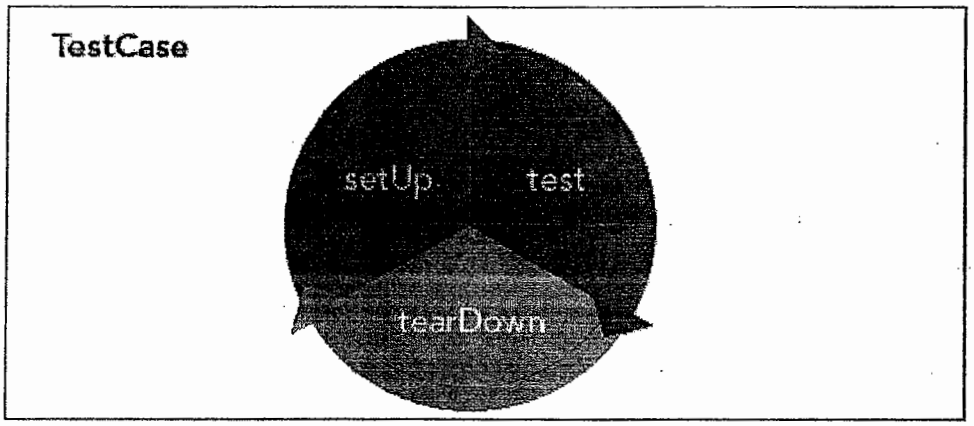
The design goals of JUnit are as follows:

- ✓ The framework must help to write ***useful*** tests.
- ✓ The framework must help us ***lower the cost*** of writing tests by reusing the code.
- ✓ The framework must help us create tests that ***retain value over time***.

JUnit Lifecycle

A JUnit test case can contain many test methods. Each method identified as a test will be executed within the JUnit test lifecycle.

Thelifecycle consists of three pieces: setup, test and teardown, all executed in sequence.



Lifecycle Phase	Method call	Method description
Setup	public void setUp()	Called to do any required preprocessing before a test. Examples include instantiating objects and inserting test data into a database
Test	public void testAdd()	Each test method is called once within the test lifecycle. It performs all required testing. Test results are recorded by JUnit for reporting to the test runner upon completion.
Teardown	public void tearDown()	Called to do any required post processing after a test. Examples include cleaning up of database tables and closing database connections.

Writing the Test Case

Creating a test case with the JUnit framework requires the following:

- ✓ The test class does not need to extend any particular class.
- ✓ Unit test methods to be marked by @Test annotation.
- ✓ All unit test methods to be public void and take no parameters.
- ✓ Test methods to make assert calls to validate the outcome.

Sample example to create JUnitTestCase

Class to be tested:

```
 1  package com.pack.example1;
 2
 3  public class Calculator {
 4
 5          public int add(int value1, int value2){
 6                  return value1 + value2;
 7          }
 8
 9          public int sub(int value1, int value2){
10                  return value1 - value2;
11          }
12  }
```

Test Case using JUnit:

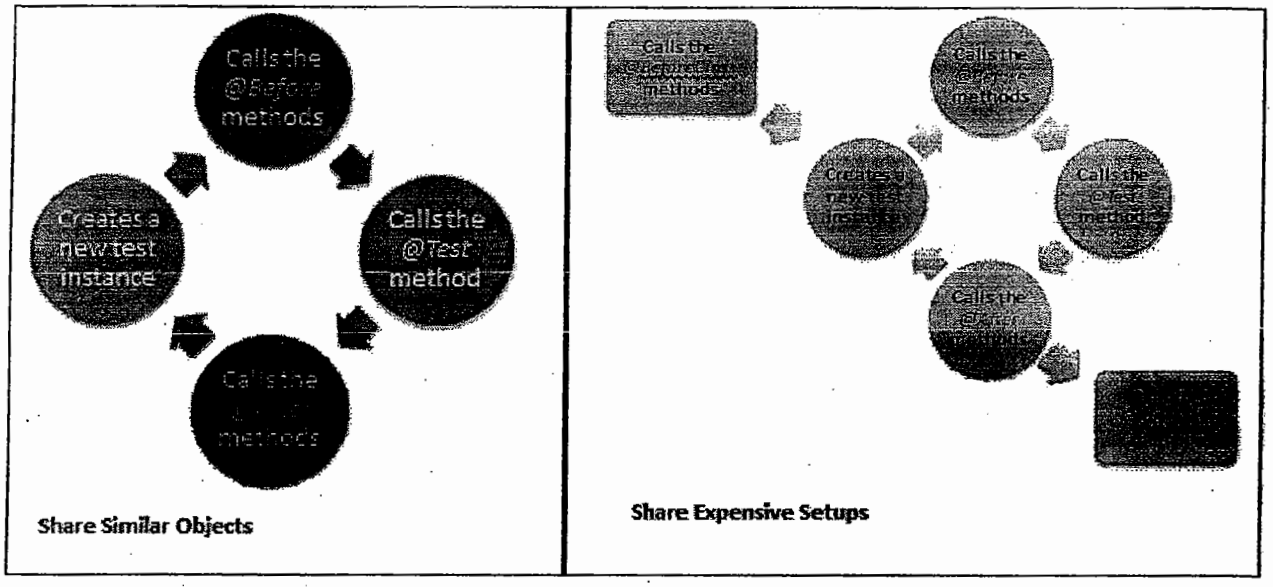
```
 1  package com.pack.example1;
 2
 3  import static org.junit.Assert.*;
 4  import org.junit.Test;
 5
 6  public class CalculatorTest4
 7  {
 8          @Test
 9          public void testAdd()
10          {
11                  int total = 8;
12                  Calculator cal = new Calculator();
13                  int sum = cal.add(5, 3);
14                  assertEquals(sum, total);
15          }
16  }
```

JUnit Test Fixtures

- There should be a well-known and fixed environment in which tests are run so that results are repeatable.
- Examples:
  - Loading a database with a specific known set of data
  - Copying a specific known set of files
  - Preparation of input data and setup or creation of fake or mock objects
- A test fixture is a fixed state of a set of objects used as a baseline for running tests.

Managing Resources with Fixtures

- Similar objects shared by several tests can be initialized and reclaimed using public void methods.
- You should annotate the public void method with
  - @BeforeClass – run before any test has been executed
  - @AfterClass – run after all the tests have been executed
  - @Before – run before each test
  - @After – run after each test



Example for sharing similar objects:

```
1  public class CalculatorTest4
2  {
3          private int value1, value2;
4          @Before
5          public void setUp() throws Exception {
6                  value1 = 5;
7                  value2 = 3;
8          }
9          @After
10         public void tearDown() throws Exception {
11                 value1 = 0;
12                 value2 = 0;
13         }
14         @Test
15         public void testAdd() {
16           . . . .
17         }
18 }
```

Example for sharing expensive setups:

```
 1 public class CalculatorTest4
 2 {
 3         private Calculator cal = null;
 4         @BeforeClass
 5         public void setUp() throws Exception {
 6                 cal = new Caluculator();
 7         }
 8         @AfterClass
 9         public void tearDown() throws Exception {
10                 cal = null;
11         }
12         @Test
13         public void testAdd() {
14           . . . .
15         }
16 }
```

Assertions

The asset methods are defined in org.junit.Assert class

Assertion	Description
assertNull(Object x)	Validates that the parameter is null
assertNotNull(Object x)	Validates that the parameter is not null
assertTrue(boolean x)	Validates that the parameter is true
assertFalse(boolean x)	Validates that the parameter is false
assertEquals(Object x, Object y)	Validates that the two objects passed are equal based on the .equals(Object obj1, Object obj2) method
assertSame(Object x, Object y)	Validates that the two objects passed are equal based on the == operator
assertNotSame(Object x, Object y)	Validates that the two objects passed are not equal based on the == operator
fail()	Programmatically fail the test.

Running Tests

- When tests are received, the user will want to run them.
- JUnit provides tools to define the suite to be run and to display its results.
- JUnitCore is a façade for running tests. It supports running JUnit 4 tests, JUnit 3 tests and mixtures.
- To run tests and see the results on the console, run this from a Java program:
  org.junit.runner.JUnitCore.runClasses(TestClass1.class, . . .);
- Or this from the command line, with both your test class and junit on the classpath:
  java org. junit.runner.JUnitCore TestClass1[. . . other test classes . . .]

JUnit 4 Annotations

JUnit 4 added annotations to the framework and eliminated the need to extend TestCase. You can direct both the lifecycle events and other aspects of the test execution with the provided annotations.

Annotation	Parameters	Use
@After	None	Method will be executed after each test method (similar to the tearDown() method inJUnit 3.x). Multiple methods may be tagged with the @After annotation, however no order is guaranteed.
@AfterClass	None	Method will be executed after all of the test methods and teardown methods have been executed within the class. Multiple methods may be tagged with the @AfterClassannotation, however no order is guaranteed.
@Before	None	Method will be executed before each test method (similar to the setUp() method inJUnit 3.x). Multiple methods may be tagged with the @Before annotation, however no order is guaranteed.
@BeforeClass	None	Executed before any other methods are executed within the class. Multiple methods may be tagged with the @BeforeClassannotation, however no order is guaranteed.
@Ignore	String (optional)	Used to temporarily exclude a test method from test execution. Accepts an optional String reason parameter.
@Test	• Class(optional) • Timeout(optional)	Used to indicate a test method. Same functionality as naming a method public void testXYZ() in JUnit 3.x. The class parameter is used to indicate an exception is expected to be thrown and what the exception is. The timeout parameter specifies in milliseconds how long to allow a single test to run. If the test takes longer than the timeout, it will be considered a failure.

Let's create a sample JUnit test case for a Calculator application using Eclipse. Steps to create JUnit test case in Eclipse:

1. Create a new Java project in eclipse (File -> New -> Java Project).
2. Provide project name and click on finish in "Create a Java Project" window.
3. Right click on "src" folder and select "class" option.
4. Provide class name as "Calculator" and click finish in "New Java Class" window.
5. Write below code in Calculator.java

```
 1 package com.pack.example1;
 2
 3 public class Calculator {
 4
 5         public int add(int value1, int value2){
 6                 return value1 + value2;
 7         }
 8
 9         public int sub(int value1, int value2){
10                 return value1 - value2;
11         }
12 }
```

6. Right click on "src" folder and select "JUnit Test Case" option.
7. Provide test case name as "CalculatorTest3" and select JUnit3.x option and select "setup()" & "tearDown()" checkboxes in "New JUnit Test Case" window and click finish.
8. A new popup will be prompted to add JUnit libraries, Click Ok.
9. The JUnit libraries will be automatically added to project by eclipse.
10. Update and add code as per below in CalculatorTest.java

```
 1 package com.pack.example1;
 2
 3 import junit.framework.TestCase;
 4
 5 import com.pack.example1.Calculator;
 6
 7 public class CalculatorTest3 extends TestCase {
 8
 9         private int value1, value2;
10
11         protected void setUp() throws Exception {
12                 super.setUp();
13                 value1 = 5;
14                 value2 = 3;
15         }
16
17         protected void tearDown() throws Exception {
18                 super.tearDown();
19                 value1 = 0;
20                 value2 = 0;
21         }
22
23         public void testAdd() {
24                 int total = 8;
25                 Calculator cal = new Calculator();
26                 int sum = cal.add(value1, value2);
27                 assertEquals(sum, total);
28         }
29 }
```

11. To run the above JUnit test case, right click on the test case class and select "Run As" option and select "JUnit Test" option.
12. The output of the test case will be shown in JUnit view.
13. Green colour indicates successful execution of test case and red colour indicates errors in test case execution.

Let's create CalculatorTest4 class using JUnit 4.x feature (using steps 7, 8, 9 in above). Update the code as shown below:

```java
package com.pack.example1;

import static org.junit.Assert.*;

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

public class CalculatorTest4 {

	private int value1, value2;

	@Before
	public void setUp() throws Exception {
		value1 = 5;
		value2 = 3;
	}

	@After
	public void tearDown() throws Exception {
		value1 = 0;
		value2 = 0;
	}

	@Test
	public void testAdd() {
		int total = 8;
		Calculator cal = new Calculator();
		int sum = cal.add(value1, value2);
		assertEquals(sum, total);
	}

}
```

//Coupon.java (Pojo class)

```
package com.sathya.sekhar.junit;
import java.util.Date;
public class Coupon
{
	private Date startDate;
	private Date endDate;
	// setters & getters
	public void setStartDate(Date startDate)
	{
		this.startDate=startDate;
	}
	public Date getStartDate()
	{
		return startDate;
	}
	public void setEndDate(Date endDate)
	{
		this.endDate=endDate;
	}
	public Date getEndDate()
	{
		return endDate;
	}
};
```

```
//CouponValidationTest.java
package com.sathya.sekhar.junit;
import java.util.*;
import org.junit.*;
public class CouponValidationTest {
        ValidateCoupons valCoupons;
        @Before
        public void setUp() throws Exception {
                valCoupons = new ValidateCoupons();
        }
        @After
        public void tearDown() throws Exception {
                valCoupons=null;
        }
        @Test
        public void testValidityOfCoupons() {
                //Create expired coupon (Today > Coupon_enddate) (Invalid Coupon)
                Coupon cp1 = new Coupon();
                Calendar startCal = Calendar.getInstance();
                Calendar endCal = Calendar.getInstance();
                startCal.set(2015,9, 13);
                endCal.set(2015,9,28);
                cp1.setStartDate(startCal.getTime());
                cp1.setEndDate(endCal.getTime());
                //Create future coupon (Today < Coupon_StartDate) (Invalid Coupon)
                Coupon cp2 = new Coupon();
                startCal.set(2015,11,1);
                endCal.set(2015,11,31);
                cp2.setStartDate(startCal.getTime());
                cp2.setEndDate(endCal.getTime());
                //Create a coupon with start date as null & Today > Coupon_enddate (Invalid Coupon)
                Coupon cp3 = new Coupon();
                endCal.set(2015,10,12);
                cp3.setStartDate(null);
                cp3.setEndDate(endCal.getTime());

                //Create a coupon with end date as null & Today > Coupon_StartDate (Valid Coupon)
                Coupon cp4 = new Coupon();
                startCal.set(2015,10,1);
                cp4.setStartDate(startCal.getTime());
                cp4.setEndDate(null);
                //Create a coupon with startdate < today < end date (Valid Coupon)
                Coupon cp5 = new Coupon();
                startCal.set(2015,10,1);
                endCal.set(2015,11,31);
                cp5.setStartDate(startCal.getTime());
                cp5.setEndDate(endCal.getTime());
                List<Coupon> cpList = new ArrayList<Coupon>();
```

Test begin

Test end

Business class

reference variable

→ 13·10·2015 } zero based index

→ 28·10·2015

→ 1·12·2015

→ 31·12·2015

→ 12·11·2015

→ 1·11·2015

→ 1·11·2015

→ 31·12·2015

```
            cpList.add(cp1);
            cpList.add(cp2);
            cpList.add(cp3);
            cpList.add(cp4);
            cpList.add(cp5);
            List<Coupon> validCouponList = valCoupons.validityOfCoupons(cpList);
            Assert.assertEquals(2, validCouponList.size());
        }
}
```

//ValidateCoupons.java (Business logic)

```
package com.sathya.sekhar.junit;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
public class ValidateCoupons {
   public List<Coupon> validityOfCoupons(List<Coupon> coupons)
            {
     Date now = new Date(System.currentTimeMillis());
     List<Coupon> couponsToRemove = new ArrayList<Coupon>();
            //for each loop
     for (Coupon coupon : coupons) {
                    //If the coupon is valid for future date with startdate as not null then add to
coupons to remove list.
       if ((coupon.getStartDate() != null) && (coupon.getStartDate().after(now))){
         couponsToRemove.add(coupon);
       }
                    //if coupon is already expired then remove that coupon
                    else if (coupon.getEndDate() != null && coupon.getEndDate().before(now)){
         couponsToRemove.add(coupon);
       }
     }
            // remove all the non valid coupons from the original list
     for (Coupon coupon : couponsToRemove) {
       coupons.remove(coupon);
     }
            //return only valid coupons
     return coupons;
   }
```

* In the above business method we have define business logic like the following:
(1) It finds whether coupon is invalid.
(2) If yes then that coupon is added to couponsRemove list.
(3) Each coupon is taken from couponsToRemove list and then that coupon is removed from coupons list.
(4) Finally it returns only valid coupons list.

Overview of Mockito framework

Mockito is used to mock java classes for testing purposes.

Suppose you write an explanation which will need to connect to DB and update tables and also hit an http server, both of which are either cumbersome to set-up or are managed by someone else. Even then, you need to be able to test your application. Mockito, or other mocking frameworks like easymock come to your rescue.

When you mock a class, you are actually creating a dummy or mock of that class, and then re-defining method to bypass the actual db connection/http connection and instead return dummy responses or objects which you are expecting the db layer/http server to return anyways. If your application works fine with these dummy objects, then when the actual db layer/http server are plugged in, the application will work as expected.
Mocking frameworks are generally used with unit testing api's like junit to test applications modularly.

To start using Mockito api
For using mockito, the only requirement is to have mockito-all-<version>.jar in the classpath. Since, mocking is usually done for unit tests, you will usually also use a unit test api like JUnit, Testing etc.

How to create a mock object

This example shows how to create a dummy or mock for an object. A mock object of Calendar class is created by using the method mock(...) of class org.mockito.Mockito.

```
package com.sathya.sekhar.mockito;
import java.util.Calendar;
import static org.mockito.Mockito.*;
public class CreateMocks
{
   public static void main(String[] args)
   {
      Calendar mockedCalendar = mock(Calendar.class);
      when(mockedCalendar.get(Calendar.YEAR)).thenReturn(2020);
System.out.println(mockedCalendar.get(Calendar.YEAR));
   }
}
```

How to stub method to accept any argument

You can use methods like anyInt() of class org.mockito.Mockito to set the mock objects behaviour when it is called with any integer as argument.

```
package com.sathya.sekhar.mockitog;
import static org.mockito.Mockito.*;
import java.util.ArrayList;
import java.util.List;

public class CustomizeMethodBehaviourAdvanced
{
   public static void main(String[] args)
   {
      List myMockedList = mock(ArrayList.class);
      when(myMockedList.get(anyInt())).thenReturn(5);
      when(myMockedList.isEmpty()).thenReturn(false);

      System.out.println(myMockedList.get(1));
      System.out.println(myMockedList.isEmpty());
   }
}
```

How to stub method to throw exception

A mocked object can also be asked to throw an exception when particular methods are called on it. In the example below, the mock object is stubbed to throw NullPointerException when the method get(..) is called on it. The method used for this is thenThrow(..) of class org.mockito.Mockito.

If we need to throws exception when a method whose return type is void is called (eg. List.clear()), then we can use the alternate way of throwing exception , i.e. doThrow(..) of class org.mockito.Mockito

```
package com.sathya.sekhar.mockito;
import static org.mockito.Mockito.*;
import java.util.ArrayList;
import java.util.List;

public class ThrowException
{
   public static void main(String[] args)
   {
      List myMockedList = mock(ArrayList.class);

      when(myMockedList.get(anyInt())).thenThrow(new NullPointerException());
      doThrow(new RuntimeException()).when(myMockedList).clear();

      System.out.println(myMockedList.get(1));
      myMockedList.clear();
   }
}
```

Example 1

```
//interface
//ICityTemp.java
public interface ICityTemp
{
    void setCity(String city);
    String getCity();
    void setTemparature(double temparature);
    double getTemparature();
}
```

Dependancy is an interface.

```
//MyTest.java(Testcase)
import org.junit.*;
import org.mockito.Mockito;
public class MyTest
{
    private Forecast forecast;
    private ICityTemp icityTemp;
    @Before
        public void setUp()
    {
        //creating mock object using Mockito
        icityTemp=Mockito.mock(ICityTemp.class);
        forecast=new Forecast(icityTemp);
    }
    @Test
        public void testWeatherForecast()
    {
        //training the mock object
        Mockito.when(icityTemp.getCity()).thenReturn("Chennai");
        Mockito.when(icityTemp.getTemparature()).thenReturn(35.0);
        String result=forecast.weatherForecast();
        Assert.assertEquals("Chennai has medium temparature",result);
    }
    @After
        public void tearDown()
    {
        forecast=null;
    }
};
```

static method

mock object for interface

dummy object / mock object

Static method

static method

class mock object

Expected value

original value

```
//Forecast.java
public class Forecast
{
    private ICityTemp icityTemp;
    public Forecast(ICityTemp icityTemp)
    {
        this.icityTemp=icityTemp;
    }
```

This is our class we want to test.

Dependancy is an interface, in order to provide loose coupling

```
    public String weatherForecast() → This is the business logic that we are testing.
    {                          interface
            String  city=icityTemp.getCity();
            double  temparature=icityTemp.getTemparature();
            String str="";
            if(temparature<=30)
                    str= city+" has minimum temparature";
            else if(temparature>30 && temparature<=40)
                    str=city+" has medium temparature";
            else
                    str=city+" has high temparature";
            return str;
    }
};
```

Example 2

```
//interface
public interface  ICityTemp
{
    void setCity(String city);
    String getCity();
    void setTemparature(double temparature);
    double getTemparature();
}

//Forecast.java
public class Forecast
{
    private ICityTemp icityTemp;
    public Forecast(ICityTemp icityTemp)
    {
            this.icityTemp=icityTemp;
    }
    public String weatherForecast()
    {
            String  city=icityTemp.getCity();
            double  temparature=icityTemp.getTemparature();
            String str="";
            if(temparature<=30)
                    str= city+" has minimum temparature";
            else if(temparature>30 && temparature<=40)
                    str=city+" has medium temparature";
            else
                    str=city+" has high temparature";
            return str;
    }
};
```

```
//MyTest.java (Testcase)
import org.junit.*;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.Mock;
public class MyTest
{
    private Forecast forecast;
    @Mock → mock object created through annotation.
    private ICityTemp icityTemp;
    @Before
        public void setUp()
    {
        MockitoAnnotations.initMocks(this); → This statement enables @mock annotation
        forecast=new Forecast(icityTemp);
    }
    @Test
        public void testWeatherForecast()
    {
        Mockito.when(icityTemp.getCity()).thenReturn("Hyderabad");
        Mockito.when(icityTemp.getTemparature()).thenReturn(35.0);
        String result=forecast.weatherForecast();
        Assert.assertEquals("Hyderabad has minimum temparature",result);
    }
    @After
        public void tearDown()
    {
        forecast=null;
    }

};
```

Log4J

By: Sekhar

Sathya Technologies

Log4j

Log4j is a Simple, Reliable, Fast and Flexible Logging Framework (APIs) written in Java which is distributed under the Apache Software License.

- Log4j is being used by C, C++, C#, Perl, Python, Ruby etc.
- Log4j is configurable through external configuration files at runtime.
- It views the logging process in terms of levels of priorities.
- Provides mechanisms to logging information to a variety of destinations, such as a database, file, console etc.
- Supports logging information in different formats.

The Three Main Components of Log4j

1. **Loggers**: Responsible for capturing logging information.
2. **Appenders**: Responsible for publishing logging information to various preferred destinations.
3. **Layouts**: Responsible to format logging information in different styles.

Log4j Features

- Log4j is thread-safe.
- Log4j is optimized for performance.
- Log4j supports multiple output appenders per logger.
- Logging behaviour can be set at runtime using a configuration file.
- Log4j uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
- The format of the log output can be easily changed by extending the Layout class.

Log4j Libraries

Log4j API package is distributed under the Apache Software License, an open source license certified by the open source initiative.

The latest log4j version, including full-source code, class files and documentation can be found at http://logging.apache.org/log4j/.

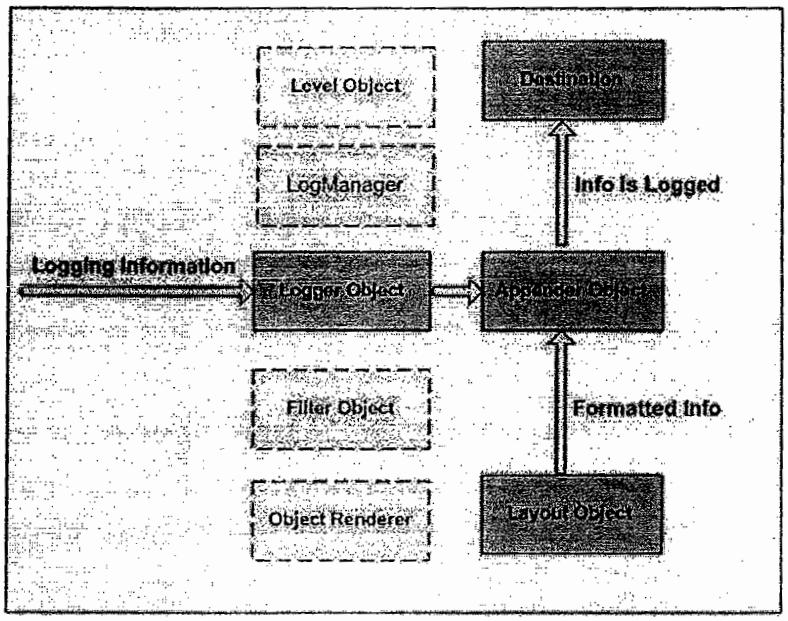
Name	Last modified	Size	Description
Parent Directory		-	
1.0.4/	2004-04-19 12:51	-	
1.1.3/	2004-04-19 12:50	-	
1.2.1/	2004-04-19 12:57	-	
1.2.10/	2005-05-27 05:30	-	

Log4j Architecture

Log4j API has been designed in layered architecture where each layer provides different objects which performs different tasks. This makes design flexible and very much extendable in future based on need.

There are two types of objects available with Log4j framework:

1. **Core Objects**: These are mandatory objects of the framework and required to use the framework.
2. **Support Objects**: These are optional objects of the framework and support core objects to perform addition but important tasks.



Core Objects

- **Logger Object** - The top level layer is Logger which provides Logger object. The Logger object is responsible for capturing logging information and they are stored in a namespace hierarchy.
- **Layout Object** - The layer provides objects which are used to format logging information in different styles. Layout layer provides support to appender objects to before publishing logging information.
- **Appender Object**-This is lower level layer which provides Appender object. The Appender object is responsible for publishing logging information to various preferred destinations such as a database, file, console, UNIX Syslog etc.

Support Objects

- **Level Object** - The Level object defines the granularity and priority of any logging information. There are seven levels of logging defined within the API: OFF, DEBUG, INFO, ERROR, WARN, FATAL, and ALL.
- **Filter Object** - The Filter object is used to analyze logging information and make further decisions on whether that information should be logged or not.

  An Appender objects can have several Filter objects associated with them. If logging information is passed to a particular Appender object, all the Filter objects associated with that Appender need to approve the logging information before it can be published to the attached destination.
- **ObjectRenderer** - The ObjectRenderer object is specialized in providing a String representation of different objects passed to the logging framework. This object is used by Layout objects to prepare the final logging information.
- **LogManager**-The LogManager object manages the logging framework. It is responsible for reading the initial configuration parameters from a system-wide configuration file or a configuration class.

Logger

Logger class provides a variety of methods to handle logging activities. The Logger class does not allow us to instantiate a new Logger instance but it provides two static methods for obtaining a Logger object:

- public static Logger getRootLogger();
- public static Logger getLogger(String name);

Logging Methods

Once we obtain an instance of a named logger, we can use several methods of the logger to log messages. The Logger class has the following methods for printing the logging information.

SN	Methods with Description
1	**public void debug(Object message)** This method prints messages with the level Level.DEBUG.
2	**public void error(Object message)** This method prints messages with the level Level.ERROR.

3	**public void fatal(Object message);** This method prints messages with the level Level.FATAL.
4	**public void info(Object message);** This method prints messages with the level Level.INFO.
5	**public void warn(Object message);** This method prints messages with the level Level.WARN.
6	**public void trace(Object message);** This method prints messages with the level Level.TRACE.

The org.apache.log4j.Level class provides following levels:

Level	Description
ALL	All levels including custom levels.
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates very severe error events that will presumably lead the application to abort.
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.
OFF	The highest possible rank and is intended to turn off logging.
TRACE	Designates finer-grained informational events than the DEBUG.
WARN	Designates potentially harmful situations.

How Level Works?

A log request of level p in a logger with level q, is enabled if p >= q. This rule is at the heart of log4j. It assumes that levels are ordered. For the standard levels, we have ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

Appenders

Apache log4j provides Appender objects which are primarily responsible for printing logging messages to different destinations such as consoles, files, database tables etc.

Each Appender object has different properties associated with it, and these properties indicate the behavior of that object.

Property	Description
layout	Appender uses the Layout objects and the conversion pattern associated with them to format the logging information.
target	The target may be a console, a file, or another item depending on the appender.
level	The level is required to control the filteration of the log messages.
threshold	Appender can have a threshold level associated with it independent of the logger level. The Appender ignores any logging messages that have a level lower than the threshold level.
filter	The Filter objects can analyze logging information beyond level matching and decide whether logging requests should be handled by a particular Appender or ignored.

If you are willing to add Appender object inside your program then you can use following method:

public void addAppender(Appenderappender);

The addAppender() method adds an Appender to the Logger object. It is possible to add many Appender objects to a logger in a comma-separated list, each printing logging information to separate destinations.

The following are some of appender options are:

- ConsoleAppender
- DailyRollingFileAppender
- FileAppender
- JDBCAppender
- JMSAppender
- RollingFileAppender
- SocketAppender

Layout

Apache log4j provides various Layout objects, each of which can format logging data according to various layouts. It is also possible to create a Layout object that formats logging data in an application-specific way.

Layout Types

The top-level class in the hierarchy is the abstract class org.apache.log4j.Layout. This is the base class for all other Layout classes in the log4j API.

The Layout class is defined as abstract within an application, we never use this class directly; instead, we work with its subclasses which are as follows:

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

To write your logging information into a file you would have to use *org.apache.log4j.FileAppender.* There are following configurable parameters of FileAppender:

Property	Description
immediateFlush	This flag is by default set to true, which means the output stream to the file being flushed with each append operation.
encoding	It is possible to use any character-encoding. By default is the platform-specific encoding scheme.
threshold	The threshold level for this appender.
Filename	The name of the log file.
fileAppend	This is by default set to true, which mean the logging information being appended to the end of the same file.
bufferedIO	This flag indicates whether we need buffered writing enabled. By default is set to false.
bufferSize	If bufferedI/O is enabled, this indicates the buffer size. By default is set to 8kb.

Log4j Configuration

Configuring log4j involves assigning the Level, defining Appender, and specifying Layout objects in a configuration file.

There are many ways to configure the Log4j framework. They are:

- BasicConfigurator
- PropertyConfigurator
- DOMConfigurator

BasicConfigurator

The simple way to configure log4j is by using BasicConfigurator.configure() method. The BasicConfigurator.configure() method by default it uses the ConsoleAppender and PatternLayout for all the loggers. This will log all messages on the console.

```
1  import org.apache.log4j.BasicConfigurator;
2  import org.apache.log4j.Logger;
3
4  public class LogExample1 {
5
6    static final Logger logger = Logger.getLogger(LogExample1.class);
7
8    public static void main(String[] args)
9    {
10     BasicConfigurator.configure();
11     logger.trace("This is a Trace Message!");
12     logger.debug("This is a Debug Message!");
13     logger.info("This is a Info Message!");
14     logger.warn("This is a Warn Message!");
15     logger.error("This is a Error Message!");
16     logger.fatal("This is a Fatal Message!");
17   }
18 }
```

PropertyConfigurator

Log4j provides you configuration file based level setting which puts you free from changing source code when you want to change debugging level.

You need to use the PropertyConfigurator.configure() method to configure log4j using a properties file. Log4j should be configured only once for the entire application.

The log4j.properties file is a log4j configuration file which keeps properties in key-value pairs.

```
1  import org.apache.log4j.Logger;
2  import org.apache.log4j.PropertyConfigurator;
3
4  public class LogExample2 {
5
6    static final Logger logger = Logger.getLogger(LogExample2.class);
7
8    public static void main(String[] args) throws Exception
9    {
10     PropertyConfigurator.configure("log4j.properties");
11     logger.trace("This is a Trace Message!");
12     logger.debug("This is a Debug Message!");
13     logger.info("This is a Info Message!");
14     logger.warn("This is a Warn Message!");
15     logger.error("This is a Error Message!");
16     logger.fatal("This is a Fatal Message!");
17   }
18 }
```

Below is the "log4.properties" file for console appender. In this case logger messages are written to console.

```
1  #Define the root logger with appender console
2  log4j.rootLogger=DEBUG, CA   reference of appender
3        Log statement from Debug to Fatal are enable
4  #Define the console appender
5  log4j.appender.CA=org.apache.log4j.ConsoleAppender
6                                  It prints log stmts of console
7  #Define the layout for console appender
8  log4j.appender.CA.layout=org.apache.log4j.PatternLayout
9  log4j.appender.CA.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

Below is the "log4.properties" file for fileappender. In this case logger messages are written to specified file.

```
1  #Define the root logger with appender file
2  log = D:/log
3  log4j.rootLogger=DEBUG, FILE  reference of file appender
4
5  #Define the file appender
6  log4j.appender.FILE=org.apache.log4j.FileAppender  It prints log stmts of files
7  log4j.appender.FILE.File=${log}/log.out
8
9  #Define the layout for file appender
10 log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
11 log4j.appender.FILE.layout.conversionPattern=%m%n
```

Note: Place the above "log4.properties" file in class path.

DOMConfigurator

You need to use the DOMConfigurator.configure() method to configure log4j using a XML configuration file.

```
1  import org.apache.log4j.Logger;
2  import org.apache.log4j.xml.DOMConfigurator;
3
4  public class LogExample3 {
5
6    static final Logger logger = Logger.getLogger(LogExample3.class);
7
8    public static void main(String[] args) throws Exception
9    {
10     DOMConfigurator.configure("log4j.xml");
11     logger.trace("This is a Trace Message!");
12     logger.debug("This is a Debug Message!");
13     logger.info("This is a Info Message!");
14     logger.warn("This is a Warn Message!");
15     logger.error("This is a Error Message!");
16     logger.fatal("This is a Fatal Message!");
17   }
18 }
```

Below is the "log4.xml" file for console appender. In this case logger messages are written to console.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
3  <log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
4   <appender name="CA" class="org.apache.log4j.ConsoleAppender">
5     <layout class="org.apache.log4j.PatternLayout">
6      <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n" />
7     </layout>
8   </appender>
9    <root>
10    <level value="DEBUG" />
11    <appender-ref ref="CA" />
12   </root>
13 </log4j:configuration>
```

Below is the "log4.xml" file for file appender. In this case logger messages are written to specified file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
 <appender name="FA" class="org.apache.log4j.FileAppender">
   <param name="File" value="sample.log"/>
   <param name="Threshold" value="WARN"/>
   <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%-4r [%t] %-5p %c %x - %m%n" />
   </layout>
 </appender>
 <root>
   <level value="DEBUG" />
   <appender-ref ref="FA" />
 </root>
</log4j:configuration>
```

Note: *Place the above "log4.xml" file in class path.*

Lets configure log4j for MyNaukriWeb application:

Solution [log4j.properties]

```
#Define the root logger with appender file
log = D:/log
log4j.rootLogger=DEBUG, FILE

#Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

#Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%-4r [%t] %-5p %c %x - %m%n"
```

Solution [SQL Query]

```
CREATE TABLE `userdetails` (
                `email` varchar(50) NOT NULL,
                `password` varchar(50) default NULL,
                `mobile` int(10) default NULL,
                `location` varchar(50) default NULL,
                PRIMARY KEY  (`email`)
              )
```

Solution [UserDetails.java]

```
1  package com.pack.beans;
2  public class UserDetails
3  {
4          private String email;
5          private String password;
6          private int mobile;
7          private String location;
8
9          public String getEmail() {
10                 return email;
11         }
12         public void setEmail(String email) {
13                 this.email = email;
14         }
15         public String getPassword() {
16                 return password;
17         }
18         public void setPassword(String password) {
19                 this.password = password;
20         }
21         public int getMobile() {
22                 return mobile;
23         }
24         public void setMobile(int mobile) {
25                 this.mobile = mobile;
26         }
27         public String getLocation() {
28                 return location;
29         }
30         public void setLocation(String location) {
31                 this.location = location;
32         }
33 }
```

Solution [RegistrationDAO.java]

```
1  package com.pack.dao;
2
3  import com.pack.beans.UserDetails;
4
5  public interface RegistrationDAO {
6
7   public void addUserDetails(UserDetails user) throws RuntimeException;
8
9  }
```

Solution [RegistrationDAOImpl.java]

```
1  package com.pack.dao;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.PreparedStatement;
6  import org.apache.log4j.Logger;
7  import com.pack.beans.UserDetails;
8
9  public class RegistrationDAOImpl implements RegistrationDAO
10 {
11 static final Logger logger = Logger.getLogger(RegistrationDAOImpl.class);
12 private Connection con = null;
13
14 public RegistrationDAOImpl() throws RuntimeException
15 {
16   logger.debug("enter");
17   try{
18     Class.forName("com.mysql.jdbc.Driver");
19     con = DriverManager.getConnection("jdbc:mysql://localhost:3306/test",
20                                                          "root", "");
21 }
22 catch(Exception e){
23         logger.error("Error is due to : " + e.getMessage());
24         throw new RuntimeException("");
25 }
26 logger.debug("exit");
27 }
28
29 public void addUserDetails(UserDetails user)
30 {
31   logger.debug("enter");
32   String sql = "insert into userdetails values(?,?,?,?)";
33   PreparedStatement p_stmt = null;
34   try{
35      p_stmt = con.prepareStatement(sql);
36      p_stmt.setString(1, user.getEmail());
37      p_stmt.setString(2, user.getPassword());
38      p_stmt.setInt(3, user.getMobile());
39      p_stmt.setString(4, user.getLocation());
40      p_stmt.executeUpdate();
41
42      logger.info("User details inserted into Database successfully
43                                  for : " + user.getEmail());
44   }
45   catch(Exception e){
46        logger.error("Error while inserting User details" +
47                                                 e.getMessage());
48   }
49 logger.debug("exit");
50 }
51 }
```

solution [RegistrationServlet.java]

```
1  package com.pack.controller;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10
11 import org.apache.log4j.Logger;
12
13 import com.pack.beans.UserDetails;
14 import com.pack.dao.RegistrationDAO;
15 import com.pack.dao.RegistrationDAOImpl;
16
17 public class RegistrationServlet extends HttpServlet {
18
19 static final Logger logger = Logger.getLogger(RegistrationServlet.class);
20
21 public void service(HttpServletRequest request, HttpServletResponse response)
22                 throws ServletException, IOException {
23
24         logger.debug("enter");
25         PrintWriter out = response.getWriter();
27         try
28         {
29         UserDetails user = new UserDetails();
30         user.setEmail(request.getParameter("email"));
31         user.setPassword((request.getParameter("password")));
32         user.setMobile(Integer.parseInt((request.getParameter("mobile"))));
33         user.setLocation((request.getParameter("location")));
34
35         RegistrationDAO dao = new RegistrationDAOImpl();
36         dao.addUserDetails(user);
37
38         out.println("Registration successfully..!");
39         }
40         catch(Exception e){
41                 logger.error("Error is due to : " + e.getMessage());
42                 out.println("Registration failed");
43         }
44         logger.debug("exit");
45 }
46 }
```

Solution [registration.jsp]

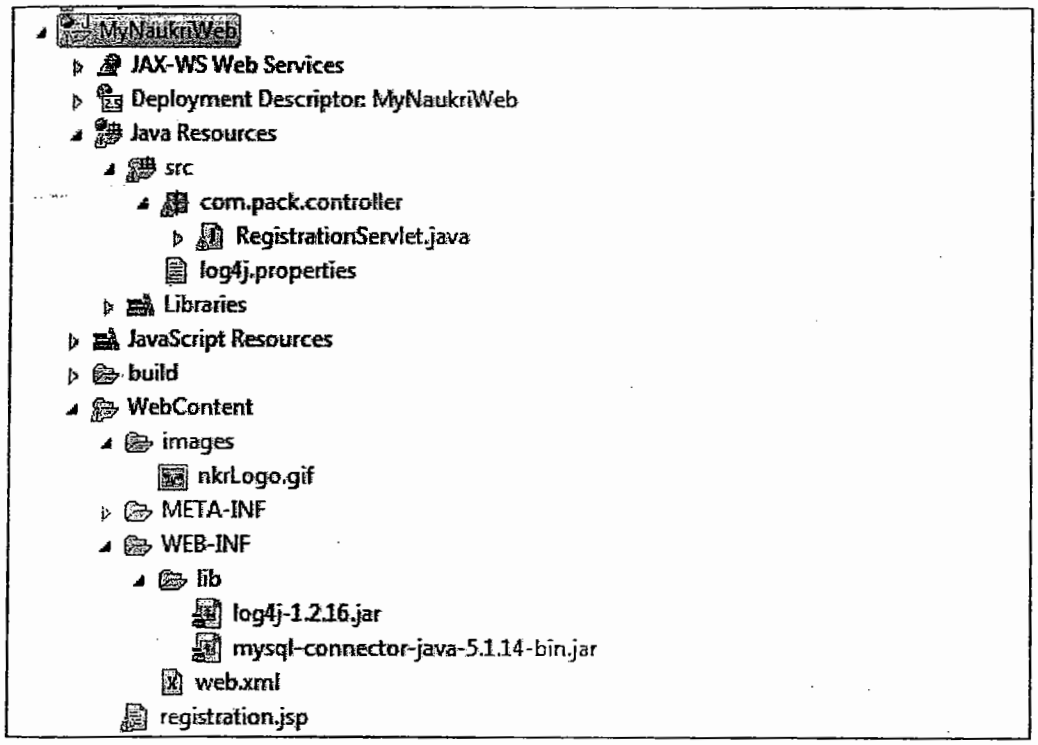
```
<html>
<head>
<title>Job Site</title>
</head>
<body>
<table width="500" border="0" cellspacing="0" cellpadding="0">
        <tr>
                <td> </td>
        </tr>
        <tr bgcolor="#36566E">
                <td height="68" width="48%">
                <div align="left"><img src="images/nkrLogo.gif"
                        width="220" height="74"></div>
                </td>
        </tr>
        <tr>
                <td> </td>
        </tr>
</table>

<form action="/MyNaukriWeb/register" method="post">
        <table width="45%" border="0">
                <tr>
                        <td>Enter your email* :</td>
                        <td><input type="text" name="email"/></td>
                </tr>
                <tr>
                        <td>Enter your password :</td>
                        <td><input type="text" name="password"/></td>
                </tr>
                <tr>
                        <td>Enter your mobile no :</td>
                        <td><input type="text" name="mobile"/></td>
                </tr>
                <tr>
                        <td>Enter your location :</td>
                        <td><input type="text" name="location"/></td>
                </tr>
                <tr>
                        <td colspan="2" align="center"><input type="submit"
                                value="Register"/></td>
                </tr>
        </table>
</form>
</body>
</html>
```

Solution [web.xml]

```
<servlet>
   <servlet-name>RegistrationServlet</servlet-name>
   <servlet-class>com.pack.controller.RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>RegistrationServlet</servlet-name>
   <url-pattern>/register</url-pattern>
</servlet-mapping>

<welcome-file-list>
   <welcome-file>registration.jsp</welcome-file>
</welcome-file-list>
```



```
----------Example2----------
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.Logger;
public class  LogExample2
{
     static final Logger logger=Logger.getLogger(LogExample2.class);
     public static void main(String[] args)
     {
          PropertyConfigurator.configure("log4j-jdbc.properties");
          logger.trace("This is a trace message");
          logger.debug("This is a debug message");
          logger.info("This is a info message");
          logger.warn("This is a warn message");
          logger.error("This is a error message");
          logger.fatal("This is a fatal message");
     }
}
----------log4j.properties----------
#log4j.properties
#Define the root logger with appender  file
log4j.rootLogger=DEBUG, FILE
#Define the File appender
log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
#Define location of log file
log4j.appender.FILE.File=D:/Logs/log.txt
log4j.appender.FILE.DatePattern='.' dd-MM-yyyy
#Define layout for the appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=%-5p  %d{dd/MM/yyyy hh:mm:ss}  %c  %L - %
----------log4j-jdbc.properties----------
# Define the root logger with appender DB
log4j.rootLogger = DEBUG, DB
# Define the DB appender
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender
# Set JDBC URL
log4j.appender.DB.URL=jdbc:odbc:oracledsn
# Set Database Driver
log4j.appender.DB.driver=sun.jdbc.odbc.JdbcOdbcDriver
# Set database user name and password
log4j.appender.DB.user=system
log4j.appender.DB.password=tiger
# Set the SQL statement to be executed.
log4j.appender.DB.sql=INSERT INTO LOGS VALUES('%d{dd-MMM-yy}','%C','%p','%m')
# Define the layout for db appender
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
==========Example3==========
import org.apache.log4j.xml.DOMConfigurator;
import org.apache.log4j.Logger;
public class  LogExample3
{
     static final Logger logger=Logger.getLogger(LogExample3.class);
     public static void main(String[] args)
     {
          DOMConfigurator.configure("log4j.xml");
          logger.trace("This is a trace message");
          logger.debug("This is a debug message");
          logger.info("This is a info message");
          logger.warn("This is a warn message");
          logger.error("This is a error message");
          logger.fatal("This is a fatal message");
     }
}
----------log4j.xml----------
<!DOCTYPE log4j:configuration  SYSTEM  "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
<appender  name="CA"  class="org.apache.log4j.ConsoleAppender">
```

```
 <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%-4r [%t] %-5p %c - %m%n"/>
 </layout>
</appender>
<root>
  <level value="DEBUG"/>
    <appender-ref ref="CA"/>
</root>
</log4j:configuration>
==================Example4==========================
import org.apache.log4j.*;
public class Sample
{
    static Logger logger=Logger.getLogger(Sample.class);
    public static void main(String[] args)
    {
        try{
            throw new Exception();
        }
        catch(Exception e)
        {
            logger.error("Exception occurred in Sample.class");
        }
    }
}
--------------------------------log4j.properties--------------------------------
log4j.rootLogger=ERROR, EMAIL
#define the appender
log4j.appender.EMAIL=com.tgerm.log4j.appender.GmailSMTPAppender
#set SMTP properties
log4j.appender.EMAIL.SMTPHost=smtp.gmail.com
log4j.appender.EMAIL.From=ssreddy2204@gmail.com
log4j.appender.EMAIL.To=srinivasr818@yahoo.com
log4j.appender.EMAIL.SMTPUsername=ssreddy2204@gmail.com
log4j.appender.EMAIL.SMTPPassword=sathyatechnologies
log4j.appender.EMAIL.Subject=Email Notification from Gmail SMTP Appender
log4j.appender.EMAIL.layout=org.apache.log4j.PatternLayout
log4j.appender.EMAIL.layout.ConversionPattern=%-5p %t %c - %m%n
==========================================================================:
```

maven

28-Nov-2015

⟹ In java build process of a project indicates compilation of sourcecode, by adding (having) necessary classpath, and then packaging a project into either jar or a war or ear.

⟹ If any modifications are done in source code, then again we need to recompile the sourcecode, by adding the necessarily jars to classpath and then repackaging an application into a jar or war or ear, is required

⟹ In order to automate the build process, we got ANT as a first build tool from Apache. Later, to overcoming short comings of ANT (Another Neat Tool), we got maven.

⟹ The benefits of using maven build tool are

1). It creates project structure automatically.

2). It downloads and adds the necessary jars from maven repository and it will compile the source code.

3). Stand alone applications. It automatically prepares a test case, with JUNIT/JUnit.

4). Maven shares the downloaded jars for building multiple java projects.

maven – downloads –

⇒ maven is released as a zip. and we can download maven from http://maven.apache.org/download.cgi

⇒ download binary zip archieve (for windows)

apache-maven-3.3.9-bin.zip

for Unix
bin.tar.gz

⇒ Extract zip, then maven is installed.

* Environment settings –

1). set Environment variable JAVA_HOME to jdk folder

Variable name: JAVA_HOME
Variable value: C:\Program files\Java\jdk 1.8.0

2). set Environment variable M2_HOME.

variable name: M2_HOME
Variable value: C:\Apache-maven-3.3.9

3) set Environment variable M2.

Variable name: M2
Variable value: C:\Apache-maven-3.3.9\bin

4) Add maven to the path.

variable name: path
variable value: existing value ; C:\apache-maven-3.3.9\bin ;

⇒ After settings, suppose if we want to test maven is working/set properly or not, then in command prompt type

mvn -v (or) mvn -version

cmd> mvn archetype command

cmd> webapp. ⟶ to filter web app archetypes

cmd> quickstart ⟶ for normal java application

cmd> mvn archetype : generate

Developing web application using maven :⟶

⟹ For developing web applications, we can use either,

1). maven-archetype-webapp (or)

2). webapp-jee5 (or)

3). webapp-javaee6 archetypes

⟹ If we select maven-archetype-webapp, then in pom.xml (project object model.xml) we explicitely need to add the servlet & JSP API dependencies. So it is always better to select webapp-jee5 (or) webapp-javaee6 archetypes.

⟹ To develop a webapplication with maven, follow the below steps.

Step1):- F:\MavenTest> mvn archetype:generate.

Step2): After displaying list of archetypes, choose filter as webapp.

Step 3):→ Identify webapp-jee5 archetype and type its no. (here 34) and later

Choose no: 5

Define value for property 'groupId': com.sathya

'artifactId': MyFirstWeb

'version': enter

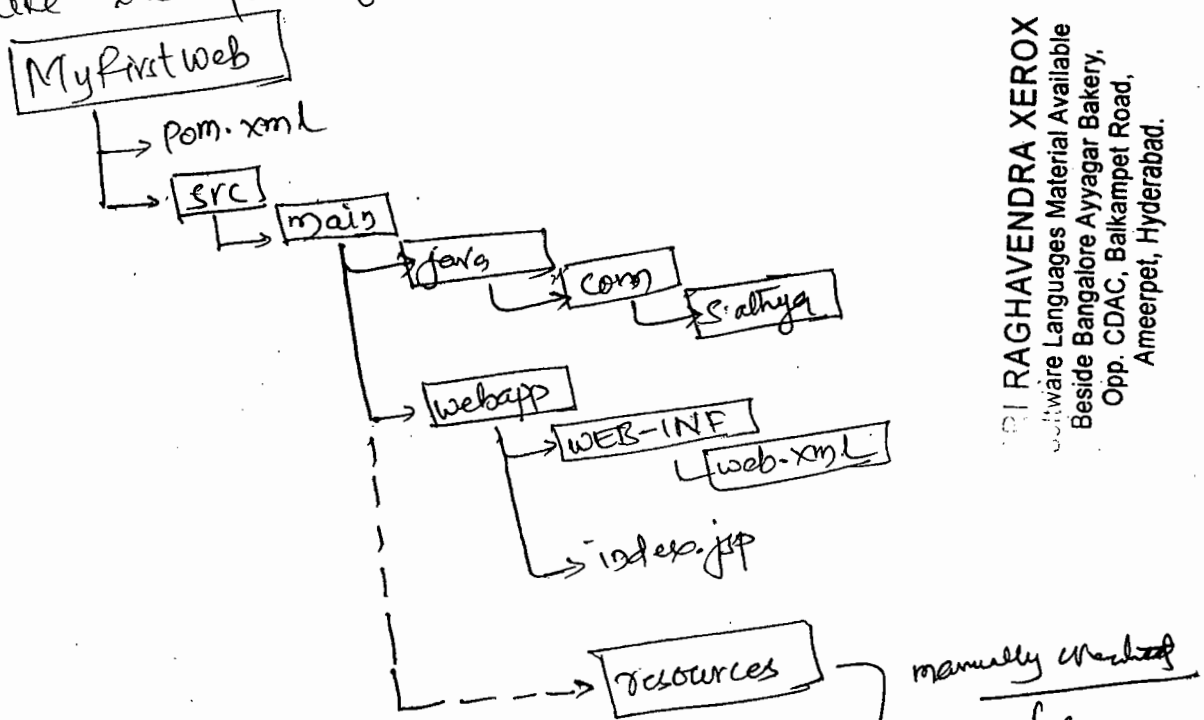
'package': enter

y: : y ↵

f:\MavenTest>

MyFirstWeb

Step 4):→ The Project structure created by maven will look like the following.



This folder we need to create for storing properties files

Step 5:- Create a Servlet and store it under com.sathya package of java folder.

```
package com.sathya;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DummyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)throws
                      ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("A Sample Response from DummyServlet");
} out.close();
}
```

Step 6:- Open index.jsp and add the following <a> tag in <body>

```
<a href="dummy"> click here </a>
```

Step 7):- Open web.xml and configure servlet like following

```
<welcome-file-list>
    <welcome-file> index.jsp </welcome-file>
</welcome-file-list>
<!-- servlet configuration -->
<servlet>
    <servlet-name> sone </servlet-name>
    <servlet-class> com.sathya.DummyServlet </servlet-class>
</servlet>
```

```
<servlet-mapping>
    <servlet-name> sone </servlet-name>
    <url-pattern> /dummy </url-pattern>
</servlet-mapping>

</web-app>
```

⇒ Open pom.xml and modify the version as 1.0.

Step8:- f:\maventest\MyfirstWeb> mvn package.

→ Under the project root folder, a target directory is created, which is parallelly to src folder. In target folder MyfirstWeb-1.0.war is generated.

Step9:- Copy the above .war file to tomcat webapps directory and then start the tomcat server.

Open the Browser and type the request

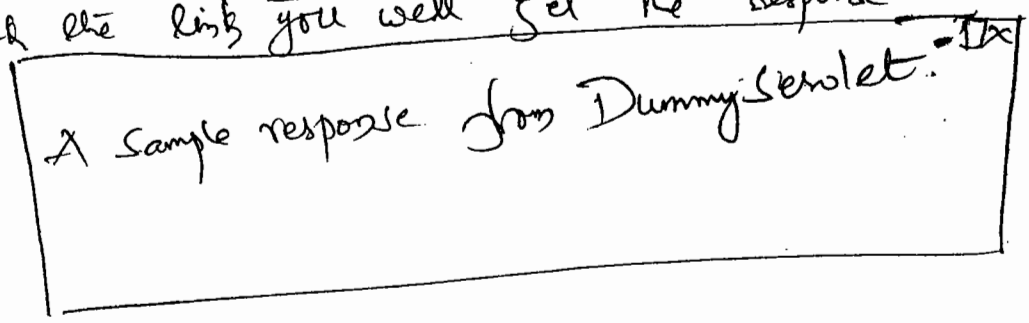
http://localhost:2015/MyfirstWeb-1.0/ ↵

you will get

Hello World

click here ↵

click the link you will get the response.



Creating a web Application Project to Eclipse | Maven + Eclipse

Step1):- Click → Window → Preferences → Expand maven → Archetypes → Add Remote Catalog button → enter Catalog file as | http://repo.maven.apache.org/maven2

Enter Description: maven central → OK

⇒ this step we have done to add more archetypes to the Eclipse from maven Central Repository.

Step2:- file → new → Maven Project → next → from the list Select webapp-jee5 archetype as filter → next → enter groupId as com.sathyatech → artifactId as TestWebApplication → PackageName com.Sathyatech → Finish.

Step3:- Expand src → main → webapp → open index.jsp and design a form like the following.

```
<body>
<form action="login" method="post">
UserName: <input type=text name="uname"/> <br>
password: <input type=password name="pwd"> <br>
<input type="submit" value="login"/>
</form>
</body>
```

Step4:- Expand java Resources → Expand src/main/java
⇒ Rlc com.saltyaTech package → new →
Servlet → Enter class name as loginServlet → next
→ add URL pattern as /login ⇒ ok → next →
select doPost() ⇒ Finish.

```
public void doPost(HttpServletRequest request
        HttpServletResponse response)
{
    // Capture the input
    String sone= request.getParameter("uname");
    String stwo= request.getParameter("pwd");
    //set the content type
    response.setContentType("text/html");
    PrintWriter out= response.getWriter();
    // check login credentials
if("saltya".equals(sone) && "sekhar".equals(stwo))
    { out.println("<h1> login successful </h1>");
    }
    else
    {
        out.println("<a href=" index.jsp> Try again</a>");
    }
    out.close();
}
}
```

Step 5:-

Open pom.xml and change version as 1.1

Rlc on Project ⟶ Run As ⟶ Select Maven Build ... ⟹ enter package in Goals ⟹ Run

Step 6:- Refresh the Project ⟹ Expand target ⟹ copy TestWebApp-1.1.war into Tomcat webapps folder and then start the Tomcat server and type the URL in the Browser

http://localhost:2015/TestWebApp-1.1 ↵

Creating Multimodule Maven Project through Eclipse

⟹ We are creating a maven project with 3 modules.

Presentation – module

Service – module

Integration – module

⟹ In presentation we use servlet, it calls a Service class and Service class calls a DAO class in Integration module.

⟹ A maven project is parent for 3 modules.

Step 1):– file → new → Maven Project → check create a simple Project → next → GroupId : com.peek → artifactId : MyApp → Select packaging : pom → Finish

Step 2):– file → new → Other → Maven → Maven Module → next → create a simple Project ☑ check box → module Name : MyApp-Integration → Browse → select MyApp (Parent Project) → OK → Finish.

Step 3):– file → new → Other → Maven → Maven module → next → Create a simple Project → module Name : MyApp-Service → Browse → MyApp → OK → Finish.

Step4): File → new → other → Maven → Maven Module → next → Module Name: MyApp-Presentation → Browse MyApp → Ok → next → Select filter: webApp-jee5 archetype → next → Change package name as com.pack → Finish.

Step5):- Open pom.xml of MyApp-Presentation module and do the following

a). Remove the GroupId and version tags

b) Add the following <dependency> tag

```
<dependencies>
  <dependency>
    <groupId> com.pack </groupId>
    <artifactId> MyApp-Service </artifactId>
    <version> 0.0.1-SNAPSHOT </version>
  </dependency>
</dependencies>
```

Step6):- Open pom.xml of myApp-service module and do following.

```
<artifactId> myApp-Service </artifactId>
<dependencies>
  <dependency>
    <groupId> com.pack </groupId>
    <artifactId> MyApp-Integration </artifactId>
    <version> 0.0.1-SNAPSHOT </version>
  </dependency>
</dependencies>
```

Step 7):- Expand Integration module → R|c on src/main/java
→ enter package name → com.pack.dao → enter class name
TestDao → finish.

```
public class TestDao {
public String findEmpName (int empno) {
    if (empno == 8866) {
      return "SCOTT";
    }
    else {
    return "Sorry, Info not exist in my DataBase";
} } }
```

Step 8:- Expand myApp-Service module → R|c on src/main/java
→ enter new class → enter package as com.pack.service,
class name as TestService → finish

```
public class TestService {
private TestDao dao = new TestDao();
public String readEmpName (int empno) {
    String str = dao.findEmpName (empno);
    return str;
} }
```

Ctrl+Shift+O → import statements

Step 9):- Expand myApp-Presentation → expand javaResources →
expand → src/main/java → R|c on com.pack → new
Servlet → enter class name as TestServlet → next →

add URL mapping as /TestServlet → OK → next →
Select doGet ⟹ finish.

```
protected void doGet( ) {
    int empno = Integer.parseInt(request.getParameter("empno"));
    TestService service = new TestService();
    // Call service method
    String str = service.readEmpName(empno);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(str);
    out.close();
  }
}
```


Step(10): Open index.jsp and add the following <form> tag

```
<body>
  <form action="testServlet">
  Empno: <input type=text name='empno'>
         <input type=submit value="click">
  </form>
</body>
</html>
```

Step(11): Rlc on myApp (parent Project) → Run As → Select → Maven Build ... ⟹ enter goals ⟹ package ⟹ Run.

Step(12): Refresh → myApp-Presentation ⟹ Copy the war file from target → myApp-Presentation and paste in Tomcat webapps folder

⇒ Start Tomcat Server

Step 13):- Open Browser and type following Request

http://localhost:2015/myApp-Presentation-0.0.1-SNAPSHOT ↵

it display index.jsp

enter empno and click the Button

Enter Empno: 8866

SCOTT

SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.

Sathya Technologies, Ameerpet

Maven

By: Sekhar

Maven

Apache Maven is a software project management and comprehension tool. Maven provides developers a complete build lifecycle framework.

As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides the following:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution

Maven can provide benefits for your build process by employing standard conventions and practices to accelerate your development cycle while at the same time helping you achieve a higher rate of success.

Maven Objective

Maven primary goal is to provide developer:

- A comprehensive model for projects which is reusable, maintainable, and easier to comprehend.
- Plugins or tools that interact with this declarative model.

Setup Maven

1. Download Maven 2.2.1 from http://maven.apache.org/download.html
2. Extract the archive (downloaded in above step), to the directory you wish to install Maven 2.2.1. The subdirectory apache-maven-2.2.1 will be created from the archive.
3. Set Maven environment variables (M2_HOME, M2 & MAVEN_OPTS):
   Set the environment variables using system properties.
   - M2_HOME = C:\Program Files\Apache Software Foundation\apache-maven-2.2.1
   - M2 = %M2_HOME%\bin
   - MAVEN_OPTS = -Xms256m -Xmx512m (this is optional)

My First Maven Project

- To create our first Maven project we are going to use Maven's archetype mechanism.
- An archetype is defined as an original pattern or model from which all other things ofthe same kind are made.

- In Maven, an archetype is a template of a project which is combined with some user input to produce a working Maven project that has been tailored to the user's requirements.

In order to create the simplest of Maven projects, execute the following from the command line:

```
mvn archetype:generate \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DgroupId=com.mycompany.app \
   -DartifactId=my-app
```

Once you have executed this command, you will notice a few things have happened. First, you will notice that a directory named **my-app** has been created for the new project, and this directory contains a file named **pom.xml** that should look like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

pom.xml

- The pom.xml contains the Project Object Model (POM) for this project. The POM is the basic unit of work in Maven.
- This is important to remember because Maven is inherently project-centric in that everything revolves around the notion of a project.
- In short, the POM contains every important piece of information about your project and is essentially one-stop-shopping for finding anything related to your project.

This is a very simple POM but still displays the key elements every POM contains, so let's walk through each of them to familiarize with the POM essentials:

- ✓ **project** - This is the top-level element in all Maven pom.xml files.
- ✓ **modelVersion** - This element indicates what version of the object model this POM is using.
- ✓ **groupId**- This element indicates the unique identifier of the organization or group that created the project. The groupId is one of the key identifiers of a project and is typically based on the fully qualified domain name of your organization.
  For example,org.apache.maven.plugins is the designated groupId for all Maven plug-ins.
- ✓ **artifactId**- This element indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file. Secondary artifacts like source bundles also use the artifactId as part of their final name. A typical artifact produced by Maven would have the form <artifactId>-<version>.<extension> (for example, myapp-1.0.jar).
- ✓ **packaging**- This element indicates the package type to be used by this artifact (e.g. JAR, WAR, EAR, etc.). This not only means if the artifact produced is JAR, WAR, or EAR but can also indicate a specific lifecycle to use as part of the build process. The default value for the packaging element is JAR so you do not have to specify this for most projects.
- ✓ **version**- This element indicates the version of the artifact generated by the project. Maven goes a long way to help you with version management and you will often see the SNAPSHOT designator in a version, which indicates that a project is in a state of development.
- ✓ **name**- This element indicates the display name used for the project. This is often used in Maven's generated documentation.
- ✓ **url**- This element indicates where the project's site can be found. This is often used in Maven's generated documentation.
- ✓ **description**- This element provides a basic description of your project. This is often used in Maven's generated documentation.

After the archetype generation of your first project, you will also notice that the following directory structure has been created:

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

As you can see, the project created from the archetype has a POM, a source tree for your application's sources and a source tree for your test sources.

This is the standard layout for Maven projects (the application sources reside in ${basedir}/src/main/java and test sources reside in ${basedir}/src/test/java, where ${basedir} re presents the directory containing pom.xml).

Compiling Application Source Files

Change to the directory where pom.xml is created by archetype:generate and execute the following command to compile your application sources:

```
mvn compile
```

Upon executing this command you should see output like the following:

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [compile]
[INFO] ------------------------------------------------------------------------
[INFO] artifact org.apache.maven.plugins:maven-resources-plugin: \
  checking for updates from central
...
[INFO] artifact org.apache.maven.plugins:maven-compiler-plugin: \
  checking for updates from central
...
[INFO] [resources:resources]
...
[INFO] [compiler:compile]
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 minutes 54 seconds
[INFO] Finished at: Fri Sep 23 15:48:34 GMT-05:00 2005
[INFO] Final Memory: 2M/6M
[INFO] ------------------------------------------------------------------------
```

The first time you execute this (or any other) command, Maven will need to download all the plugins and related dependencies it needs to fulfill the command.

Note: If you execute the command again, Maven will now have what it needs, so it won't need to download anything new and will be able to execute the command much more quickly.

The compiled classes were placed in ${basedir}/target/classes, which is another standard convention employed by Maven.

Compiling Test Sources and Run Unit Tests

Execute the following command:

```
mvn test
```

Upon executing this command you should see output like the following:

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] artifact org.apache.maven.plugins:maven-surefire-plugin: \
   checking for updates from central
...
[INFO] [resources:resources]
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] [compiler:testCompile]
Compiling 1 source file to C:\Test\Maven2\test\my-app\target\test-classes
...
[INFO] [surefire:test]
[INFO] Setting reports dir: C:\Test\Maven2\test\my-app\target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0 sec

Results :
[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 15 seconds
[INFO] Finished at: Thu Oct 06 08:12:17 MDT 2005
[INFO] Final Memory: 2M/8M
[INFO] ------------------------------------------------------------------------
```

Some things to notice about the output are:

- Maven downloads more dependencies this time. These are the dependencies and plugins necessary for executing the tests.
- Before compiling and executing the tests Maven compiles the main code.

If you simply want to compile your test sources (but not execute the tests), you can execute the following:

```
mvn test-compile
```

Creating a Jar

Making a JAR file is straight forward enough and can be accomplished by executing the following command:

```
mvn package
```

If you take a look at the POM for your project you will notice the packaging element is set to jar. This is how Maven knows to produce a JAR file from the above command.You can now take a look in the ${basedir}/target directory and you will see the generated JAR file.

Install Jar in Local Repository

Now you'll want to install the artifact you've generated (the JAR file) in your local repository (~/.m2/repository is the default location). To do so execute the following command:

```
mvn install
```

Upon executing this command you should see the following output:

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Quick Start Archetype
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[INFO] [compiler:compile]
Compiling 1 source file to <dir>/my-app/target/classes
[INFO] [resources:testResources]
[INFO] [compiler:testCompile]
Compiling 1 source file to <dir>/my-app/target/test-classes
[INFO] [surefire:test]
[INFO] Setting reports dir: <dir>/my-app/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
[surefire] Running com.mycompany.app.AppTest
[surefire] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.001 sec

Results :
[surefire] Tests run: 1, Failures: 0, Errors: 0

[INFO] [jar:jar]
[INFO] Building jar: <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing <dir>/my-app/target/my-app-1.0-SNAPSHOT.jar to \
   <local-repository>/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5 seconds
[INFO] Finished at: Tue Oct 04 13:20:32 GMT-05:00 2005
[INFO] Final Memory: 3M/8M
[INFO] ------------------------------------------------------------------------
```

Note that the surefire plugin (which executes the test) looks for tests contained in files with a particular naming convention. By default the tests included are:

- `**/*Test.java`
- `**/Test*.java`
- `**/*TestCase.java`

And the default excludes are:

- `**/Abstract*Test.java`
- `**/Abstract*TestCase.java`

You have walked through the process for setting up, building, testing, packaging, and installing a typical Maven project. This is likely the vast majority of what projects will be doing with Maven.

Other Useful Maven Commands

`mvn clean` –This will remove the target directory with all the build data before starting so that it is fresh.

`mvneclipse:eclipse` –If you are using Eclipse IDE

External Dependencies

The dependencies section of the pom.xml lists all of the external dependencies that our project needs in order to build (whether it needs that dependency at compile time, test time, run time, or whatever). Right now, our project is depending on JUnit only.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- For each external dependency, you'll need to define at least 4 things: groupId, artifactId, version, and scope.
- The groupId, artifactId, and version are the same as those given in the pom.xml for the project that built that dependency.
- The scope element indicates how your project uses that dependency, and can be values like compile, test, and runtime.
- With this information about a dependency, Maven looks in local repository (~/.m2/repository is the default location) to find all dependencies. In a previous section, we installed the artifact from our project (my-app-1.0-SNAPSHOT.jar) into the local repository.

  Note: Once it's installed there, another project can reference that jar as a dependency simply by adding the dependency information to its pom.xml
- Whenever a project references a dependency that isn't available in the local repository, Maven will download the dependency from a remote repository into the local repository.
- You probably noticed Maven downloading a lot of things when you built your very first project. By default, the remote repository Maven uses can be found at http://repo.maven.apache.org/maven2/.

  Note: You can also set up your own remote repository (maybe a central repository for your company) to use instead of or in addition to the default remote repository.

Now that we know the information we need, we can add the dependency to our pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

Now, when we compile the project (mvn compile), we'll see Maven download the log4j dependency for us.

Building Other Types of Projects using Maven

Note that the lifecycle applies to any project type. For example, back in the base directory we can create a simple web application:

```
mvnarchetype:generate \
    -DarchetypeGroupId=org.apache.maven.archetypes \
    -DarchetypeArtifactId=maven-archetype-webapp \
    -DgroupId=com.mycompany.app \
    -DartifactId=my-webapp
```

Note that these must all be on a single line. This will create a directory called my-webapp containing the following project descriptor:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-webapp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>my-webapp</finalName>
  </build>
</project>
```

Note the <packaging> element - this tells Maven to build as a WAR. Change into the webapp project's directory and try:

```
mvn clean package
```

You'll see target/my-webapp.war is built, and that all the normal steps were executed.

Build More Than One Project at Once

The concept of dealing with multiple modules is built in to Maven 2.0. In this section, we will show how to build the WAR above, and include the previous JAR as well in one step.

Firstly, we need to add a parent pom.xml file in the directory above the other two, so it should look like this:

```
+- pom.xml
+- my-app
|  +- pom.xml
|  +- src
|     +- main
|        +- java
+- my-webapp
|  +- pom.xml
|  +- src
|     +- main
|        +- webapp
```

The POM file you'll create should contain the following:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-app</module>
    <module>my-webapp</module>
  </modules>
</project>
```

We'll need a dependency on the JAR from the webapp, so add this to my-webapp/pom.xml:

```
  ...
  <dependencies>
    <dependency>
      <groupId>com.mycompany.app</groupId>
      <artifactId>my-app</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    ...
  </dependencies>
```

Finally, add the following <parent> element to both of the other pom.xml files in the subdirectories:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  ...
```

Now, try below command from the top level directory, run:

```
mvn clean install
```

The WAR has now been created in my-webapp/target/my-webapp.war, and the JAR is included:

```
$ jar tvf my-webapp/target/my-webapp-1.0-SNAPSHOT.war
    0 Fri Jun 24 10:59:56 EST 2005 META-INF/
  222 Fri Jun 24 10:59:54 EST 2005 META-INF/MANIFEST.MF
    0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/
    0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/
    0 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/
 3239 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/pom.xml
    0 Fri Jun 24 10:59:56 EST 2005 WEB-INF/
  215 Fri Jun 24 10:59:56 EST 2005 WEB-INF/web.xml
  123 Fri Jun 24 10:59:56 EST 2005 META-INF/maven/com.mycompany.app/my-webapp/pom.properties
   52 Fri Jun 24 10:59:56 EST 2005 index.jsp
    0 Fri Jun 24 10:59:56 EST 2005 WEB-INF/lib/
 2713 Fri Jun 24 10:59:56 EST 2005 WEB-INF/lib/my-app-1.0-SNAPSHOT.jar
```

- Firstly, the parent POM created (called app), has a packaging of pom and a list of modules defined.
- This tells Maven to run all operations over the set of projects instead of just the current one.
- Next, we tell the WAR that it requires the my-app JAR. This does a few things: it makes it available on the CLASSPATH to any code in the WAR, it makes sure the JAR is always built before the WAR, and it indicates to the WAR plugin to include the JAR in its library directory.
- You may have noticed that junit-3.8.1.jar was a dependency, but didn't end up in the WAR. The reason for this is the <scope>test</scope> element - it is only required for testing, and so is not included in the web application as the compile time dependency my-app is.
- The final step was to include a parent definition.

Maven Eclipse IDE Integration

Eclipse provides an excellent plugin m2eclipse which seamlessly integrates Maven and Eclipse together.Some of features of m2eclipse are listed below:

- You can run Maven goals from Eclipse.
- You can view the output of Maven commands inside the Eclipse using its own console.
- You can update maven dependencies with IDE.
- You can Launch Maven builds from within Eclipse.
- It does the dependency management for Eclipse build path based on Maven's pom.xml.
- It resolves Maven dependencies from the Eclipse workspace without installing to local Maven repository (requires dependency project be in same workspace).

- It automatic downloads required dependencies and sources from the remote Maven repositories.
- It provides wizards for creating new Maven projects, pom.xml and to enable Maven support on existing projects.
- It provides quick search for dependencies in remote Maven repositories.

Installing m2eclipse Plugin

1. Click "Help" menu and select "Eclipse Marketplace" option.
2. Search for "m2eclipse" and click on install to install m2eclipse plugin.
3. Restart the eclipse.
4. Click "Help" menu and select "About Eclipse" option; you can verify the M2 icon.

Import a Maven Project in Eclipse

1. Open eclipse.
2. Select File > Import option.
3. Select "Existing Maven Project" option in Maven type and click "Next" button.
4. Browse the project in file system and click Finish to import project.
5. You can compile, package, install by right clicking on project and select "Run As" option.

Create a Maven Project in Eclipse

1. Open eclipse.
2. Select File > New > Maven Project option.
3. Select "Create s simple project" option and click Next.
4. Provide groupId, artifactId, version, packaging and name.
   Note: packaging is jar in case of standalone project, war in case of web application.
5. Click Finish.

5-DEC-2015

SVN (SubVersion)

Source code.

→ This tool, used to how to share files, directories, documents team members of the project/across multiple users.

→ It maintains the version history, what changes are done, who done, when and what changes are done at the source code.

→ A Version Control System is a Server(like), which manages files, directories, documents, changes made to the files, versions, and the history of the changes.

→ With Version Control System only, a project source code, will be sharable across Team members in network.

→ With Version Control system only, The modifications of one developer in source code, becomes visible to other developers.

→ SVN is a most popular Version Control tool/System, and it is an open source from Apache Software foundation.

→ Before SVN, there was another Version Control System called CVS (Concurrent Version System). SVN is developed based on CVS only.

→ Some other popular Version Control Systems are

1) Git (open source)
2) Vesta (open source)
3) ClearCase (commercial)
4) Perforce (Commercial)

5) Microsoft Visual Source Safe (VSS)(Commercial)

etc.....

Axile → Scrum, Sprint

Terminology:-

1) **Repository:** → A Repository is a directory created in SVN server by a Team Lead / Project Lead.

⇒ A Repository will manage files, directories, versions, changes and history.

⇒ A Repository contains 3 folders

i) Trunk
ii) Tag
iii) Branch

Trunk: ⇒ A Trunk is a folder where a project files, directories are stored.

⇒ All the changes, versions and their history will be stored in trunk only.

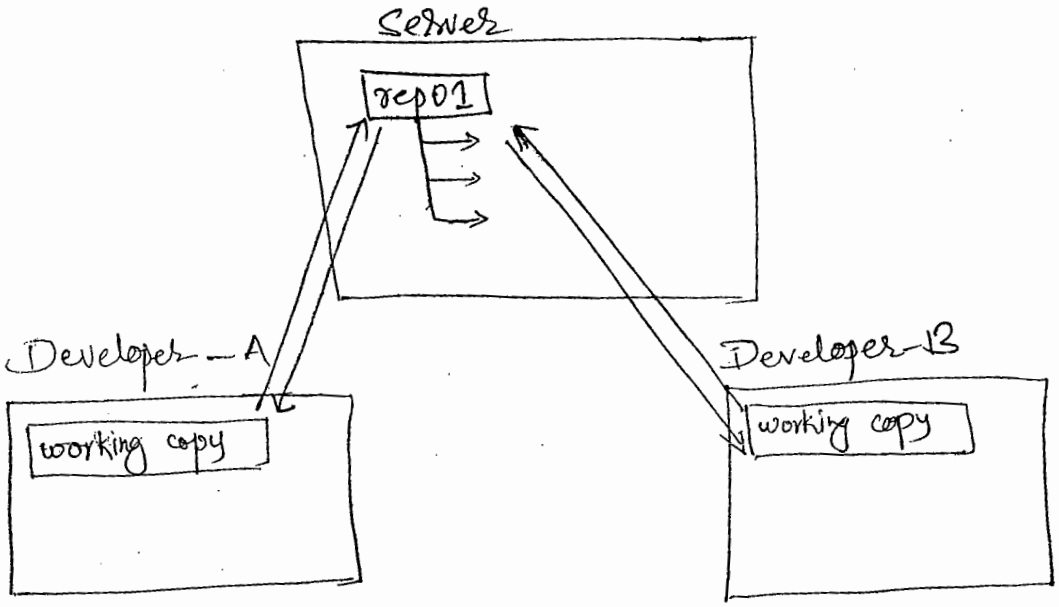
⇒ A trunk is also called main line of Application development.

Tag: ⇒ A tag is created from trunk for a particular release.

Branch: ⇒ If any bug is identified, in a tag, then a Branch is created from a tag, to resolve issue.

⟹ Again a branch also contain 3 subfolders. called i) Trunk
ii) Tag and
iii) Branch

2> Working Copy (Sand Box):—



⟹ A working copy contains files locally on each developer machine. These files are taken from Repository of the server.

⟹ Each developer will have own working copy, to make the changes locally.

⟹ Each working copy again contains 3 folders.
i> Branch
ii) Tag
iii) Trunk

3) Check-in (commit) :⇒

⇒ Writing the changes done by a developer in a local working copy into Repository is called check-in operation.

4) Check-out (update) :⇒

⇒ Reading the changes done by another developer, from Repository into local working copy is called check-out operation.

⇒ If a developer check-in then, the changes are visible to other developers.

⇒ If a developer check-out then, the changes are made by other developers are visible to the current developer.

* Download SVN server from

https://www.visualsvn.com/server/download/

⇒ A developer machine needs SVN client, to connect with SVN server.

⇒ Download and install TortoiseSVN from

https://www.tortoisesvn.net/downloads.html

⇒ After installing successfully, the tortoiseSVN restart the system/pc and then Right click on desktop and in the pop up menu, tortoiseSVN option will be visible.

---

* Creating A Repository in SVN server

⇒ Start ⇒ All programmes ⇒ VisualSVN ⇒ VisualSVN Server Manager

⇒ Right click on Repositories ⇒ Create new Repository

⇒ Regular FSFS Repository ⇒ next enter name as **My Repo** ⇒ next ⇒ select **Single project Repository**

⇒ next ⇒ create.

---

* Creating Users ⇒

⇒ Right Click on Users ⇒ Create User ⇒ Enter username password & confirm password ⇒ OK

Eg:- Username: Dhanush
password: javapsychology
confirm password: javapsychology

⇒ Create one more user in the above manner.
Username: Sathya Seekhar
password: Sathya
Confirm password: Sathya

* Adding users to Repository :→

⇒ Right click on myRepo ⇒ All Tasks ⇒ manage security ⇒ Remove all ⇒ add ⇒ select users and user2 ⇒ OK ⇒ OK.

* Creating a working copy :⇒ 6-Dec-2015

⇒ Create a folder with a name (check1) in any location of your system (eg: E)

⇒ Open check1 folder ⇒ Right click and select SVN checkout ⇒ enter URL of Repository eg

https://DHANUSH-PC/SVN/myRepo/

⇒ OK

⇒ Now we can observe 3 folders
  i) branches,
  ii) Tags
  iii) Trunk in the check1 folder

⇒ Similarly create another folder check2 for another user like this create the folders based on the no. of users i.e. team members.

Adding a new file to the trunk :-

⇒ A user/developer can add a new file to Repository.

⇒ In case of new file, first a user has to add the file and then commit the file (Check in).

⇒ for eg: Open trunk folder check1, create a text file a.txt and add a line of code such as

This is first line by user1

⇒ Right click on a.txt ⇒ select Tortoise SVN ⇒ add ⇒ OK.

⇒ Right click on a.txt ⇒ select SVN commit ⇒ OK.

⇒ Open trunk folder of check2 ⇒ Right click ⇒ SVN update (checkout). Now the file is copied from Repository to the trunk of check2

⇒ Reflecting the changes :-

⇒ Open trunk of check1 ⇒ open a.txt ⇒ add a line This is line

⇒ Right click on a.txt ⇒ SVN commit ⇒ OK

⇒ Open trunk of check2 Right click on a.txt ⇒

`SVN update` i.e. check out.

Now we can see user1 changes in user2.

* Conflict :-

⇒ If a user is directly modifying a file, without doing checkout operation is caused to conflict, that is by committing the file.

⇒ If conflict occurs 3 versions of a file are displayed.

i) mine
ii) R3
iii) R4

such as

⇒ When conflict occured then SVN tool doesn't automatically resolve the conflicts, a developer has to communicate with other developer to resolve the conflict.

⇒ Do the necessary changes in `a.txt`, remove mine, R3, R4 files, Then commit a.txt.

⇒ To overcome the conflicts problem we need to follow `Check out — modify — checkin` model always is useful.

Locking :—

→ A user can lock a file for modifications and till that lock is released, another user is not allowed to make any changes.

→ Go to [trunk] of [check1] ⇒ Right click on [a.txt] ⇒ [tortoise svn] ⇒ [get lock]

→ Open [trunk] of [check2] ⇒ open [a.txt] add some lines of text ⇒ [svn commit] → Error will be displayed because of it is locked by user1 in check1

→ Go to [check1] ⇒ Right click on [a.txt] in [trunk] ⇒ [tortoise svn] ⇒ release lock.

Now lock is released.

⇒ Now from [check2] ⇒ [a.txt] can be committed

→ In this way we should resolve the problems.

* Creating a Tag :—

⇒ A tag is created for each release of a project.

⇒ Every tag indicates a stable point of the project.

⇒ A tag is mainly used for future reference.

⇒ For a first release, a tag contains origin of the project to a specific version / revision.

⇒ Second tag contains code from next version of tag1 to current version and so on.....

⇒ Open [check1] ⇒ Right click ⇒ select [Tortoise svn]

⇒ select [Branch/tag] enter [/tags/tag1] in toPath text field ⇒ select [Head Revision] ⇒ OK.

⇒ Now [tag1] is created in Repository

⇒ To get [tag1] into working copy: [Right click] inside [check1] ⇒ [svn update]

⇒ Now open [tags] folder in [check1], tag1 is visible.

* Creating a Branch:-

⇒ Branches created in Repository and in a tag

⇒ Branches in Repository are for developing modules independently, mostly we create one branch for one module.

⇒ If any bug identified in a release then in order to fix/resolve the bug, a branch created in tag.

⇒ Branches in Repository, are create in development and branches in tag is created in production.

⇒ To create a branch in Repository Right click in [check1] ⇒ [tortoise svn] ⇒ select [Branch/tag] enter [/branches/branch1] in toPath ⇒ select Head Revision ⇒ OK.

Now Branch created in Repository.

⇒ To get the Branch into the working copy,
Right click in [Check 1] ⇒ [SVN update]

* **Merging:—**

⇒ Open Branches then [Branch 1] in Check1 working copy
⇒ open a.txt ⇒ add some lines of code ⇒ commit

⇒ Inside [check 1], Right click and select Tortoise SVN
⇒ select [merge] ⇒ select merge a range of revision
⇒ Enter URL to merge. as

[http://dhanush-pc/svn/myRepo/branches/branch1]

⇒ select all revisions ⇒ next ⇒ merge.

Working with Eclipse & SVN

⇒ mostly SVN plug-in automatically comes along with Eclipse.

⇒ We can verify whether SVN plug-in exist in Eclipse or not, by going to

Window ⇒ ShowView ⇒ Other ⇒ Verify SVN

⇒ if not exist then we need to install, SVN plug-in to Eclipse.

⇒ Click on Help ⇒ Eclipse MarketPlace ⇒ Select Popular ⇒ install Subversive

Note:- for this SVN plug-in installation Internet connection is mandatory.

⇒ Click on Window ⇒ ShowView ⇒ Other ⇒ Expand SVN ⇒ select SVN Repositories ⇒ Select SVN Kit ⇒ next ⇒ next ⇒ finish

⇒ SVN Repository View ⇒ Right click ⇒ new ⇒ Repository location ⇒ enter Repository URL ⇒ https://dhcp-pc/svn/myRepo/ ⇒ enter username, password ⇒ finish.

⟹ Right click on Repository URL ⟹ checkout As ⟹ Finish ⟹ select Java project ⟹ next ⟹ enter Project name ⟹ finish.

Eg: Projectname is Project1

⟹ Open a.txt add some lines of code.

Right click on a.txt ⟹ Team ⟹ commit.

SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.

SVN

SVN :- Sub Version tool

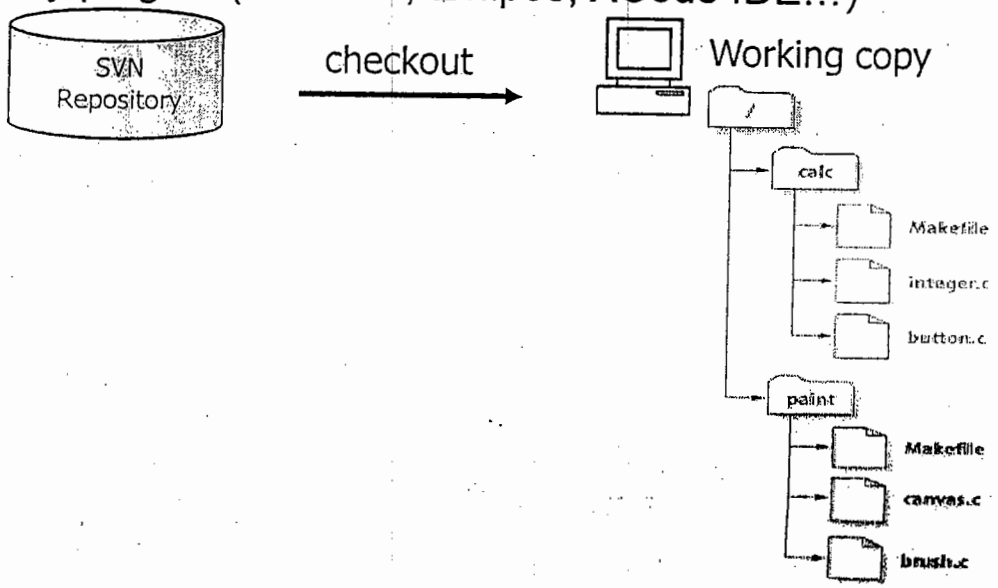
→It is a software versioning and revision control system distributed under an open source license.

→It is the project of Apache Software foundation

→ **2015-09-23 — Apache Subversion 1.9.2 Released**

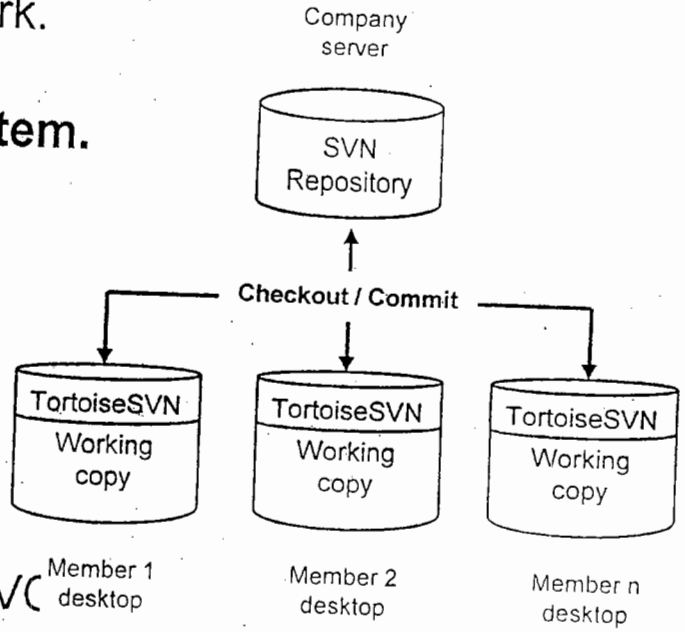
What is SVN?

- Subversion (SVN) is a free open-source version control system.
- The SVN system is needed to maintain the current and historical versions of files such as source code, web pages, and documentations.
- Managed folders/files are placed into a *repository*. A *repository* is much like an ordinary file server.
- Subversion allows you to recover old versions of your data, or examine the history of how your data changed.
  - Free
  - Support Windows, UNIX, Linux
  - Large community: many clients/shells (TortoiseSVN, Syncro, SmartSVN…), many plugins (VS .NET, Eclipse, XCode IDE…)



What is Version Control System?

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.



Following are the goals of a Version Control System.
Allow developers to work simultaneously.
Do not overwrite each other's changes.
Maintain history of every version of everything.

A VCS is divided into two categories.

Centralized Version Control System (CVCS), and
Distributed/Decentralized Version Control System (DVC

In this class, we will concentrate only on the Centralized Version Control System and especially **Subversion**. Subversion falls under centralized version control system, meaning that it uses central server to store all files and enables team collaboration.

Version Control Terminologies

Repository:

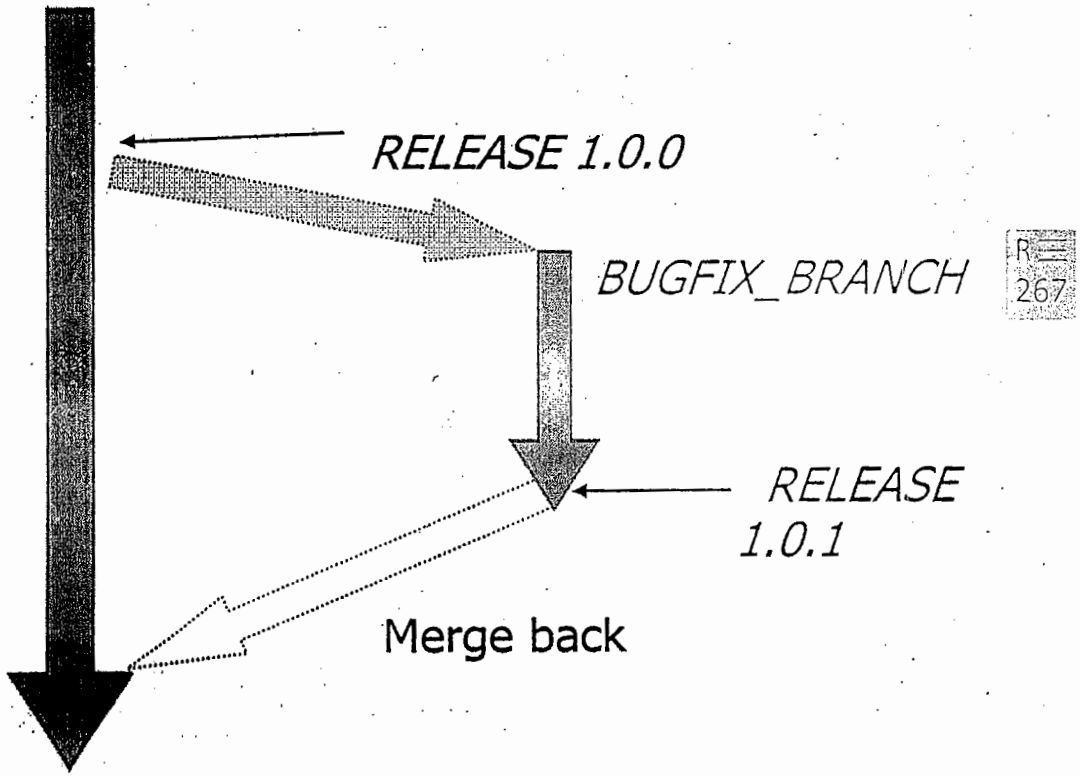
A repository is the heart of any version control system. It is the central place where developers store all their work.
Repository not only stores files but also the history.
Repository is accessed over a network, acting as a server and version control tool acting as a client.
Clients can connect to the repository, and then they can store/retrieve their changes to/from repository. By storing changes, a client makes these changes available to other people and by retrieving changes, a client takes other people's changes as a working copy.

- **trunk**
- A trunk in SVN is main development area, where major development happens.
- -A branch in SVN is sub development area where parallel development on different functionalities happens. After completion of a functionality, a branch is usually merged back into trunk.

  branch: Branch operation is used to create another line of development. It is useful when you want your development process to fork off(separate paths.) into two different directions. For example, when you release version 5.0, you might want to create a branch so that development of 6.0 features can be kept separate from 5.0 bug-fixes.
-

Merging From a Branch

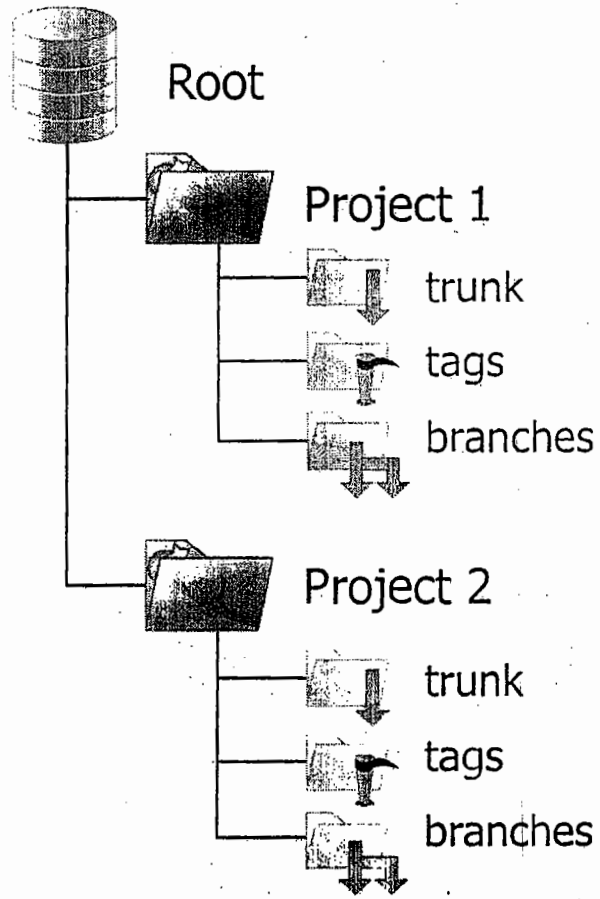
- What's with the bug you've fixed on the bug-fix-branch?
- What about your current development?
- You have to merge the changes made in the branch back to the main line.



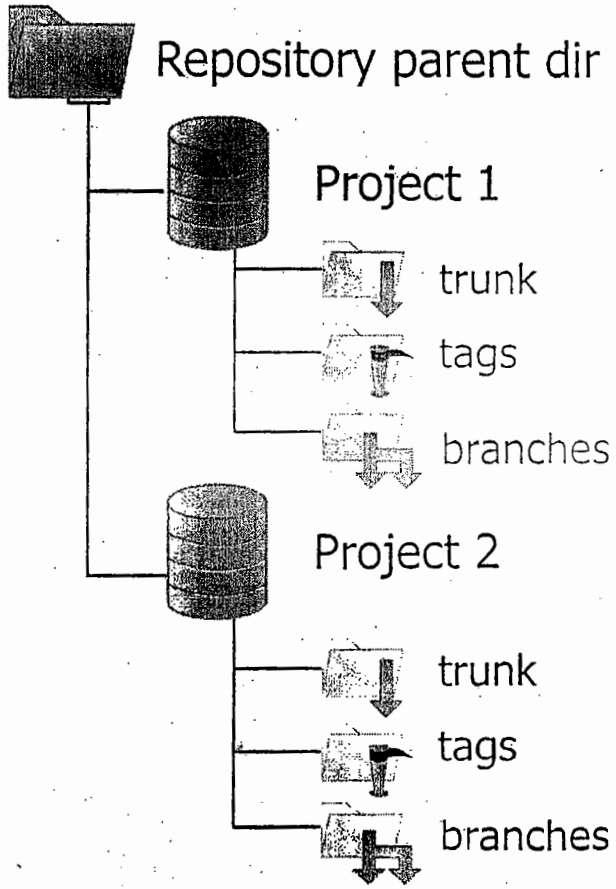
- tag:
- tag is usually used to create read only snapshot of either trunk or branch, which has been released, for future use. You can think tag as stable snapshot of code at any point, and can be used to as backup or restore.

Subversion Repository Layout

One repository, many projects

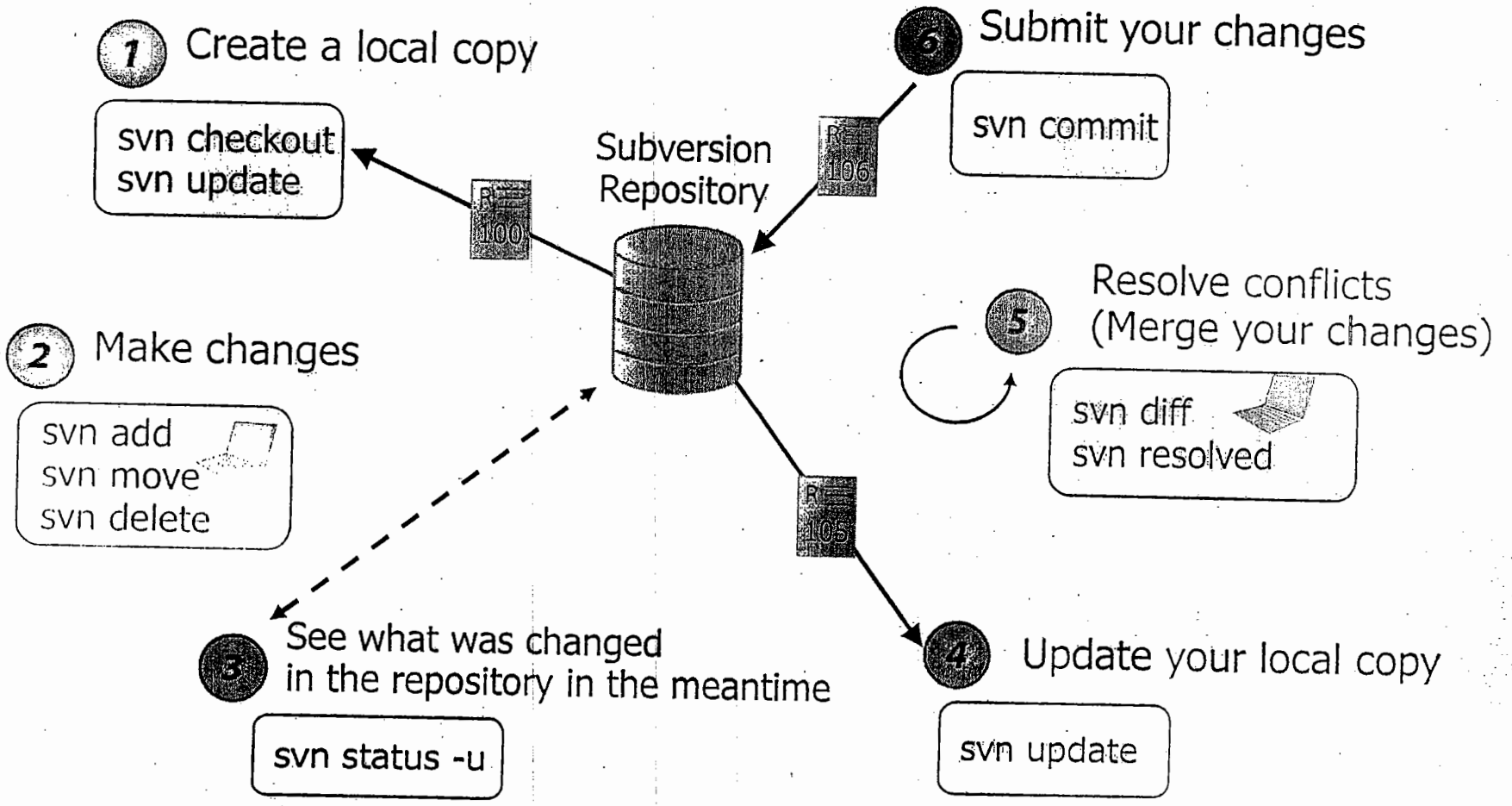


One project per repository



The Work Cycle

Revision number is increased for every transaction that changes the repository.

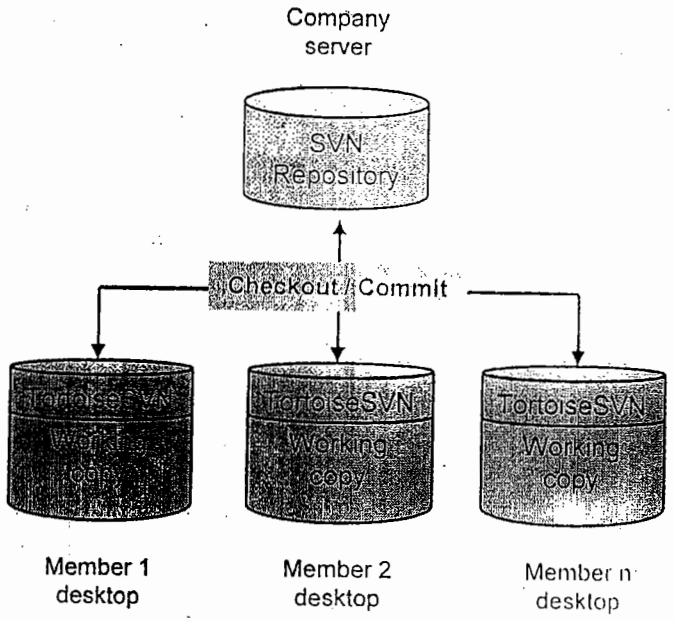


SVN Clients and Plugins

- SVN Client for Windows:
  http://tortoisesvn.tigris.org

- SVN Plugin for Eclipse:
  http://subclipse.tigris.org/

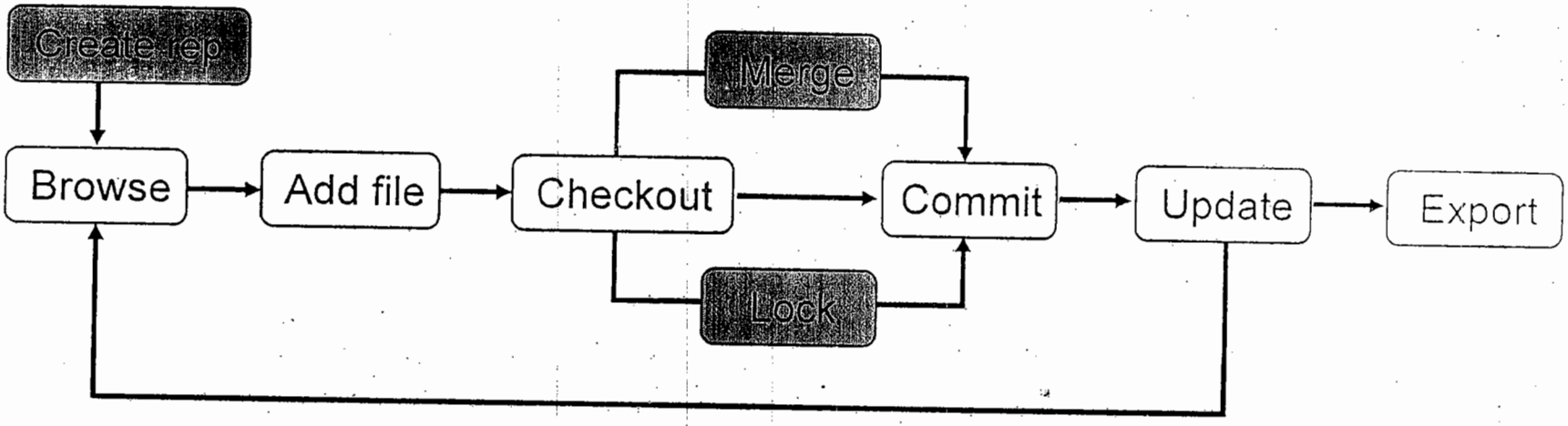
- SVN Plugin for Visual Studio .NET:
  http://ankhsvn.tigris.org/



Common functions in TortoiseSVN

1. Setup TortoiseSVN, grant access right
2. Browse repository
3. Add file/folder to repository
4. Check out a working copy
5. Lock/Release repository files
6. Commit to repository
7. Check modification
8. Update working copy
9. Revision log dialog
10. View differences
11. Clean up
12. Status of version controlled files/folders

Common functions in TortoiseSVN (cont.)



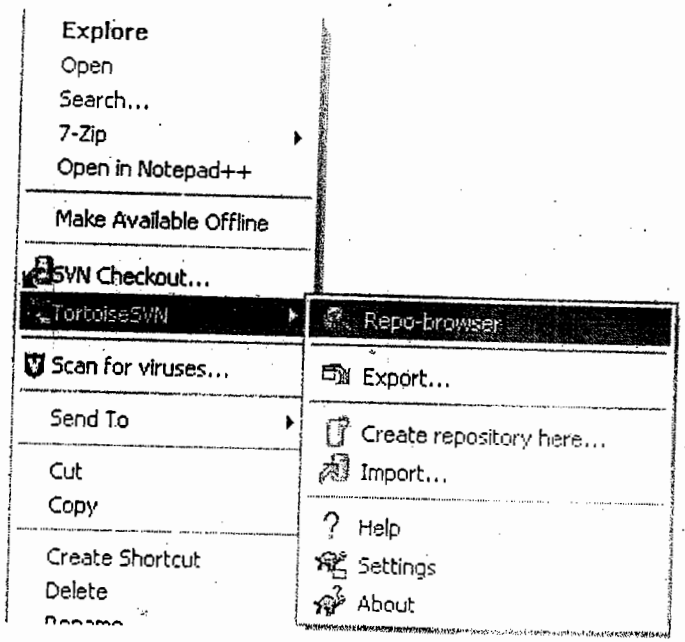
Setup TortoiseSVN, grant access right (1/2)

- Find the latest TortoiseSVN at: http://tortoisesvn.tigris.org/download.html
- Download and install the package: TortoiseSVN-xxx.msi

After installing, TortoiseSVN is integrated into Windows Explorer:

- Open Windows Explorer
- Right click, popup menu opened
- Select TortoiseSVN

Note: for stand-alone PC, TortoiseSVN can be used as a source control system, it's not necessary to setup a SVN server.



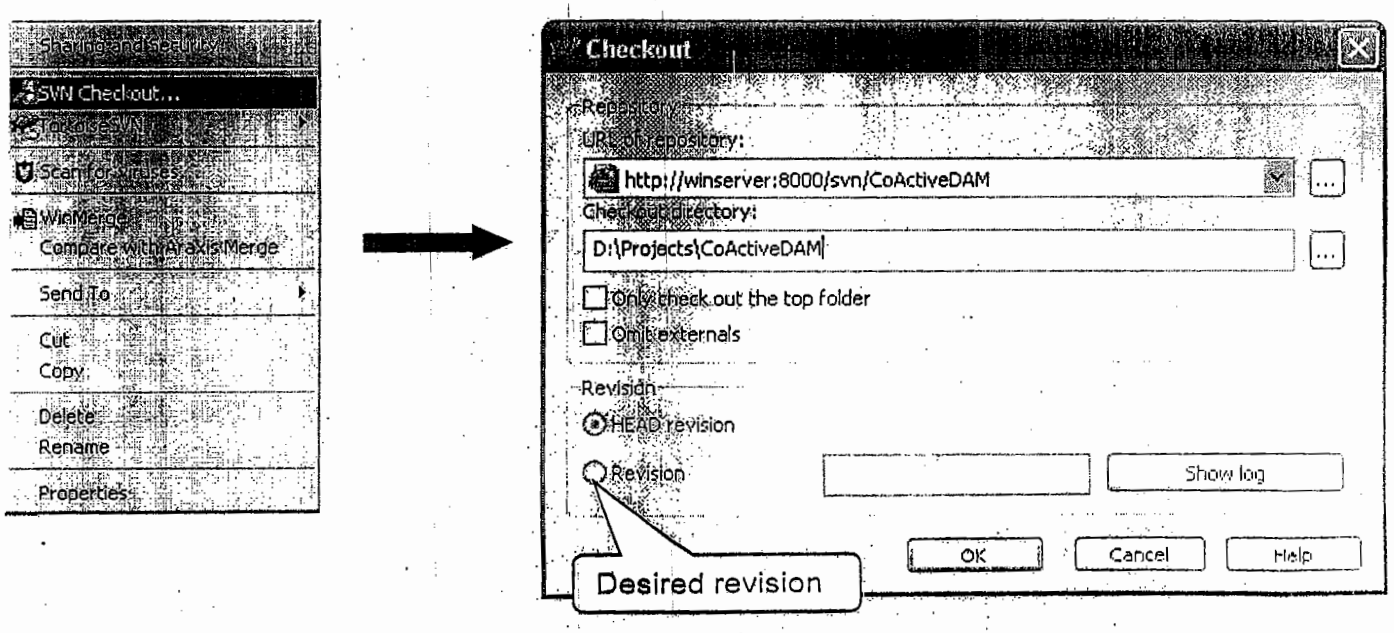
Check out a working copy

SVN: Checkout (head rev.) = VSS: Get latest version

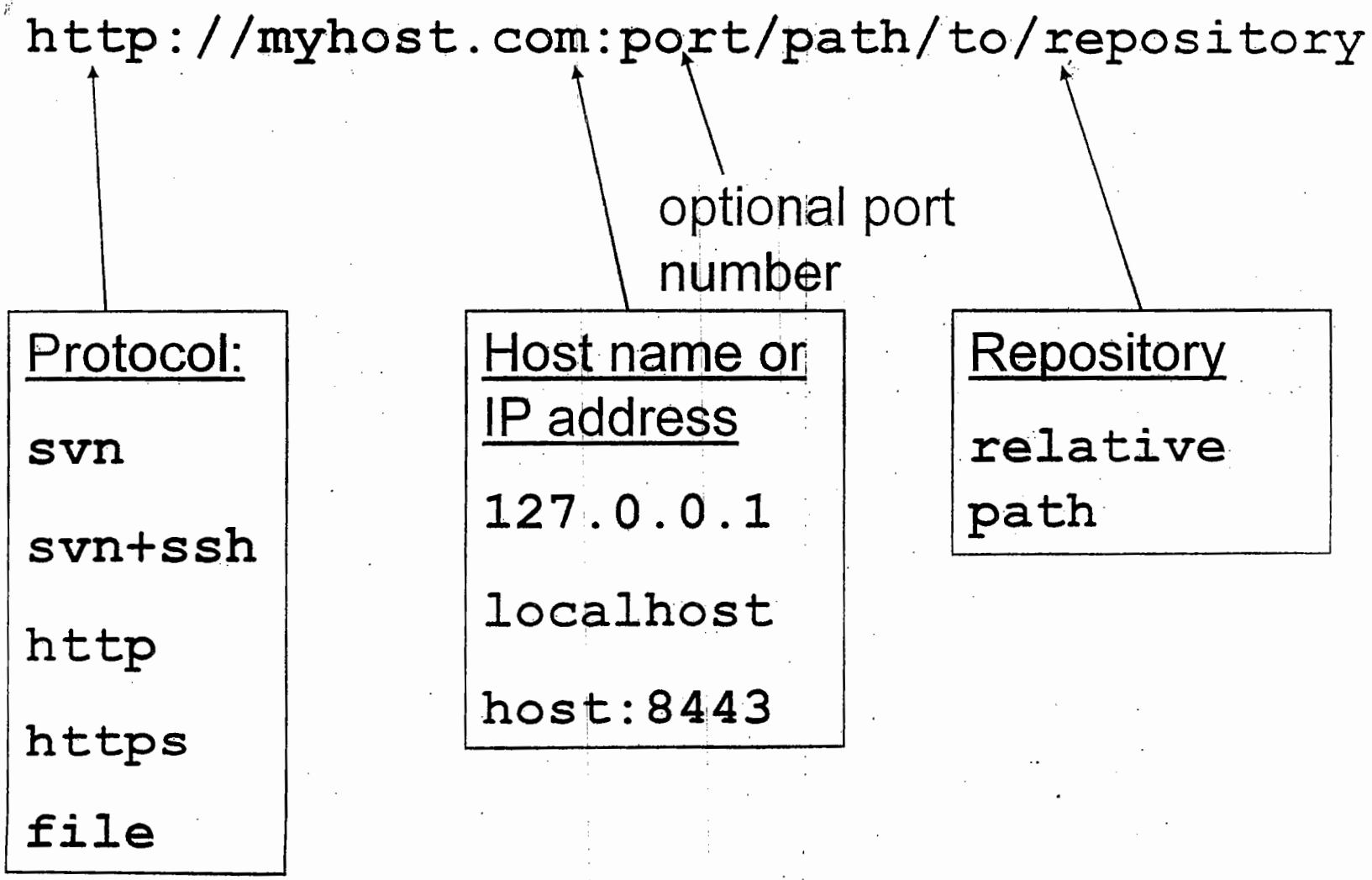
- In Windows Explorer, select working folder, right click>SVN Checkout...

  OR

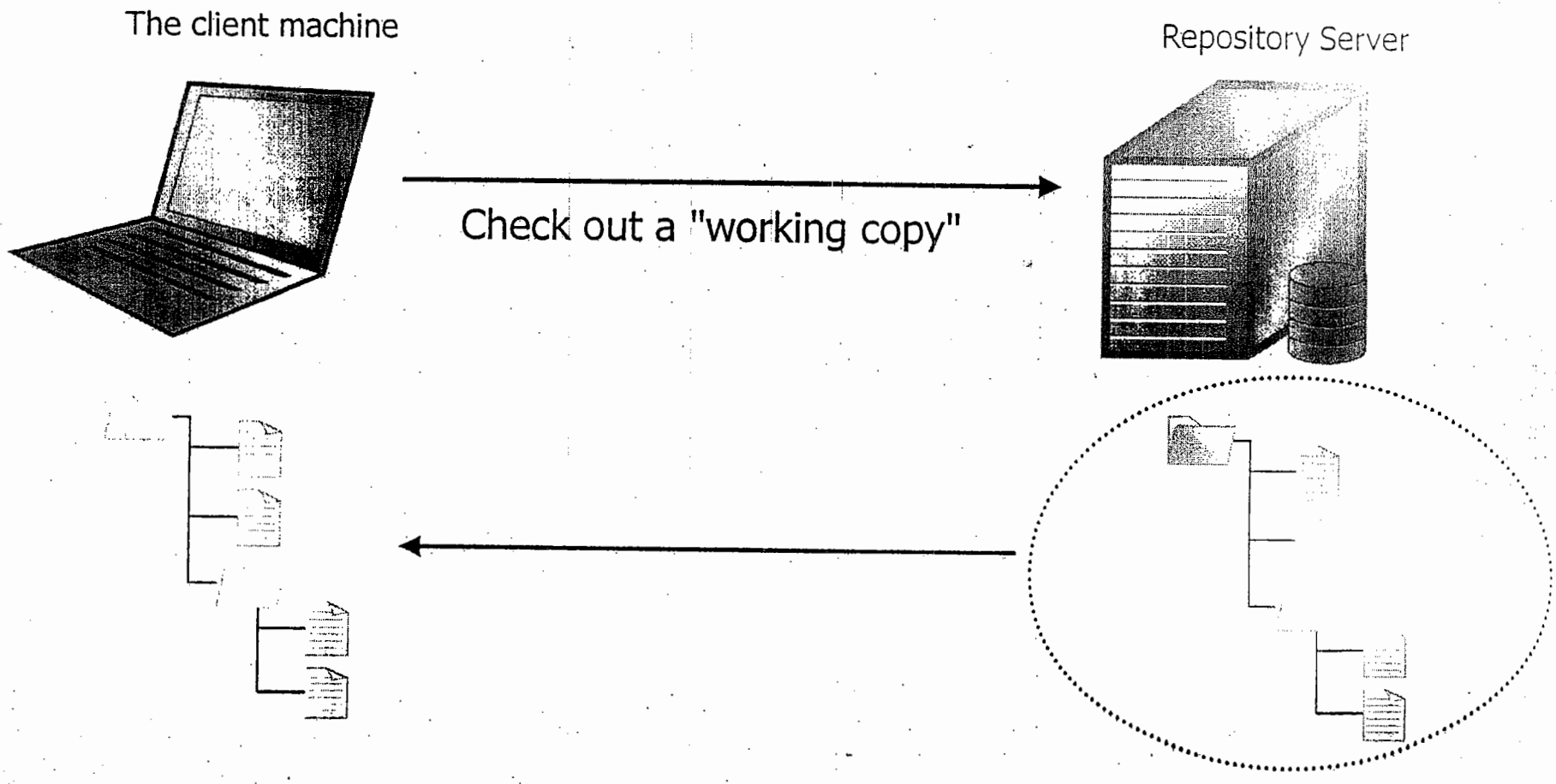
- In Repo-browser, select the folder to check out, right click>Checkout...



URLs and Protocols

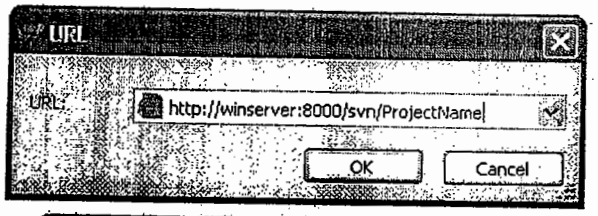


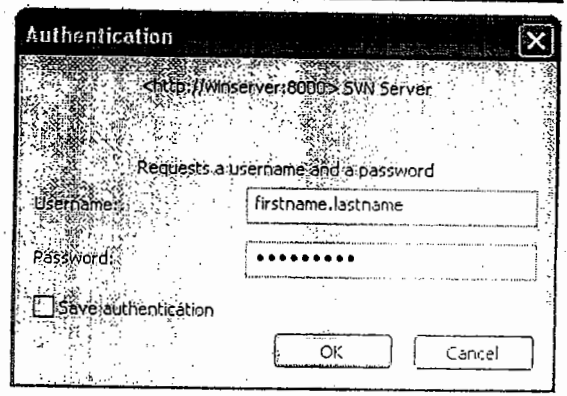
Check Out and the "Working Copy"

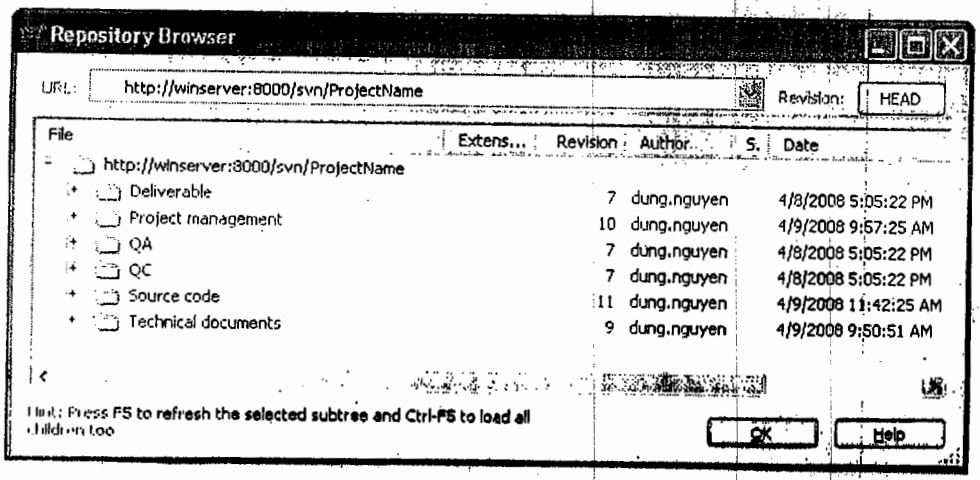


Browse repository

- In Windows Explorer, right click>TortoiseSVN>Repo-browser
- Put the repository URL into the dialog box
- Login with given username and password
- The Repo-browser opened



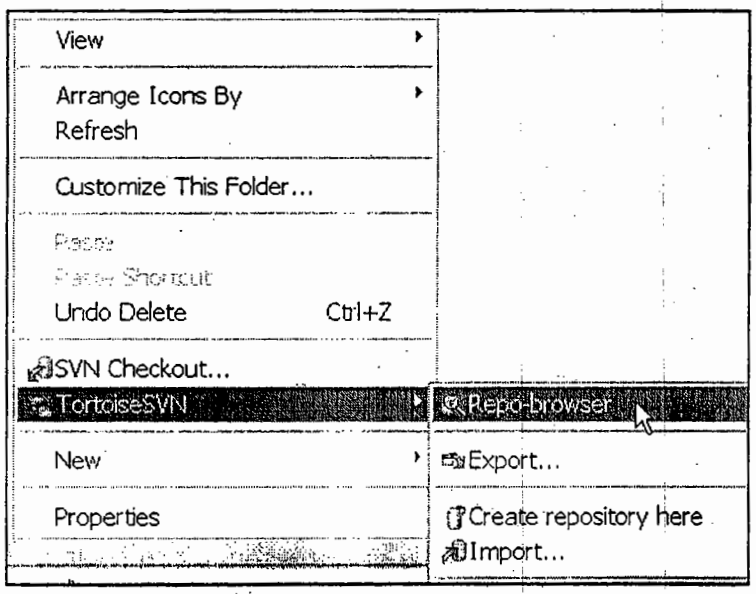




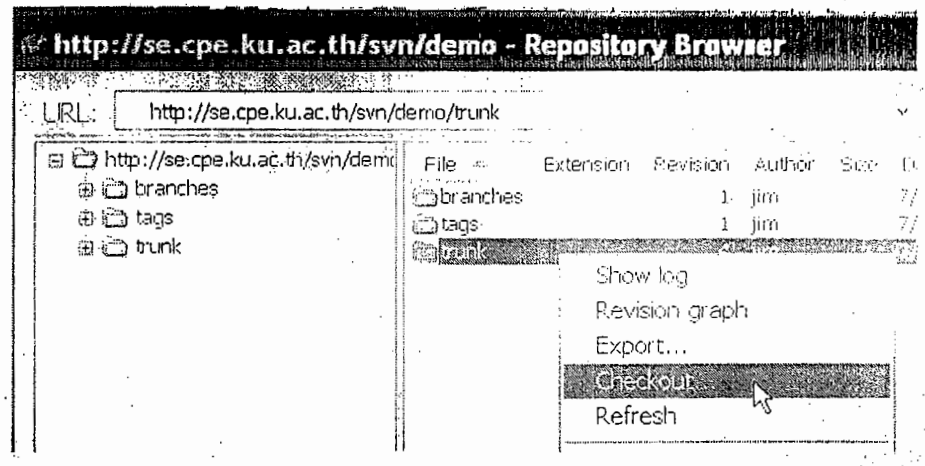
Check Out using TortoiseSVN

Using Windows Explorer, right-click in a directory.

1. If not sure of path to check-out then use Repo-browser first.



2. In Repo-browser, right-click on folder or file you want to check-out.

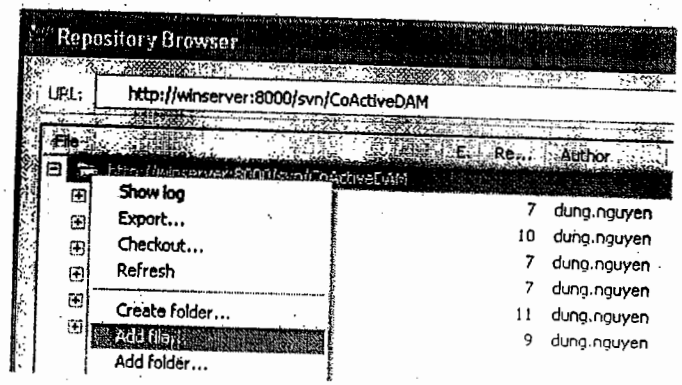


Add file/folder to repository

A. Without (local) working copy folder

- In Repo-browser, right click>Add folder…
- Select source file/folder to be added

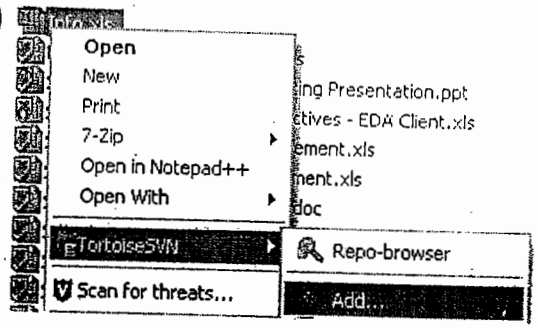
=> *New files/folders will be added under current selected folder.*



B. With (local) working copy folder (recommended)

- Copy new file/folder to the working copy folder
- In Windows Explorer, select new file/folder, right click>TortoiseSVN>Add…
- TortoiseSVN>Commit…

=> *New file/folder has a check mark.*



Check-out: Advice

Don't check-out the entire repository!

Only check out the part that you need.

- For developers, this is usually "/repo/trunk"
- For documenters, maybe "/repo/doc"
- Multi-project repo: //se.../jim/hibernate-demo/trunk/

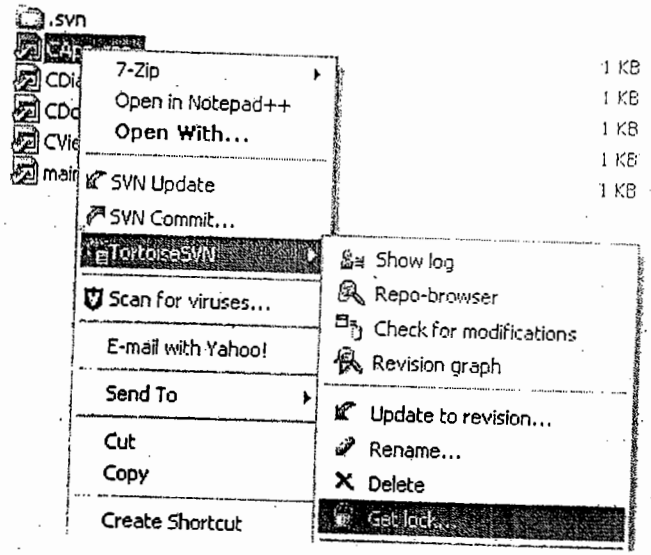
Lock/Release repository files (1/2)

SVN: Checkout + Lock = VSS: Check-out

This is to avoid the conflict when another user updates the repository while you are working on your working copy.

How to:
In the working copy, select file to lock,
right-click>TortoiseSVN>Get lock...

=> Locked file has a lock-icon mark.
Other users will not be able to commit until you release the locked files.



Lock/Release repository files (2/2)

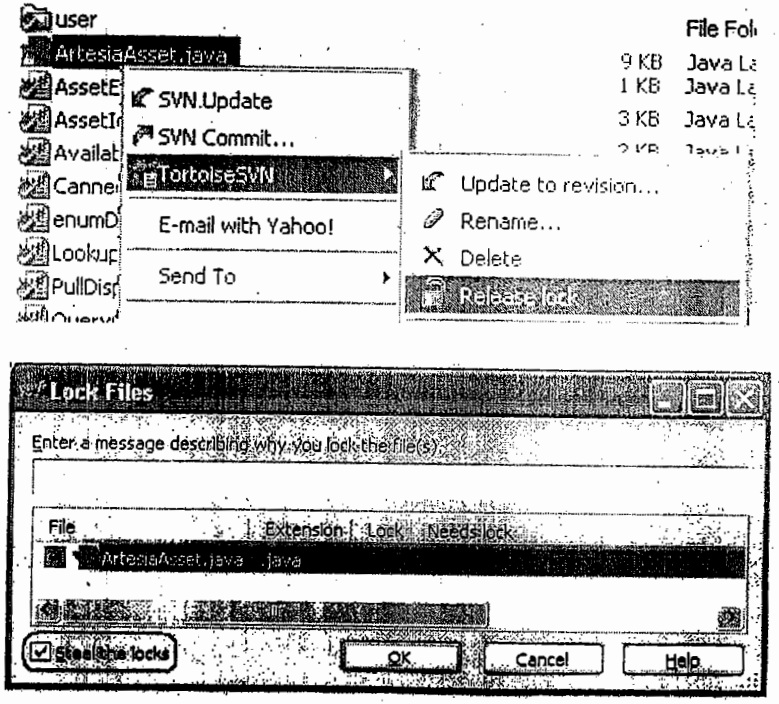
This is to release the locked files for others to commit.

How to:

Select locked file>TortoiseSVN>
Release lock

Steal the locks:
- Select locked file>TortoiseSVN>
  Get lock...
- Check Steal the locks

This way will steal other's lock and replace by your lock. Not recommended. It'd be better to ask the author to release his lock.



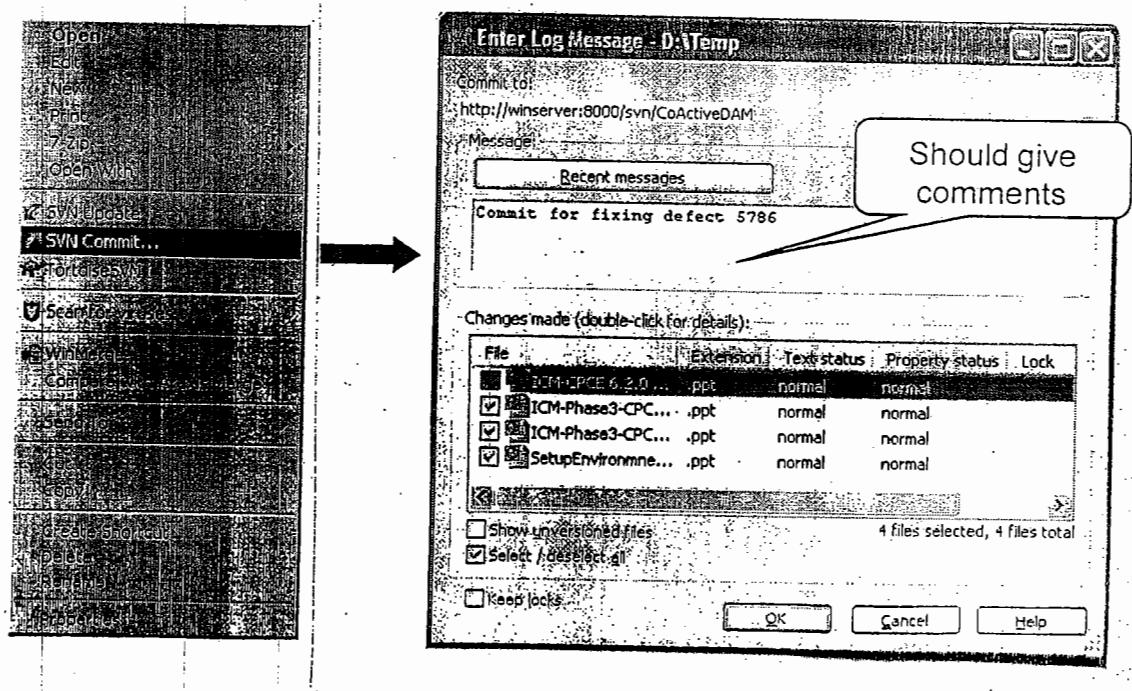
Commit to repository

SVN: Commit = VSS: Check-in

This is to send the changes you made on working copy to the repository.

How to:
On the working copy, right click>SVN Commit…

*Before committing, you should make sure that your working copy is **up-to-date**.*



Single "commit" for related files

- Commit all files related to the same task as one commit.
- This makes comments more meaningful.

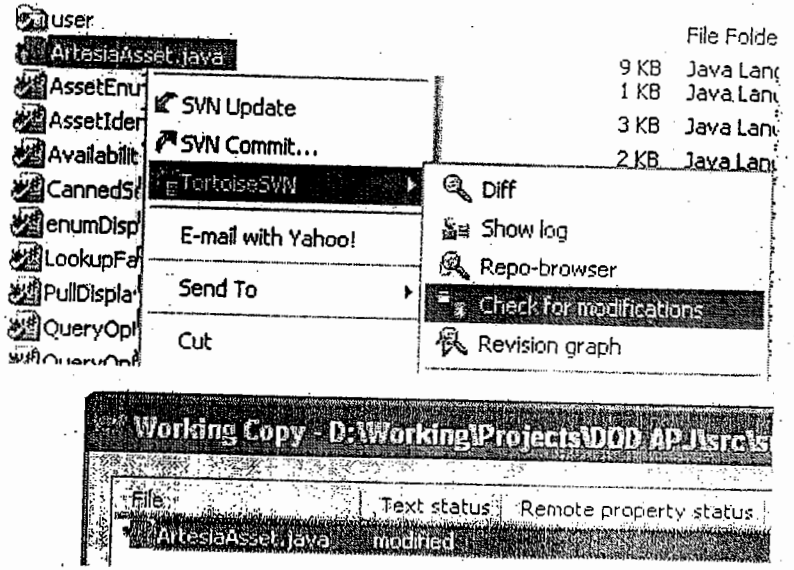
Check modification

This is to know the modification status of selected file in comparison with that file on the repository.

How to:

- On working copy, select modified file
- right click>TortoiseSVN>Check for modifications

The Working Copy dialog shows the status of the selected files in comparison with that file on the repository.



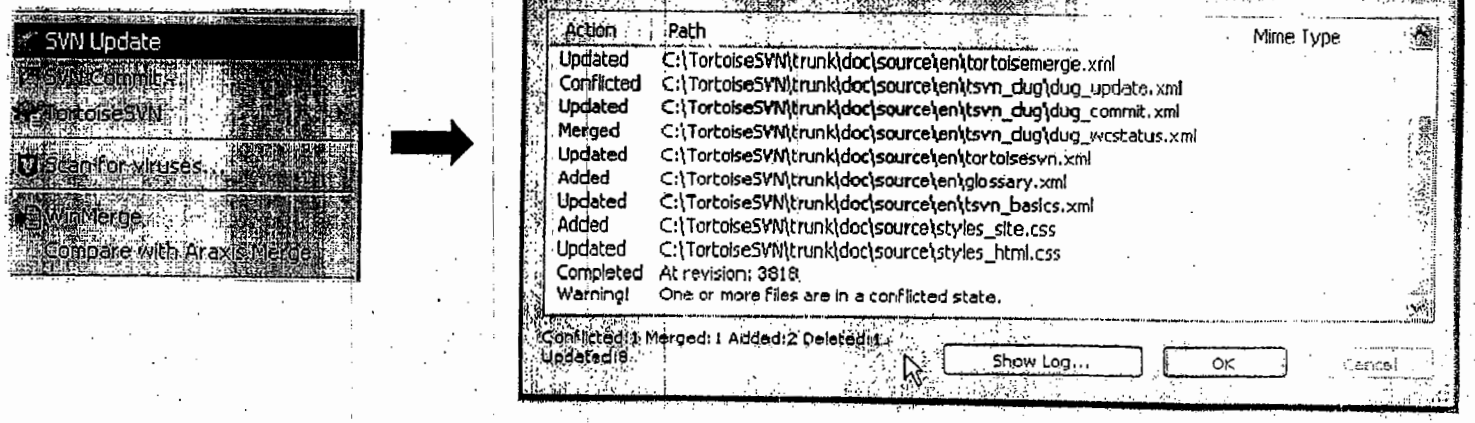
Update working copy (1/2)

SVN: Update = VSS: Get latest version

This is to update your working copy with new changes on the repository.

Periodically, you should ensure that changes done by others get incorporated in your working copy.

How to:
On the working copy, select folder to be updated, right click>SVN Update...

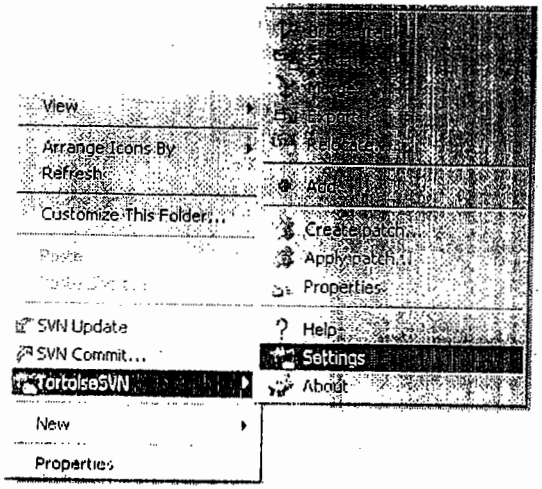


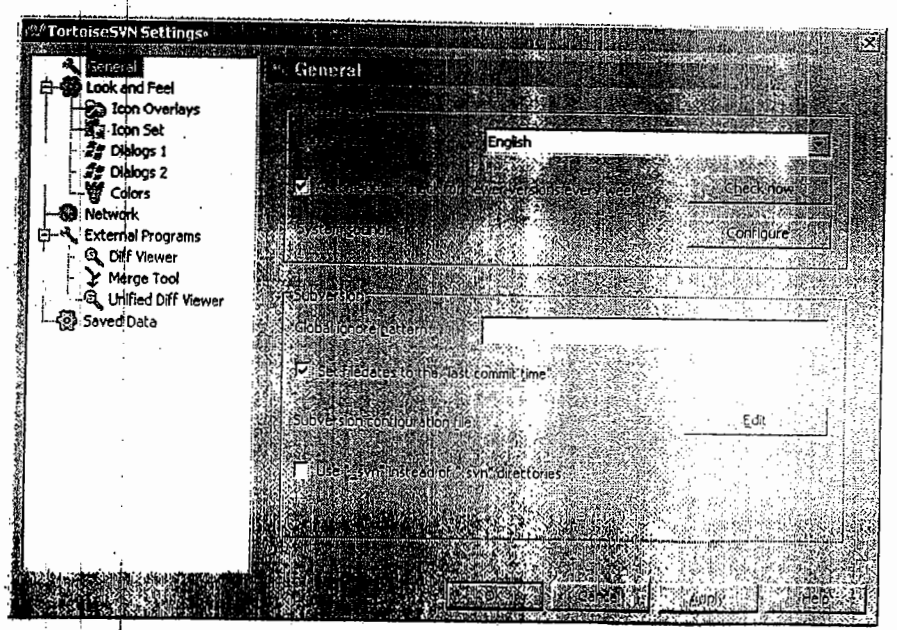
Update working copy (2/2)

To make date time of local files match with date time of files at SVN.

How to:

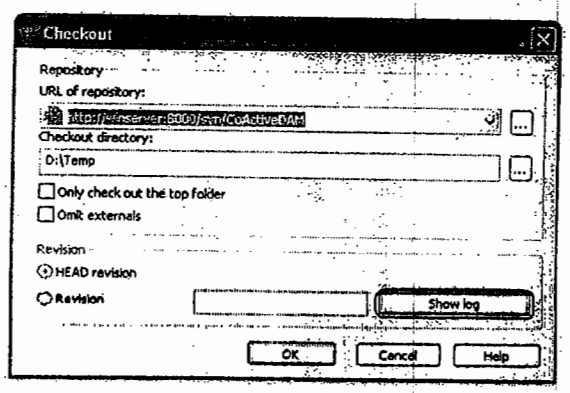
- TortoireSVN>Settings to open TortoiseSVN Settings dialog.
- Turn on option [Set filedates to "last commit time"].

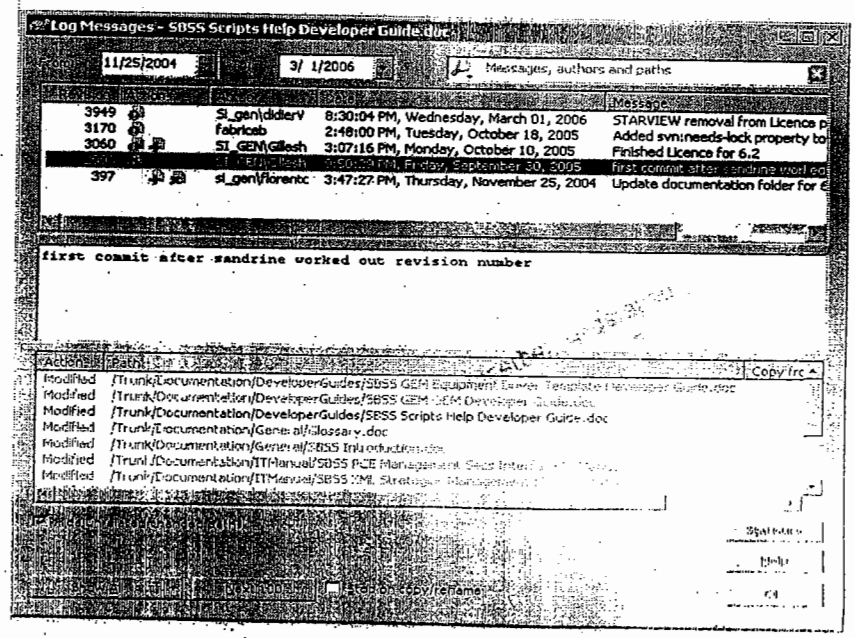




Show Revision log message

- For every change you make and commit, should provide a log message describes the meaning of your change.
- The Log Messages dialog retrieves all those log messages and allows you to select the desired revision.
- Revision number is applied for the whole repository.



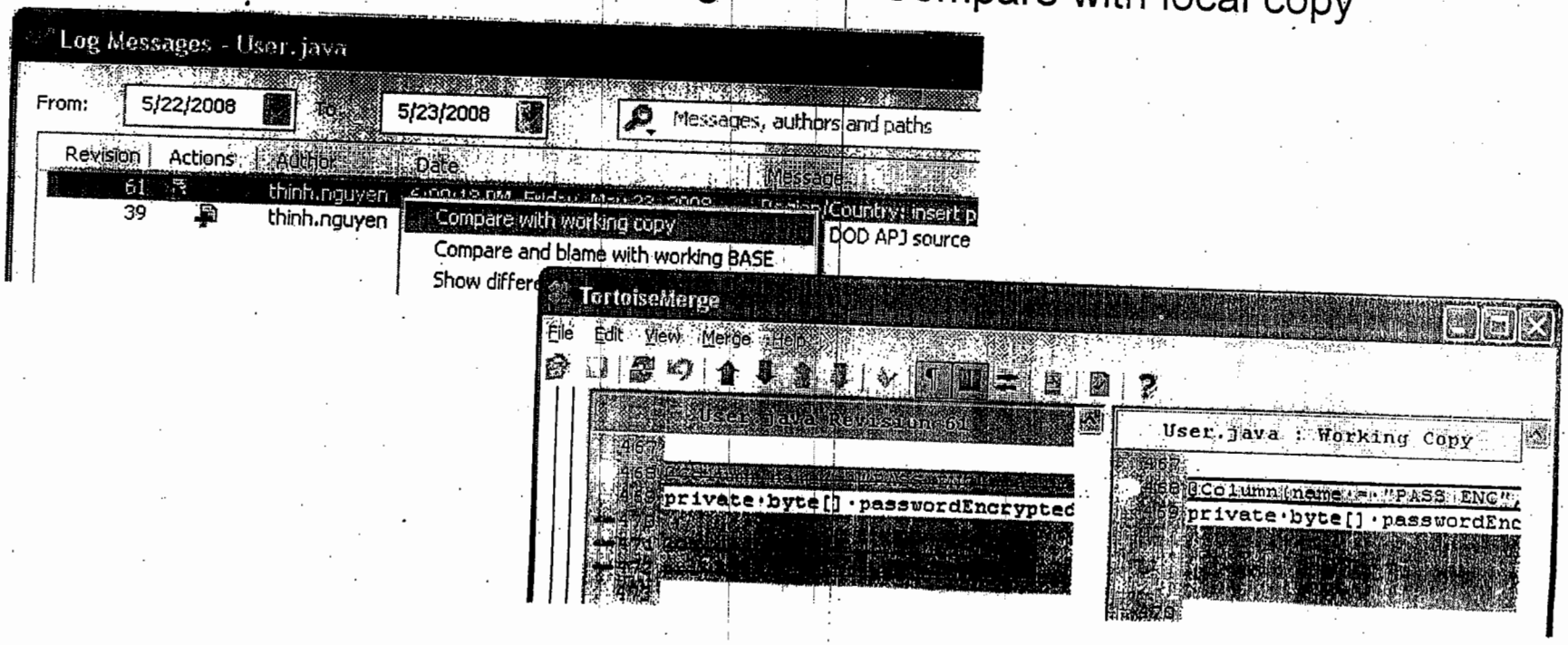


View differences

Compare working copy with a given SVN revision.

How to:

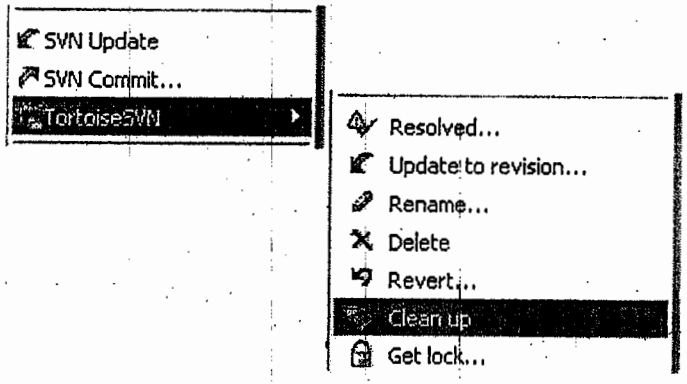
- On working copy, select file to compare
- Right click>Show log => Log messages dialog opened
- Select the revison to compare, right click>Compare with local copy



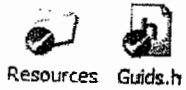
Clean up

If a Subversion command cannot complete successfully, perhaps due to server or network problems, your working copy can be left in an inconsistent state.

It is a good idea to do “clean up” at the top level of the working copy.

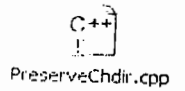


Status of version controlled files/folders



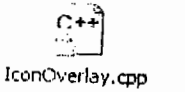
Green checkmark: normal status, under control

Red exclamation: file has been modified since last update and needs to be committed.



Yellow exclamation: a conflict occurs during an update.

Grey checkmark: this file needs to be locked first before editing.



Locked lock: file is locked. Need to unlock for other to commit.

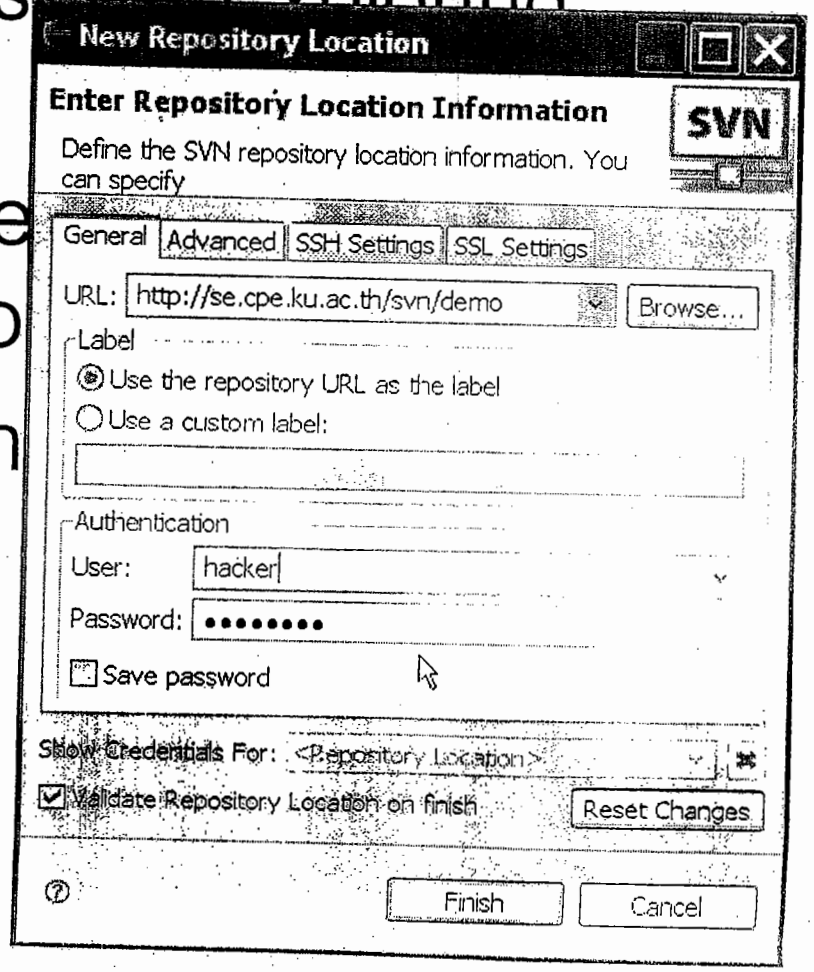
Red deletion: missing file or file to be deleted under version control.

Blue plus: File to be added to version control.

(1) Check out using Eclipse

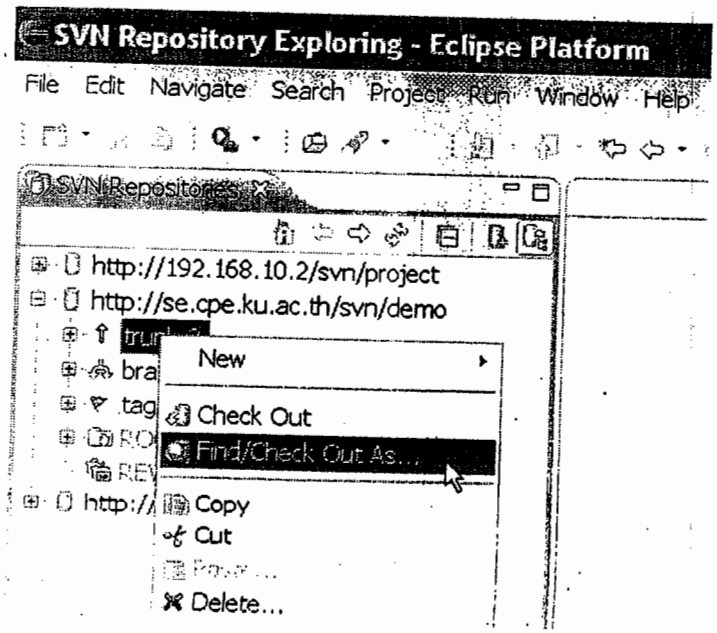
Many ways to do it. Here is a simple way:

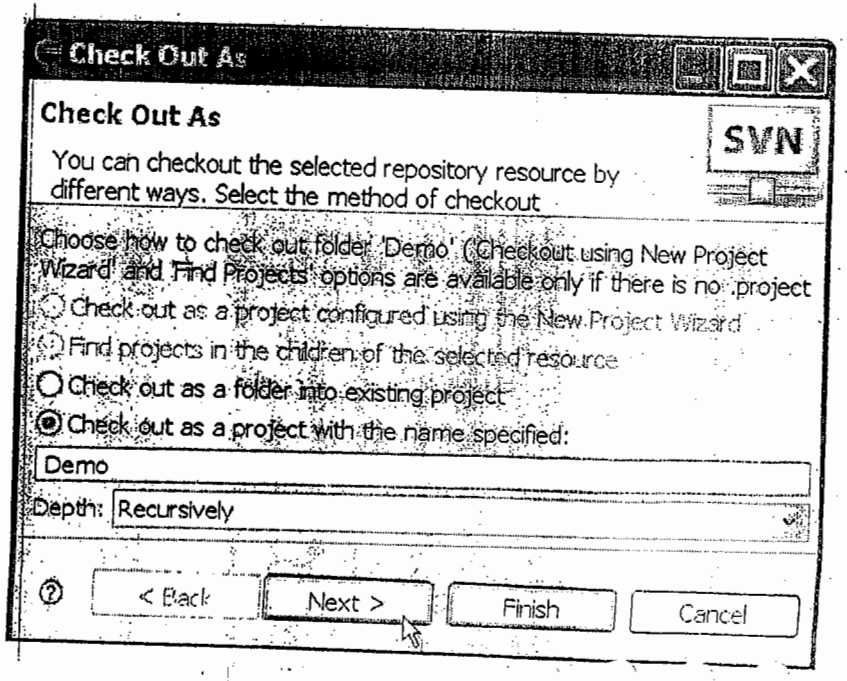
1. Switch to "SVN Repository Exploring Mode".
2. Right click and choose New => Repository Lo
3. Enter URL and (option authentication info.



Check out using Eclipse

- Now you can browse the repository.
- Choose the part you want to check-out (usually "trunk")
- Right click and choose "Check Out as..."
  ("Check Out as..." gives you a chance to change local project name if you want.)





Resolve Conflicts

- "Conflict" means you have made changes to a file, and the version in the repository has been changed, too.
- So there is a "conflict" between your work and work already in the repository.

Resolving Conflicts

The choices are:

(1) merge local & remote changes into one file.

(2) accept remote, discard your local changes.

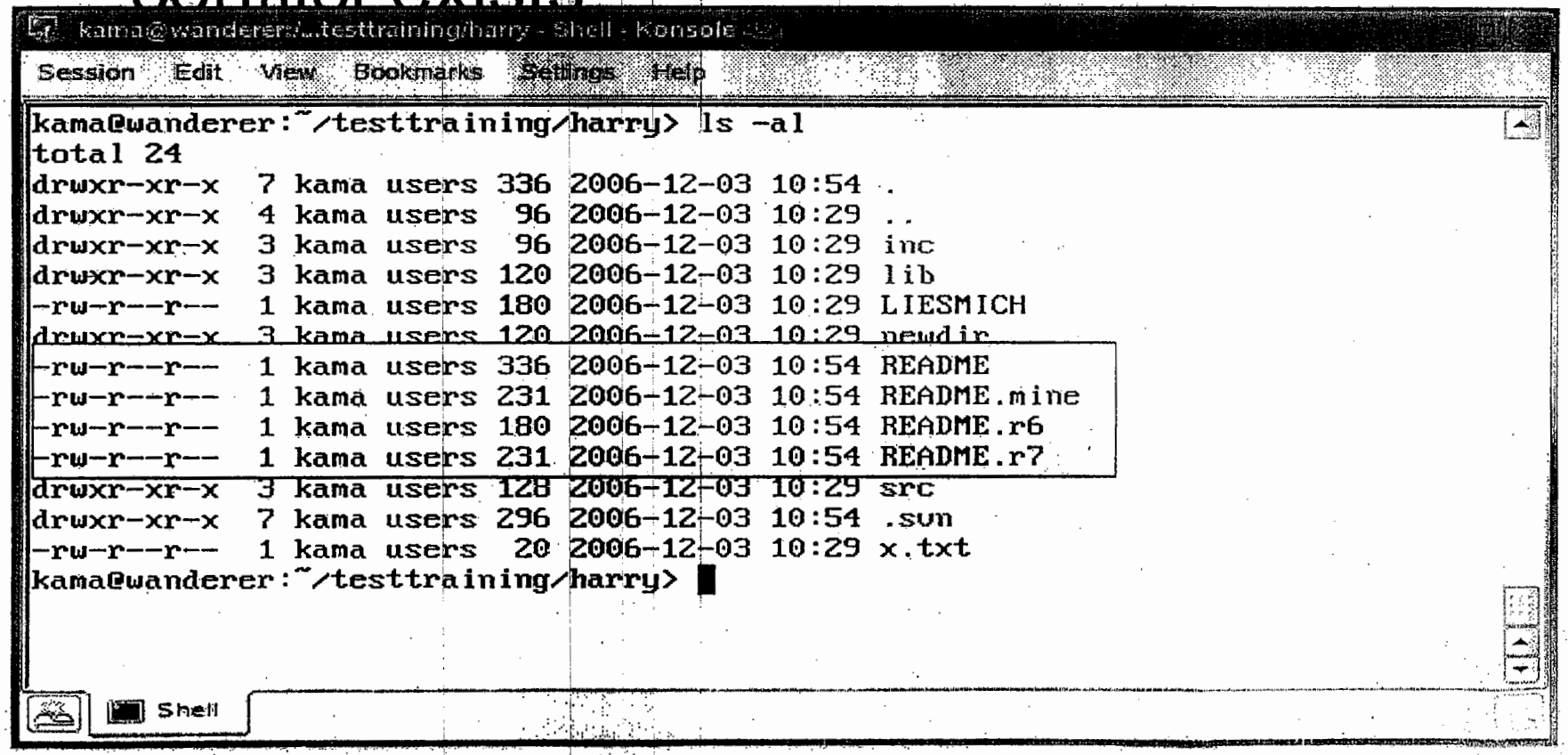
(3) override remote, use your local changes.

After resolving all conflicts, mark the file as "resolved".

- Subversion will delete the extra 3 copies.

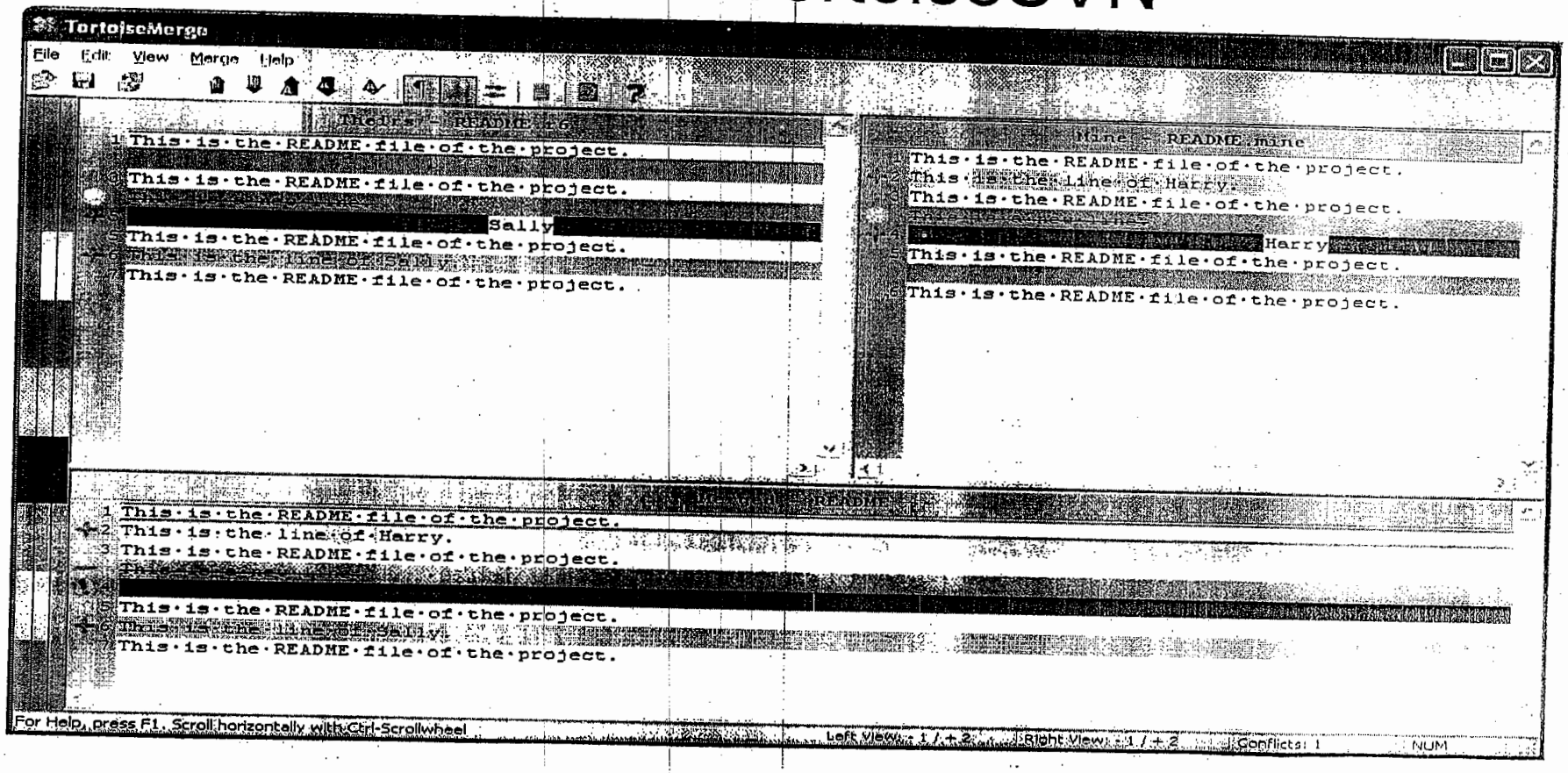
Conflict Support Files

Subversion client creates 4 files when a conflict exists



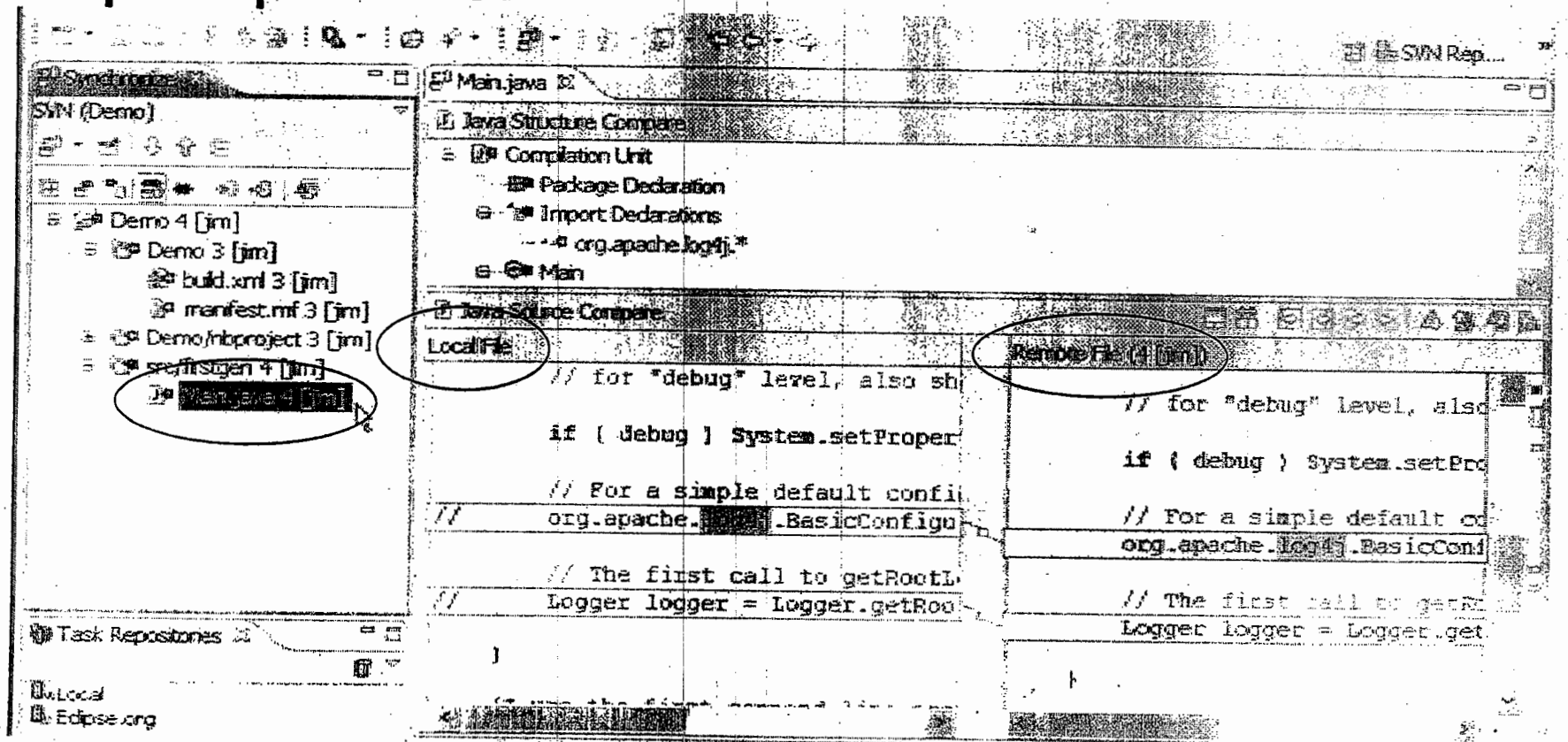
Resolving Conflicts with TortoiseSVN

Edit-Conflict tool of TortoiseSVN



Resolving Conflicts in Eclipse

- Use "Compare Editor" in Synchronize perspective.

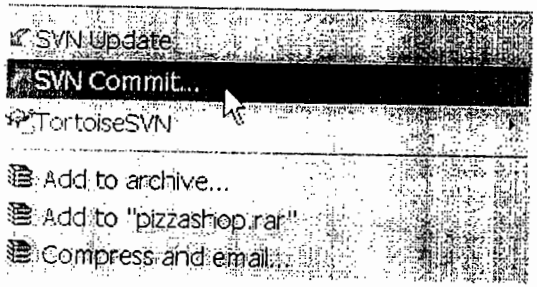


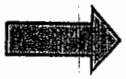
Work Cycle: Commit

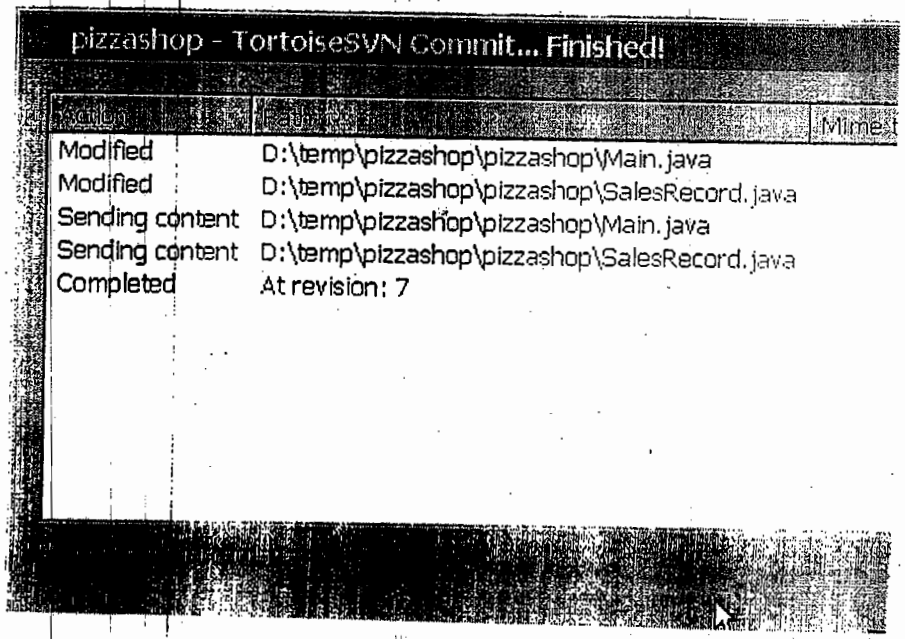
- After "update" and resolving conflicts, commit your work.
- Command line:

```
svn commit -m "description of your changes"
```

- TortoiseSVN:







Commit in IDE

Eclipse:

- right click on file or project => Commit

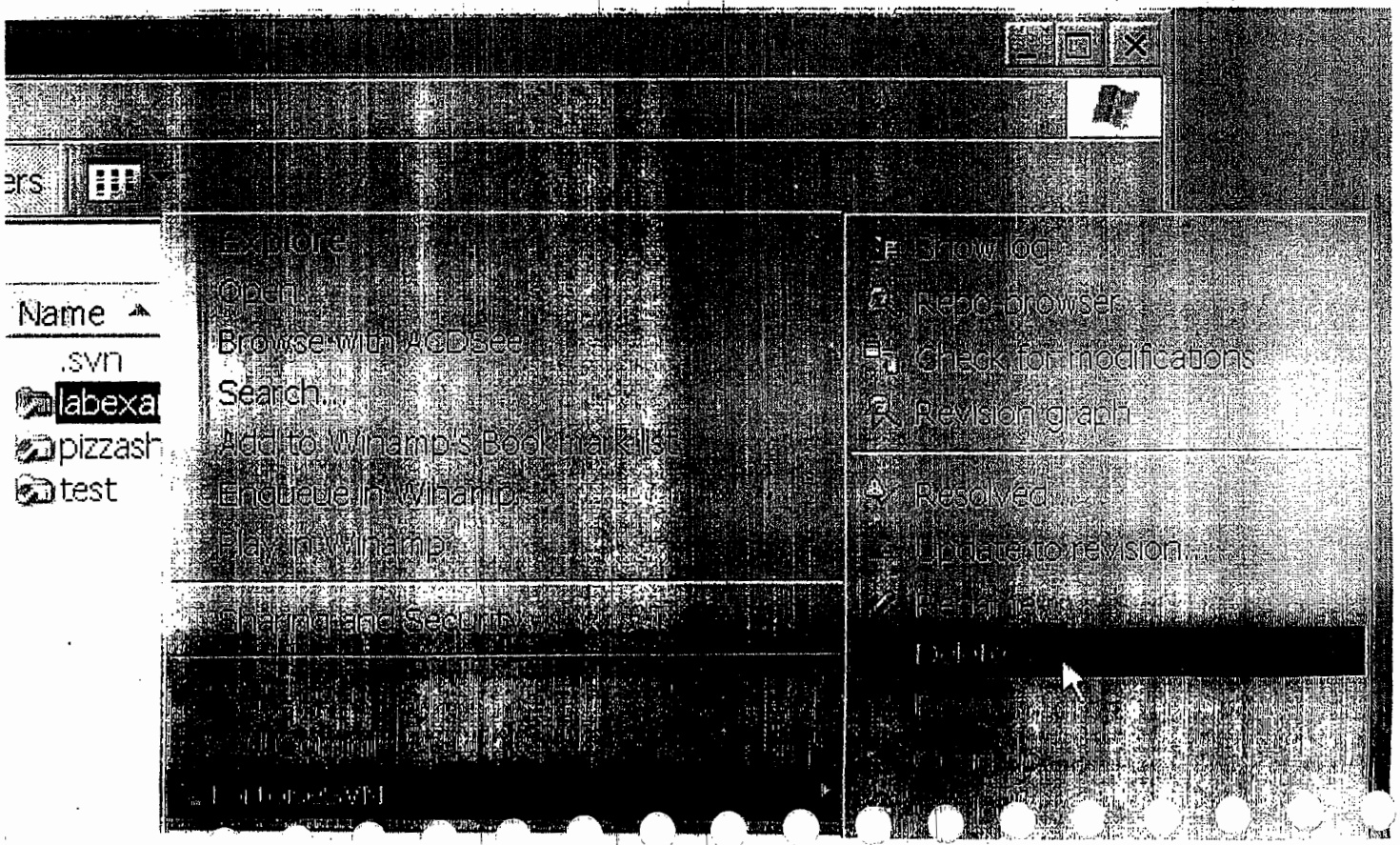
NetBeans:

- Team menu => Commit...

Move, Rename, Delete via TortoiseSVN

TortoiseSVN integrates into Windows Explorer.

- Right click on file to open menu.



Best practices

- Best practices for the correct-usage
- Avoid conflict over multi-user repository
- Enforce control and cooperation among team

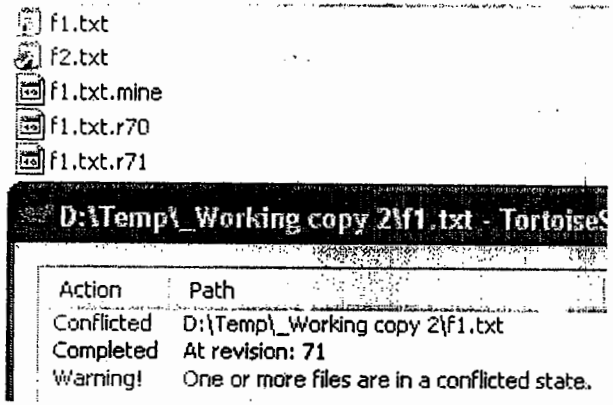
- **Give comments for the commit**
  - Professional working style.
  - Give the meaning for the update you made.
  - Help to find the desired revision quickly.

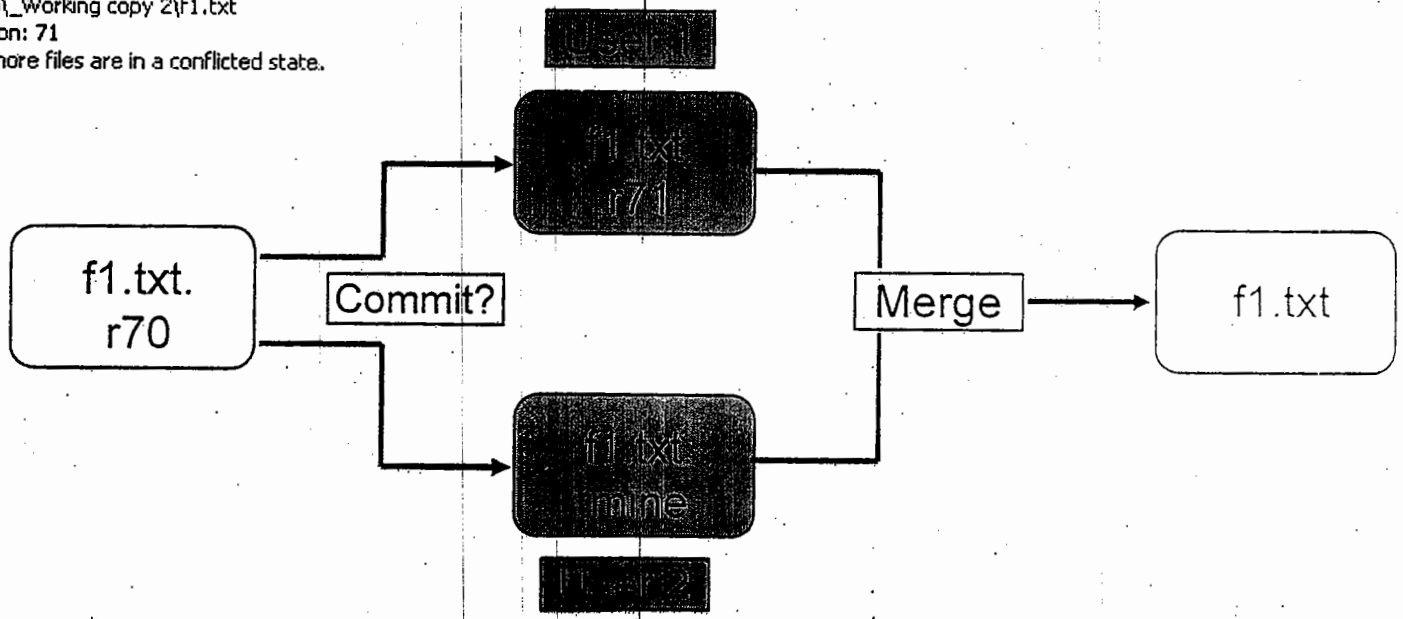
- **Update latest version before committing**
  - To assure your update is made on the latest version of the repository.
  - Right-click>SVN Update

Best practices

- **Self-manage the merging, do not let TortoiseSVN to merge files**
  - In case there is conflict between your working copy and the repository, you should merge file by yourself.
  - Steps:
    - Perform Update => TortoiseSVN creates 3 files: your original file, the latest file from repository and the SVN merged file.
    - Don't use the merged file.
    - Manually compare and merge your original file with latest file.
    - Overwrite your result onto the SVN merged file.
    - Commit your merged file to the repository.

Best practices



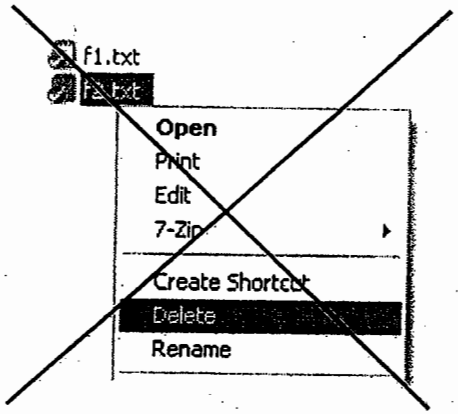


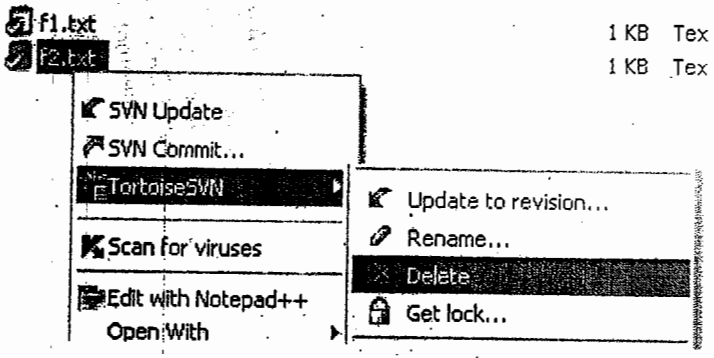
Best practices

- **Checkout-Lock-Edit-Commit-Unlock**
  (VSS: single checkout)
  - Apply the auto need-lock property.
  - How to use?
    - Before editing one file, perform Get lock => file is ready to update, also the repository file is locked so the other cannot commit their changes.
    - Commit your file after editing
    - Release lock => the other can edit and commit their changes.
    - **Do not** use Windows Explorer to set file attributes.

Best practices

- **Use TortoiseSVN commands (right click, drag and drop) don not use the Windows Explorer commands.**
  - This to keep the consistency between your working copy to the repository. The actions have effect on both your working copy and repository.





SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.

Jasper Reports

D:12/12/15

⇒ Jasper Reports is a Reporting tool for Java application.

⇒ Jasper Reporting is an open source reporting tool from Jaspersoft.

⇒ For Reporting capabilities, Jaw applications can use a repository tool use Jasper reports (or) dynamic reports (or) BIRT tool.

⇒ Jasper Reports is a tool which is completely developed in Java. so it is a platform.

⇒ Jasper Reports tool can't be used directly by end users.

⇒ Jasper Reports is a Popular reporting tool because of the following features

i) It can represent the data either in a text (or) graphics format.

ii) It can generate reports by collecting the data from dataBase, data source (db) (or) xml (or) xl (or) java etc...

iii) It can generate watermarkers to make report as confidential.

iv) It can export a report into a variety of formats like pdf, xL, HTML etc.

v) Jasper Reports can be used to generate reports for standalone applications (and) server side applications also.

⇒ We can download Jasper Reports Library and JasperReports studio from the location

http://community.jaspersoft.com/download

⇒ Download Japer Reports - 6.x - project.zip and then extract the zip file.

⇒ Download TIBCOJaspersoftstudio-6.2.0.final-windows-installer-x86.exe

Report Layouts -

A Reports layout in Jasper Reports contains seven sections

1). Title
2). Page Header
3). Column Header
4). Details
5) Column Footer
6). Page Footer
7). Summary

⇒ In a report detail section is mandatory and all the remaining are optional

⇒ Title comes only at on top of first page of the report and summary come on the bottom of the last page of the report

Different Phases of a Report:—

1). Design Phases
2). Compile Phases
3). Execution Phases
4). Export Phases.

1). **Design Phase:—** A report layout is created by constructing or creating .jrxml file.

⇒ Creating .jrxml file manualy is a complex work to the developer, so we use jaspersoft studio, which is a GUI tool, for creating .jrxml files

2). **Compile Phases:—** .jrxml file is compile and a Jasper Report object is generated.

3). **Execution Phases:—** A compile Report is filled with data and a jasper print object is generated.

4). Export phases:

A report is exported to different format

⇒ To compile, execute and export, we use Manager class provided by jasper reports api.

1. Jasper Compiles Manager
2. Jasper fill Manager
3. Jasper Export Managers.

```
JasperReport jr = JasperCompileManager.CompileReport
                  ("empReport.jrxml"); → Compile Phase

JasperPrint jp = JasperFillManager.fillReport(jr, ...);
                  // execution phase

JasperExportManager.exportReport(jp, "emps.pdf");
                  // export phase.
```

Jasper Report

D: 13/12/2015

Example 1:-

⇒ This Example generates a report by reading the data from xml file.

⇒ Prepare a contacts.xml like the following

```
<contacts>
    <contact>
        <name> scott </name>
        <mobile> 9912345678 </mobile>
        <email> scott@yahoo.co.in </email>
    </contact>
    <contact>
        <name> DHANUSH </name>
        <mobile> 8882223339 </mobile>
        <email> dhanush@gmail.com </email>
    </contact>
    <contact>
        <name> javaguru </name>
        <mobile> 9912543578 </mobile>
        <email> javaguru@gmail.com </email>
    </contact>
    <contact>
        <name> sekhar </name>
        <mobile> 9912543454 </mobile>
```

```
        <email> sekhar818@yahoo.co.in </email>
    </contact>
    <contact>
        <name> sathya </name>
        <mobile> 04065538958 </mobile>
        <email> www.sathyatech.com </email>
    </contact>
</contacts>
```

Steps to write

Step1):- Open Jaspersoft Studio.

Step2):- file ⇒ new ⇒ Jasper Report

⇒ select Blank A4 template

⇒ enter file name as report1.jrxml

⇒ Finish.

Step3):- Select Repository View at left side.

⇒ Right click on Data Adaptors

⇒ Create Data Adapter

⇒ select xml document

⇒ enter name xml Data Adapter

⇒ select xml file ⇒ contacts.xml

select Radio Button.

◉ Use the report Xpath xml expression when filling the report

⇒ test Button

⇒ Finish.

Step4): Click on Data set icon.

⇒ select xml Data Adapter on top

⇒ select Language ▼ → xPath

⇒ doreble clicks on Contacts and then contact

⇒ Read Fields Button on top

⇒ OK.

Step5): delete page Header, Column Footer, Page Footer and summary sections.

(by moving up the cursor)

⇒ Drag and drop Static text from Right side into the Title and change the Text to title as

xml Contacts Report

Step6): At left side, in the outlook view expand fields.

⇒ drag and drop name, mobile, email.

into detail section.

⇒ automatically Column headers are added

⇒ Change the appearance

after column header like font, size, type, color etc.... by Right Click on the column name and show properties.

Step7): Create a java Project in eclipse. with name as Example1.

⇒ copy report1.jrxml into src

By Right click on the Reports we will get the location, and copy the xml file from the location and paste.

⇒ Now Add the jasper Report jars to the project

Steps): Right click on Project

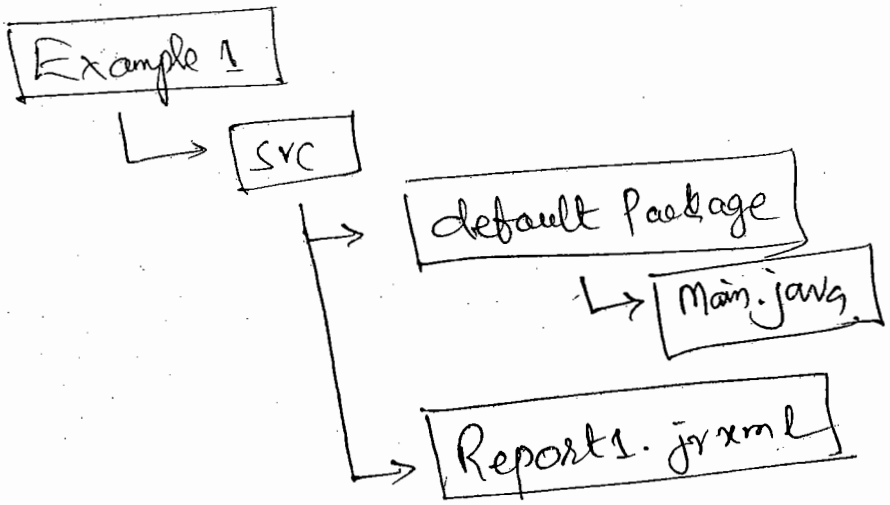
Example1 ⇒ Build Path ⇒ Configure Build Path

libraries ⇒ Add External Jars

⇒ select

commons beanutils
commons - collections
commons - digester
commons - logging
} from lib folder

itext, xalan
jaspersreports → from dist folder
xalan xalan (from lib folder)



⇒ Refer Handout page 2 of Example Programmes

Main.java from line no. 90 to 122.

Example 2

In this example we are generating a report which contains data employees, and also summary contains a chart (pie-3rd chart), which shows the no. of employees working in each department.

Step1):- Open Jasper Studio.

Step2):- file → new → Jasper Report

→ Select (Blank A4

→ next

→ enter file name sub-report.jrxml.

→ next

→ finish.

Step3):→ Click on Report Explorer on left side

Right click on Data Adapter

→ create new Data Adapter

→ select DataBase JDBC connection

→ next

→ enter name

Oracle Data Adapter

→ enter Driver: oracle.jdbc.driver.OracleDriver

URL: jdbc:oracle:thin:@localhost:1521:xe

username: System

password: admin.

⇒ Select Drivers Class Paths

⇒ Add ⇒ Select ojdbc 6.jar

⇒ Test

⇒ it displays successful message

⇒ Finish

Step 4): Click on dataset icon

⇒ Select Oracle Data Adapter on top

⇒ Select language SQL

⇒ Type the SQL command in the box

Select deptno, count(*) as empscount

from emp group by deptno;

⇒ Read fields

⇒ OK.

Step 5): ⇒ delete title, page header, column header, detail, column footer, page footer sections

in Right side ⇒ Palettee

drag and drop the chart tool into summery section

select pie-3rd chart.

⇒ next

⇒ enter Key, Value, label values as

click on Browse button of Key

⇒ select DeptNO

⇒ click on Browse button of Value

select emps-count

⇒ click on Browse button of label & enter

```
$F{DEPTNO}.toString()+'='+$F{EMPS-COUNT}.toString()
```

⇒ finish

we will get the chart in summary section

Step6):- in Eclipse

file ⇒ new ⇒ Jasper Report

⇒ select Blank A4

⇒ next

⇒ enter file name

~~main.jr~~.. main_report.jrxml

⇒ next

⇒ Finish.

Step7):- Click on dataset icon

⇒ select Oracle Data Adapter on top

⇒ select Language SQL

⇒ type the command

Select * from emp

⇒ Read Fields ⇒ OK

Step 8) → delete Page Header, Column footer, Page footer,

from palette (Right side) drag and drop Static Text into Title. and change Title as EmployeeReport.

⇒ drage and drop fields from outline (left side) into details section.

EmpNO
empName
SAL
DeptNO
} drag and drop.

⇒ from Palette (right side)

drag and drop SubReport tools into Summary section.

⇒ Select an existing report by

⇒ click on select existing Report

⇒ select Report file

⇒ Select SubReport.jrxml

⇒ next

Select Radio button

◉ Use same JDBC Connection .....

⇒ finish

Step 9):-

In Eclipse create a dynamic web Project.

with Application name as Example 2

→ copy sub-report.jrxml, main-report.jrxml } web content
copy sub-report.jasper into src

not into src of Example 2

⇒ copy the following jars to lib folder

commons-beanutils
commons-collections
commons-digester
commons-logging
itext
xalan
jasperreports
ojdbc6
servlet-api.jar

groovy-all-2.4.3.
jcommon
jfreechart

Step 10): Right click on WebContent ⇒ new ⇒ jsp file/html

name as index.html ⇒ Finish.

index.html

```
<center>
  <a href="servlet1"> Click Here </a>
</center>
```

Step 11): Right click on src ⇒ new ⇒ Servlet

⇒ enter name as ServletExample ⇒

package name as pack1 ⇒ next

⇒ enter URL pattern as /servlet1

⇒ next ⇒ select doGet ⇒ finish.

Step 12): Add the following logic in doGet()

```
doGet() {
response.setContentType("application/pdf");
try {
Class.forName("oracle.jdbc.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:
        thin:@localhost:1521:xe", "system", "admin");

JasperReport jr = JasperCompileManager.compileReport(
    getServletContext().getRealPath("/") +
    "main_report.jrxml");
JasperPrint jp = JasperFillManager.fillReport(jr, new
        HashMap(), con);
byte[] byteo = JasperExportManager.exportReportToPdf(jp);
    sos.write(byteo);
    sos.close();
}
catch(Exception e) {
  e.printStackTrace();
}
}
```

Notes → Copy the xml files

1) sub-report.jrxml
2) main-report.jrxml

and paste into webcontent

Note:- Copy sub-report.jasper file int src.

SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.

Sathya Technologies, Ameerpet

Jasper Reports

By: Sekhar

What is JasperReports?

JasperReports is an open-source Java class library designed to aid developers with the task of adding reporting capabilities to Java applications. Since it is not a standalone tool, it cannot be installed on its own. Instead, it is embedded into Java applications by including its library in the application's CLASSPATH. JasperReports is a Java class library, and is not meant for end users, but rather is targeted towards Java developers who need to add reporting capabilities to their applications.

Although JasperReports is primarily used to add reporting capabilities to web-based applications via the Servlet API, it has absolutely no dependencies on the Servlet API or any other Java EE library. It is, therefore, by no means limited to web applications. There is nothing stopping us from creating standalone desktop or command-line Java applications to generate reports with JasperReports. After all, JasperReports is nothing but a Java class library providing an API to facilitate the ability to generate reports from any kind of Java application.

JasperReports requires a **Java Development Kit (JDK) 1.3** or newer in order to successfully compile applications incorporating the JasperReports Java class library, and a **Java Runtime Environment 1.3** or newer to successfully execute these applications. Older versions of JasperReports required a JDK to successfully execute JasperReports applications (strictly speaking, JasperReports required tools.jar to be in the CLASSPATH, and tools.jar is included in the JDK, not the JRE).

JasperReports was started by **Teodor Danciu, in 2001**, when he was faced with the task of evaluating reporting tools for a project he was working on. He registered the project on http://sourceforge.net in September, 2001. JasperReports version 0.1.5 was released in November, 2001. Since then, JasperReports has become immensely popular, and is currently one of the most popular (if not the most popular) Java reporting tools available. As a testament to JasperReports' enormous popularity, a Google search for **java reporting tool** returns the JasperReports website as its first result.

Features of JasperReports

In addition to textual data, JasperReports is capable of generating professional reports including images, charts, and graphs. Some of the major JasperReports features include:

- It has flexible report layout.
- It is capable of presenting data textually or graphically.
- It allows developers to supply data in multiple ways.
- It can accept data from multiple datasources.
- It can generate watermarks.
- It can generate subreports.
- It is capable of exporting reports to a variety of formats.

Flexible Report Layout

JasperReports allows us to separate data into optional report sections.These sections include:

- The report title, which will appear once at the top of the report.
- A page header, which will appear at the top of every page.
- A detail section, which typically contains the primary report data.
- A page footer, which will appear at the bottom of every page.
- A summary section, which will appear at the end of the report.

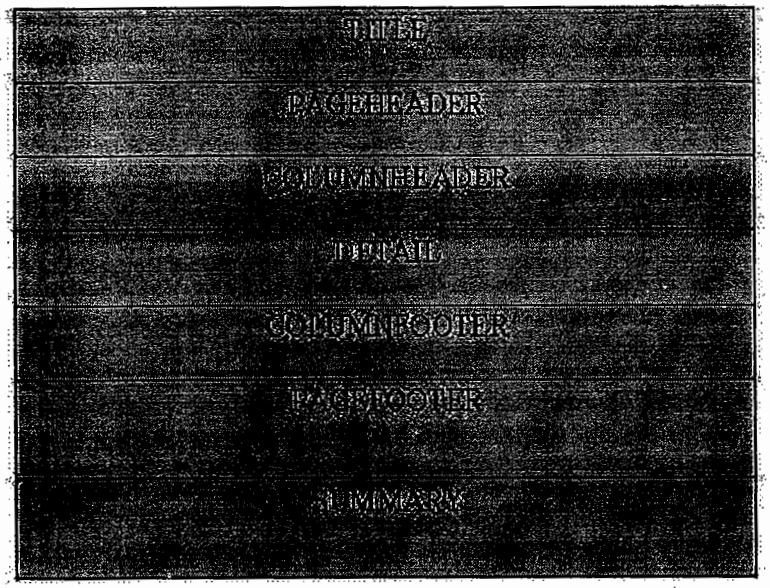
JasperReports allows the creation of elaborate dynamic layouts based on the contents of the report. For example, depending on the value of a report field, data can be hidden or displayed in a report, or data can be grouped into logical sections.

What is Report

A report is meaningful and well summarized information of data from the database. Usually the routine activities are automated and data summarized into a decision-supporting format "Reports". Reports act as wonders when it converts the usual messy data into charming charts, graphs and other graphical representations.

Report template

Generally the following report layout is followed to generate reports by many of the commercial report generating tools.



Following is the description of each element mentioned in the diagram.

Element	Description
title	Title contains the title of the report. It appears only once at the very beginning of the report, for example, "SathyaStudentReport".
pageHeader	PageHeader may contain date and time information and/or organization name. This appears at top of each page.
columnHeader	ColumnHeader lists the names of those specific fields which you want to display in the report, for example, "Author Name", "Starting Hour", "Finishing Hour", "Hours Worked" and "Date" etc.
detail	Detail is the part where entries of the specific fields (listed in columnHeader) are shown, for example "sathya", "9:00", "18:00", "9", "10.02.2013".
columnFooter	ColumnFooter may display summation of any of the fields, for example, "Total Hours Worked: 180"
pageFooter	PageFooter may contain page count information. It appears at the bottom of each page, for example, "1/23".
summary	Summary contains information inferred from "detail" part, for example, After the number listing of worked hours for each author, total hours worked for each author can be put in visual chart like pie chart, graph, etc for better comparison.

Important Tags of JRXML

The JRXML templates (or JRXML files) in JasperReport are standard XML files, having an extension of .jrxml. All the JRXML files contain tag <jasperReport>, as root element. This in turn contains many sub-elements (all of these are optional). JasperReport framework can handle different kinds of data sources. In this tutorial we shall show how to generate a basic report, just by passing a collection of Java data object (using Java beans), to the JasperReport Engine.

asperReports' reports are defined in XML files, which by convention have an extension of **jrxml**. A typical **jrxml** file contains the following elements:

- **<jasperReport>** - the root element.
- **<title>** - its contents are printed only once at the beginning of the report
- **<pageHeader>** - its contents are printed at the beginning of every page in the report.
- **<detail>** - contains the body of the report.
- **<pageFooter>** - its contents are printed at the bottom of every page in the report.
- **<band>** - defines a report section, all of the above elements contain a band element as its only child element.

All of the elements are optional, except for the root **jasperReport** element. Here is an example jrxml file that will generate a simple report displaying the string "Hello World!"

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport
PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="Simple_Report">
<detail>
        <band height="20">
                <staticText>
                        <reportElement x="180" y="0" width="200" height="20"/>
                        <text><![CDATA[Hello World!]]></text>
                </staticText>
        </band>
</detail>
</jasperReport>
```

Jasper Managers classes

net.sf.jasperreports.engine.JasperCompileManager

This is the class that exposes all the library's report compilation functionality. It has various methods that allow the users to compile JRXML report templates found in files on disk or that come from input streams. It also lets you compile in-memory report templates by directly passing a **net.sf.jasperreports.engine.design.JasperDesign** object and receiving the corresponding **net.sf.jasperreports.engine.JasperReport** object. Other utility methods include report template verification and JRXML report template generation for in-memory constructed net.sf.jasperreports.engine.design.JasperDesign class instances. These instances are especially useful in GUI tools that simplify report design work. *net.sf.jasperreports.engine.JasperCompileManager:* Used to compile a JRXML report template.

public String compileToFile(String sourceFileName) throws JRException
Compiles the XML report design file specified by the parameter. The result of this operation is another file that will contain the serialized JasperReport object representing the compiled report design, having the same name as the report design as declared in the XML plus the *.jasper extension, located in the same directory as the XML source file.

public JasperReport compile(String sourceFileName) throws JRException
Compiles the XML report design file received as parameter, and returns the compiled report design object.

public void compileToStream(InputStream inputStream, OutputStream outputStream) throws JRException
Compiles the XML representation of the report design read from the supplied input stream and writes the generated compiled report design object to the output stream specified by the second parameter.

public void writeToXmlFile(String sourceFileName, String destFileName) throws JRException
Generates the XML representation of the report design loaded from the first file parameter and place it in the file specified by the second parameter. The result is "UTF-8" encoded.

net.sf.jasperreports.engine.JasperFillManager

This class is the facade to the report-filling functionality of the JasperReports library. It exposes a variety of methods that receive a report template in the form of an object, file, or input stream, and also produces a document in various output forms (object, file, or output stream). Along with the report template, the report-filling engine must also receive data from the data source, as well as the values for the report parameters, to generate the documents. Parameter values are always supplied in a java.util.Map object, in which the keys are the report parameter names.

The data source can be supplied in two different forms, depending on the situation. Normally, it is supplied as a net.sf.jasperreports.engine.JRDataSource object, as just mentioned. However, since most reports are filled with data from relational databases, JasperReports has a built-in default behavior that lets people specify an SQL query in the report template itself. This SQL query is executed to retrieve the data for filling the report at runtime.

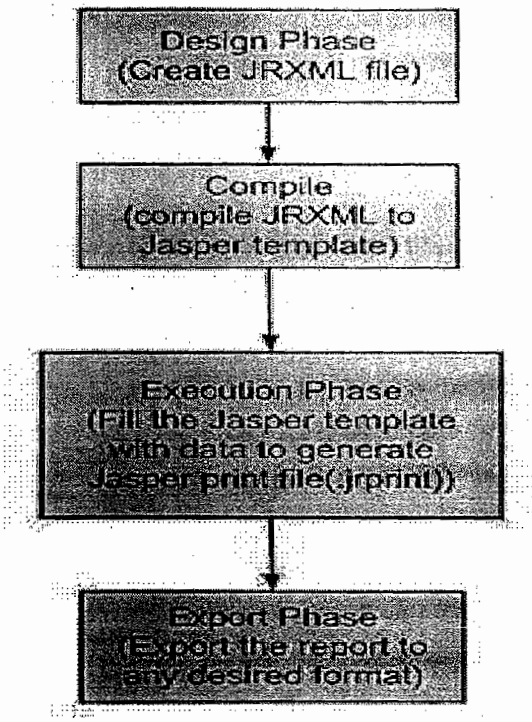
In such cases, the only thing JasperReports needs is a java.sql.Connection object, instead of the usual data source object. It needs this connection object to connect to the relational database management system through JDBC and execute the report query. It automatically creates a net.sf.jasperreports.engine.JRResultSetDataSource behind the scenes to wrap the java.sql.ResultSet object returned after the execution of the query and passes it to the normal report-filling process.
net.sf.jasperreports.engine.JasperFillManager: Used to fill a report with data from a datasource

net.sf.jasperreports.engine.JasperExportManager

As already mentioned, JasperReports can transform generated documents from its proprietary format into more popular documents formats such as PDF, HTML, or XML. Over time, this part of the JasperReports functionality has been extended to support other formats, including RTF, XSL, and CSV.
This manager class has various methods that can process data that comes from different sources and goes to different destinations (files, input and output streams, etc.).
*net.sf.jasperreports.engine.JasperExportManager:*Used to obtain PDF, HTML, or XML content for the documents produced by the report-filling process

Jasper Report - Life Cycle

The main purpose of JasperReports is to create page oriented, ready to print documents in a simple and flexible manner. The following flow chart depicts a typical work flow while creating reports.



As in the image the life cycle has following distinct phases

1. Designing the report In this step we create the JRXML file, which is an XML document that contains the definition of the report layout. We can use any text editor or iReportDesigner to manually create it. If iReportDesigner is used the layout is designed in a visual way, hence real structure of the JRXML can be ignored.
2. Compiling the report In this step JRXML is compiled in a binary object called a Jasper file(*.jasper). This compilation is done for performance reasons. Jasper files are what you need to ship with your application in order to run the reports.
3. Executing the report(Filling data into the report) In this step data from the application is filled in the compiled report. The class net.sf.jasperreports.engine.JasperFillManager provides necessary functions to fill the data in the reports. A Jasper print file (*.jrprint) is created, which can be used to either print or export the report.
4. Exporting the report to desired format In this step we can export the Jasper print file created in the previous step to any format using JasperExportManager. As Jasper provides various forms of exports, hence with the same input we can create multiple representations of the data.

```
import java.io.*;
import java.sql.*;
import java.util.*;
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.design.JasperDesign;
import net.sf.jasperreports.engine.xml.JRXmlLoader;

public class ReportGenerator {
Connection conn;
public void generateReport() {
try {
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test?user=root&password=root");
System.out.println("Loading Report Designs");
InputStream input = new FileInputStream(new File("jrxml/SathyaReport.jrxml"));
JasperDesign jasperDesign = JRXmlLoader.load(input);

System.out.println("Compiling Report Designs");
JasperReport jasperReport = JasperCompileManager.compileReport(jasperDesign);

System.out.println("Creating JasperPrint Object");
Map<String, String> parameters = new HashMap<String, String>();
parameters.put("ReportTitle", "PDF JasperReport");

System.out.println("Filling Report to File");
JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, conn);

//Exporting the report
OutputStream output = new FileOutputStream(new File("jrxml/sathya.pdf"));
JasperExportManager.exportReportToPdfStream(jasperPrint, output);

System.out.println("Report Generation Complete");
conn.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (JRException e) {
e.printStackTrace();
} catch (ClassNotFoundException e) {
e.printStackTrace();
} catch (SQLException e) {
e.printStackTrace();
}
}

public static void main(String[] args) {
new ReportGenerator().generateReport();
}
}
```

```
    Title:  Generating  report with  xml file as data source
----------report.jrxml--------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
        http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="report3"
        language="groovy" pageWidth="595" pageHeight="842" columnWidth="555"
        leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20"
        uuid="83fcd3f0-dd18-4efa-8dda-270a2400e0de">
    <property name="ireport.zoom" value="1.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="0"/>
    <queryString language="xPath">
        <![CDATA[/records/contact]]>
    </queryString>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>
    <field name="phno" class="java.lang.String">
        <fieldDescription><![CDATA[phno]]></fieldDescription>
    </field>
    <field name="email" class="java.lang.String">
        <fieldDescription><![CDATA[email]]></fieldDescription>
    </field>
    <background>
        <band splitType="Stretch"/>
    </background>
    <title>
        <band height="37" splitType="Stretch">
            <staticText>
                <reportElement x="124" y="0" width="213" height="37"
                 uuid="a84b943c-dfb4-4be9-b595-2a93361bc72d"/>
                <textElement>
                    <font size="20" isBold="true"/>
                </textElement>
                <text><![CDATA[Report for Xml Data]]></text>
            </staticText>
        </band>
    </title>
    <columnHeader>
        <band height="35" splitType="Stretch">
            <staticText>
                <reportElement x="0" y="0" width="185" height="20"
                uuid="03b734d9-85e2-41bf-b673-db91146a86f1"/>
                <textElement>
                    <font size="14" isBold="true"/>
                </textElement>
                <text><![CDATA[     Name]]></text>
            </staticText>
            <staticText>
                <reportElement x="185" y="0" width="185" height="20"
                uuid="5346f94d-699c-4d8b-8604-1eadb1dc43f8"/>
                <textElement>
                    <font size="14" isBold="true"/>
                </textElement>
                <text><![CDATA[phno]]></text>
            </staticText>
            <staticText>
                <reportElement x="370" y="0" width="185" height="20"
                uuid="a9f5483a-5346-4f26-826c-371183f5d77b"/>
                <textElement>
                    <font size="14" isBold="true"/>
                </textElement>
                <text><![CDATA[email]]></text>
            </staticText>
```

```
            </band>
        </columnHeader>
        <detail>
            <band height="43" splitType="Stretch">
                <textField>
                    <reportElement x="0" y="0" width="185" height="20"
                    uuid="8c1474fb-846b-4ba1-b4e6-66d9b44a3950"/>
                    <textFieldExpression><![CDATA[$F{name}]]></textFieldExpression>
                </textField>
                <textField>
                    <reportElement x="185" y="0" width="185" height="20"
                    uuid="c767f664-35d6-4e99-adac-d2153c26cc23"/>
                    <textFieldExpression><![CDATA[$F{phno}]]></textFieldExpression>
                </textField>
                <textField>
                    <reportElement x="370" y="0" width="185" height="20"
                    uuid="788748c0-7c83-485f-b82b-5027f71f0f16"/>
                    <textFieldExpression><![CDATA[$F{email}]]></textFieldExpression>
                </textField>
            </band>
        </detail>
</jasperReport>
--------------------Main.java---------------------------------------
//Main.java
import java.util.HashMap;

import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.engine.data.JRXmlDataSource;
public class Main {
    public static void main(String[] args)
    {
        try
        {
            JRXmlDataSource  jds=new  JRXmlDataSource("D:/contacts.xml",
                                                      "/records/contact");
            //Empty hashmap object, because we don't require any parameters
            //values to the query
            HashMap  hashMap = new HashMap();
            //compile
            JasperReport  design=JasperCompileManager.compileReport("./src/report.jrxml")
            //fill or execute
            JasperPrint  jp = JasperFillManager.fillReport(design, hashMap,jds);
            //export
            JasperExportManager.exportReportToPdfFile(jp,"D:/contact.pdf");
            System.out.println("Report exported to pdf file");
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
============================================================
    Title:  Generating  a  report  with subreport  as a chart with  web application
----------------subreport.jrxml-------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
    <property name="ireport.zoom" value="1.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="0"/>
    <queryString>
        <![CDATA[select deptno,count(*)as empscount from emp group by deptno]]>
```

```
    </queryString>
    <field name="deptno" class="java.lang.Integer">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="empscount" class="java.lang.Long"/>
    <background>
        <band splitType="Stretch"/>
    </background>
    <summary>
        <band height="197" splitType="Stretch">
            <pie3DChart>
                <chart>
                    <reportElement x="88" y="27" width="236" height="146"
                    uuid="d1e61c2e-e9e0-40d3-a123-b6270aeb316d"/>
                    <chartTitle/>
                    <chartSubtitle/>
                    <chartLegend/>
                </chart>
                <pieDataset>
                    <keyExpression><![CDATA[$F{deptno}]]></keyExpression>
                    <valueExpression><![CDATA[$F{empscount}]]></valueExpression>
                    <labelExpression><![CDATA[$F{deptno}+' = '+$F{empscount}]]>
                    </labelExpression>
                </pieDataset>
                <pie3DPlot>
                    <plot/>
                    <itemLabel/>
                </pie3DPlot>
            </pie3DChart>
        </band>
    </summary>
</jasperReport>
----------mainreport.jrxml----------
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
    <property name="ireport.zoom" value="1.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="0"/>
    <parameter name="SUBREPORT_DIR" class="java.lang.String" isForPrompting="false">
        <defaultValueExpression><![CDATA["C:\\Users\\Sekhar\\Downloads\\"]]>
        </defaultValueExpression>
    </parameter>
    <queryString>
        <![CDATA[select  empno,ename,sal,deptno from emp]]>
    </queryString>
    <field name="empno" class="java.lang.Integer">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="ename" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="sal" class="java.lang.Integer">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="deptno" class="java.lang.Integer">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <variable name="sal_1" class="java.lang.Integer" resetType="Column"
            calculation="Highest">
        <variableExpression><![CDATA[$F{sal}]]></variableExpression>
    </variable>
    <background>
        <band splitType="Stretch"/>
    </background>
    <title>
        <band height="32" splitType="Stretch">
```

```
199                 <staticText>
200                     <reportElement x="143" y="0" width="218" height="25"
201                     uuid="afeae165-ac74-4605-b791-a8f9160577a0"/>
202                     <textElement>
203                         <font size="16" isBold="true"/>
204                     </textElement>
205                     <text><![CDATA[  Employees  Report]]></text>
206                 </staticText>
207             </band>
208         </title>
209         <columnHeader>
210             <band height="26" splitType="Stretch">
211                 <staticText>
212                     <reportElement x="15" y="2" width="100" height="20"
213                     uuid="c5bb27eb-0f46-4783-b435-084664140fe7"/>
214                     <textElement>
215                         <font size="14" isBold="true"/>
216                     </textElement>
217                     <text><![CDATA[empno]]></text>
218                 </staticText>
219                 <staticText>
220                     <reportElement x="162" y="2" width="100" height="20"
221                     uuid="7bb88742-8203-4d32-974e-8efb4aa6e9cb"/>
222                     <textElement>
223                         <font size="14" isBold="true"/>
224                     </textElement>
225                     <text><![CDATA[ename]]></text>
226                 </staticText>
227                 <staticText>
228                     <reportElement x="299" y="2" width="100" height="20"
229                     uuid="5fb78226-56e6-4242-9877-528ae4a7cf1e"/>
230                     <textElement>
231                         <font size="14" isBold="true"/>
232                     </textElement>
233                     <text><![CDATA[sal]]></text>
234                 </staticText>
235                 <staticText>
236                     <reportElement x="442" y="2" width="100" height="20"
237                     uuid="b031e183-bf5f-44f4-82c3-b14b03224b75"/>
238                     <textElement>
239                         <font size="14" isBold="true"/>
240                     </textElement>
241                     <text><![CDATA[deptno]]></text>
242                 </staticText>
243             </band>
244         </columnHeader>
245         <detail>
246             <band height="30" splitType="Stretch">
247                 <textField>
248                     <reportElement x="15" y="4" width="100" height="20"
249                     uuid="47743a1b-99fd-4ef0-ad35-f0b13be2cd6d"/>
250                     <textElement>
251                         <font size="12" isBold="true"/>
252                     </textElement>
253                     <textFieldExpression><![CDATA[$F{empno}]]></textFieldExpression>
254                 </textField>
255                 <textField>
256                     <reportElement x="162" y="3" width="100" height="20"
257                     uuid="82535289-99bb-4585-be5d-0e8bd800dba8"/>
258                     <textElement>
259                         <font size="12" isBold="true"/>
260                     </textElement>
261                     <textFieldExpression><![CDATA[$F{ename}]]></textFieldExpression>
262                 </textField>
263                 <textField>
264                     <reportElement x="299" y="3" width="100" height="20"
```

```
                    uuid="ceed15eb-8451-416a-b556-6d312741ceec"/>
                    <textElement>
                        <font size="12" isBold="true"/>
                    </textElement>
                    <textFieldExpression><![CDATA[$F{sal}]]></textFieldExpression>
                </textField>
                <textField>
                    <reportElement x="442" y="2" width="100" height="20"
                    uuid="845d9d28-c740-4502-abed-38762be26b22"/>
                    <textElement>
                        <font size="12" isBold="true"/>
                    </textElement>
                    <textFieldExpression><![CDATA[$F{deptno}]]></textFieldExpression>
                </textField>
            </band>
        </detail>
        <columnFooter>
            <band height="2"/>
        </columnFooter>
        <pageFooter>
            <band splitType="Stretch"/>
        </pageFooter>
        <summary>
            <band height="220" splitType="Stretch">
                <subreport>
                    <reportElement x="116" y="26" width="200" height="100"
                    uuid="d704ba7c-3acf-41e4-96ed-452d58ec9775"/>
                    <connectionExpression><![CDATA[$P{REPORT_CONNECTION}]]>
                    </connectionExpression>
                    <subreportExpression><![CDATA[$P{SUBREPORT_DIR} + "report2.jasper"]]
                    </subreportExpression>
                </subreport>
            </band>
        </summary>
</jasperReport>
---------- web.xml ----------
<web-app>
  <servlet>
    <servlet-name>sone</servlet-name>
      <servlet-class>Servlet1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>sone</servlet-name>
      <url-pattern>/chart</url-pattern>
  </servlet-mapping>
</web-app>
----------Servlet1.java----------
import  java.sql.*;
import  net.sf.jasperreports.engine.*;
import  java.util.*;
import  javax.servlet.*;
import  javax.servlet.http.*;
import  java.io.*;
public class Servlet1  extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response) throws
    ServletException,IOException
    {
        response.setContentType("application/pdf");
        response.setHeader("Content-disposition","attachment;filename=report.pdf");
        try
        {
            ServletOutputStream sos=response.getOutputStream();
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection(
                        "jdbc:mysql://localhost:3306/test","root","root");
```

```
331                 JasperReport jr=JasperCompileManager.compileReport(
332                                 getServletContext().getRealPath("/")+"mainreport.jrxml");
333                 JasperPrint jp = JasperFillManager.fillReport(jr,new HashMap(),con);
334                 byte pdfbytes[ ]=JasperExportManager.exportReportToPdf(jp);
335                 sos.write(pdfbytes);
336                 sos.close();
337             }
338             catch(Exception e)
339             {
340                 e.printStackTrace();
341             }
342         }
343 };
344
345
```