NEED OF LOG4J

BUG/Problem:- After Application is developed then it will be handover to End Client/Customer. At the time of Running Application some problems may occur while processing a request. In programming concept it is called as Exception.

These problems try to identify at the time of development to avoid few. Some will be handled once they are occurred. Those are analyzed and solved by developer. Give priority to Avoid problems if not handle the problems.

To avoid problems concepts are :   1. Validations   2. Standard Coding

To handle the problems concepts are: 1. Error pages   **2.Log4J**

1. Validations : At UI page like Register, Data Edit, Login, Feedback form, Comment etc.. at these levels we must use Java Script or equal Programming to avoid invalid input. Example:-

```
Register Page (JSP)
ID      [_____] Only numarics [0-9]
Name [_____] only characters [a-z]
Email [_____] Pattern __@__.__ format
```

Write Scripting code for above checking.
_____

2. Standard Coding : If we use coding standards we can reduce [30-40]% problems in application.

Examples : -

i) String s=(...) input is provided at run time
    int len=s.length();
 above code is converted to...
    String s=....;
    if(s!=null) int len=s.length();
    else sysout("No Valid Data");

ii) **int arr[]={....};**
    sysout(arr[4]);
may throw ArrayIndexOutOfBounds Exception above code is converted to...

```
   int arr[]={....};
   if(arr.length > 4){
    sysout(arr[4]);
   }else {
    sysout("Array Size is less");
   }
```

 iii) **Object ob=ses.get(T.class,id);**   (T=className provided at runtime)
    Address a=(Address)ob;
it may throw ClassCastException to avoid this
    if(ob instanceof Address){
     Address a=(Address)ob;
    } else {   sysout("ob is not address");   }

iv) List<Employee> has Employee objects read 1st index employee and find that employee object empName length

code
List<Employee> empList=....
Employee e=list.get(0);
String empName=e.getEmpName();
int len=empName.length();
  above code can be written as.....
if(empList!=null && empList.size>0 && empList.get(0)!=null){
  Employee e=list.get(0);
  if(e!=null && e.getEmpName()!=null) {
    int len=e.getEmpName().length();
  }
}
_____

3. Error Pages in Web Applications:- If server has encountered any problem then it will throw An equal HTTP Error. Example 404-Page Not Found, 500-Internal server Error, 400-Syntax Error, 405-Method Not Allowed, 401-UnAuthorized etc...

        To show one simple message to end user we must define JSP/HTML page and make them as isErrorPage="true" and configure in **web.xml** as

```
<error-page>
  <error-code>404</error-code>
  <location>/error.jsp</location>
</error-page>
<error-page>
 <exception-type>java.lang.NullPoniterException</exception-type>
 <location>/showError.jsp</location>
</error-page>
```

Create on "error.jsp" ex:

```
<%@ page ... isErrorPage="true"%>
<html><body> Welcome to Error Page ..Wait 5 sec to redirect to Home Page..
<%response.setHeader("Refresh", "5;index.jsp"); %> </body> </html>
```

Log4J

1. It is a Tracing or Logging Tool used in Specially in Production Environment. It is used to find success messages, information, warnings, errors in application while using it.
2. By Default any Message/Error will be printed on Console, which is temporary location, it can show only few lines like last 100 lines.
3. To see all problems from beginning to till date use Log4j concept.
4. Log4J can write problems/error to File(.log), Database, Email, Network etc..
5. Log4J provides Error Details like Exception type, class and method with line number it occured, Date and time also other information ..
6. Log4J also supports writing errors to Console also.
7. Using System.out.println prints only on console, but not to file or any other memory.

Log4J has 3 components:

Layout	Appender	Logger

1. Logger (LOG) Object: This object must be created inside the class as a instance variable. It is used to enable Log4J service to current class. If this object is not created in the class then Log4J concpet will not applicable(will not work)for that class It has 5 priority methods with names (along with order)

Order	method	Name
1	debug(obj)	DEBUG
2	info(obj)	INFO
3	warn(obj)	WARN
4	error(obj)	ERROR
5	fatal(obj)	FATAL
-NA-	-NA-	OFF

NA : Not Applicable

a. debug(msg) : It prints a message with data. It is used to print a final result of process. Like EmpId after saved is : 2362.
b. info(msg) : It is used to print a simple message. Like process state-I done, if block end. Email sent etc..

    c.   warn(msg): It is used to print warning messages. Like Collection is not with Generic, local variable not used, resource not closed etc...

    d.   error(msg): It is used to print Exceptions like NullPointer, ArrayIndex, SQLException etc..

    e.   fatal(mgs) : It indicates very high level problem. Like Server/DB Down, Connection timeout, Network broken,   Class Not Found etc...

OFF is used to disable Log4J concept in application. Log4J code need to be deleted.

_____

2. Appender : It will provide details for "Where to print Message?". Means in what type of memories, messages must be stored. To write Logger Statements to

i.	File	use	FileAppender
ii.	Database	use	JdbcAppender
iii.	Email	use	SmtpAppender
iv.	Console	use	ConsoleAppender
v.	Network	use	Ftp(Telnet)Appender

In one Application we can use more than one appender also.

_____

3. Layout : It provide the format of message to be printed. Possible layouts are:

    i.   Simple Layout : Print message as it is

    ii.   HTML Layout : Print message in HTML format(<html><body>....)

    iii.   XML Layout: Print message in XML format (<Errors><Type>..<Message>..)

    iv.   Pattern Layout : Prints messages in given pattern. example pattern: Date-Time / Line Number : Class-method :- Message

In development mostly used Appender is FileAppender and Layout is Pattern Layout.

Log4J Example Programming:-

Step#1 Create Maven Project

    I.   Click on File Menu

    II.   Choose "new" option

    III.   Choose Maven Project

    IV.   Select "Create Simple Project" checkbox

    V.   Click on "Next" Button

    VI.   Enter GroupId, ArtifactId, version details

    VII.   Click on "finish" button

Step#2 Specify Log4J dependency in pom.xml

<dependency>
 <groupId>log4j</groupId>
 <artifactId>log4j</artifactId>
 <version>1.2.17</version>
</dependency>

Step#3 Create one class under "src/main/java" and write Log4J code.

ex: code

```java
package com.app;
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class Product {
      private static Logger log=Logger.getLogger(Product.class);
      public static void main(String[] args) {
            Layout layout=new SimpleLayout();
            Appender ap=new ConsoleAppender(layout);
            log.addAppender(ap);
            log.info("Hello");
      }
}
```

Run application "ctrl+F11" to see output.

Note:

1. To Create "Logger" class object to current class use static factory method getLogger(_.class).
It returns Logger Object and enable the Log4J to current class.

2. Create a Layout(Abstract Class) object   using it's implemented classes, list given below

    a. SimplLayout   (C)
    b. XMLLayout   (C)
    c. HTMLLayout   (C)

    d.   PatternLayout(C)

3. Create Appender(I) object using it's implemented classes and also provide layout details to Appender object.

    a.   ConsoleAppender   (C)
    b.   FileAppender   (C)
    c.   JdbcAppender   (C)
    d.   SmtpAppender   (C)

4. At last Appender object must be added to Logger Object then only it is considered, by using log.addAppender() method.

5. If No appender is provided to Logger object then Log4J throws WARNING as log4j:WARN No appenders could be found for logger (com.app.Product).

_____

Layouts: Layout Defines Format of Message to be displayed on UI/DB/Console. Possible Layout classes are listed below with description.

1. SimpleLayout : This class prints message as it is provided in log methods.

2. HTMLLayout : This class provides message in HTML format. To see full output of this. Copy output to .html file

Example Code:

```
package com.app;
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;


public class Product {
    private static Logger log=Logger.getLogger(Product.class);
    public static void main(String[] args) {
            Layout layout=new HTMLLayout();
            Appender ap=new ConsoleAppender(layout);
            Log.addAppender(ap);
```

```
              Log.debug("Hello");
              Log.info("Hello");
              Log.warn("Hello");
              Log.error("Hello");
              Log.fatal("Hello");

        }
}
```

Example: screen

Time	Thread	Level	Category	Message
0	main	DEBUG	com.app.Product	Hello
1	main	INFO	com.app.Product	Hello
1	main	WARN	com.app.Product	Hello
1	main	ERROR	com.app.Product	Hello
1	main	FATAL	com.app.Product	Hello

Log session start time Thu Sep 21 21:14:19 IST 2017

3. XMLLayout: It prints log messages in XML format. In above code modify Layout as XMLLayout and run as same.

Example Code:

```java
package com.app;

import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.XMLLayout;


public class Product {
      private static Logger log=Logger.getLogger(Product.class);
      public static void main(String[] args) {
              Layout layout=new XMLLayout();
              Appender ap=new ConsoleAppender(layout);
              log.addAppender(ap);
```

```
            Log.debug("Hello");
            Log.info("Hello");
            Log.warn("Hello");
            Log.error("Hello");
            Log.fatal("Hello");

    }
}
```

Output Looks as :

```
<log4j:event logger="com.app.Product" timestamp="1506008834191"
level="DEBUG" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="INFO" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="WARN" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="ERROR" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>

<log4j:event logger="com.app.Product" timestamp="1506008834192"
level="FATAL" thread="main">
<log4j:message><![CDATA[Hello]]></log4j:message>
</log4j:event>
```

4. PatternLayout : This class provides output pattern for a message that contains date, time, class, method, line number, thread name, message etc..

Observe Some example patterns given below, for complete details click on below link

(https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html)

a)Date&Time pattern

- %d   = date and time
  examples:
a) %d
b) %d {dd-MMM-yy hh:mm:ss SSS}
c) %d {dd-MM-yy hh:mm:ss SSS}
d) %d {HH:mm:ss}
e) Here meaning of every word used
f) in date pattern is,
g) dd   = date
h) MMM   = Month Name
i) MM   = Month number
j) yy   = Year last two digitis
k) yyy   = Year in 4 digits
l) hh   = Hours in 12 format
m) HH   = Hours in 24 format
n) mm   = Minutes
o) ss   = Secs
p) SSS   = MillSecs

- %C = Class Name
- %M   = Method Name
- %m = Message
- %p = Priority method name(DEBUG,INFO..)
- %L = Line Number
- %l = Line number with Link
- %n = New Line(next line)
- %r = time in milli sec.
- %% = To print one '%' symbol.
- we can also use symbols like - [] , /

One Example Pattern with all matching "%d-%C[%M] : {%p}=%m<%L> %n "

_____

Appender:- An appender provides the destination of log message. Here example appenders are

a) ConsoleAppender
b) FileAppender

c) JdbcAppender
d) SmtpAppender

Example of FileAppender:-

```java
package com.app;

import org.apache.log4j.Appender;
import org.apache.log4j.FileAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;


public class Product {
    private static Logger log=Logger.getLogger(Product.class);
    public static void main(String[] s)    throws Exception {
        Layout layout=new HTMLLayout();
        Appender ap=new FileAppender(layout,"myapp.log");
        log.addAppender(ap);

        log.debug("Hello");
        log.info("Hello");
        log.warn("Hello");
        log.error("Hello");
        log.fatal("Hello");
    }
}
```

log4j.properties

log4j.properties file: This file is used to specify all the configuration details of Log4J. Especially like Appender Details and Layout Details with Patterns and also root Logger details. This File contains details in key=value format (.properties). Data will be shown in below order like

1. rootLogger
2. appenders
3. layouts

a) In this file (log4j.properties) use symbol '#' to indicates comments.
b) We can specify multiple appenders in log4j.properties file Like 2 File Appenders, one JdbcAppender, One SmtpAppender etc..

c) Every Appender must be connected with layout

d) Appender name must be define before use at rootLogger level.

e) Appender name can be anything ex: abc,hello,sysout,file,db,email etc..

f) log4j.properties file must be created under src folder (normal project) or src/main/resource folder (maven project).

g) Make rootLogger = OFF to disable Log4J completely without deleting code in any class or properties file

h) log4j.properties file will be auto detected by Log4J tool. No special coding is required.

Example #1 Console Appender

Create one properties file under "src/main/resources" with name "log4j.properties"

I. Right click on "src/main/resources"

II. Choose "new"

III. Select "other.."

IV. Search and select "file" for option

V. Click on next

VI. Enter file name and click finish.

_____

Example : log4j.properties (code)

```
# Root logger details
log4j.rootLogger=DEBUG,stdout

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd} %p %c:%L - %m%n
```

Test class:-

```
package com.app;
import org.apache.log4j.Logger;
public class Product {
    private static Logger log=Logger.getLogger(Product.class);
        public static void main(String[] srs){
                log.debug("Hello");
                log.info("Hello");
```

```
                        log.warn("Hello");
                        log.error("Hello");
                        log.fatal("Hello");


        }
}
```

output:

```
2017-09-21 21:47:59 DEBUG com.app.Product:6 - Hello
2017-09-21 21:47:59 INFO com.app.Product:7 - Hello
2017-09-21 21:47:59 WARN com.app.Product:8 - Hello
2017-09-21 21:47:59 ERROR com.app.Product:9 - Hello
2017-09-21 21:47:59 FATAL com.app.Product:10 - Hello
```

Using of Logger methods:

1. debug() : to specify a message with value (message with result after operation)
2. info() : to indicate current state (block end, method started , service finished, email sent etc..)
3. warn() : to provide warning message (Variable not used, List has no generic, connection not closed)..
4. error/fatal : used in catch blocks. Here error - used for unchecked exception and   fatal - checked exception in general

example :- spring controller

```
package com.app.controller;
@Controller
public class HomeController {
      private static Logger log=Logger.getLogger(HomeController.class);

      @RequestMapping("/url")
      public String saveEmp(...){
            log.info("Save Method started");
            try{
                  log.info("Before save employee..");
                  int empId=service.save(emp);
                  log.debug("emp saved with id :"+empId);
            }catch(Exception e){
                  log.error(e); // log.fatal(e);
```

```
            }
            log.warn("EMPID not used in program");
            log.info("save service is at end");
            return "Home";
        }
}
```

Log4j Examples

pom.xml (to run all below examples)

```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>sathyatech</groupId>
        <artifactId>log4jexample</artifactId>
        <version>1.0</version>
        <dependencies>
                <dependency>
                        <groupId>log4j</groupId>
                        <artifactId>log4j</artifactId>
                        <version>1.2.17</version>
                </dependency>
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.6</version>
                </dependency>
                <dependency>
                        <groupId>javax.mail</groupId>
                        <artifactId>mail</artifactId>
                        <version>1.4</version>
                </dependency>
        </dependencies>

</project>
```
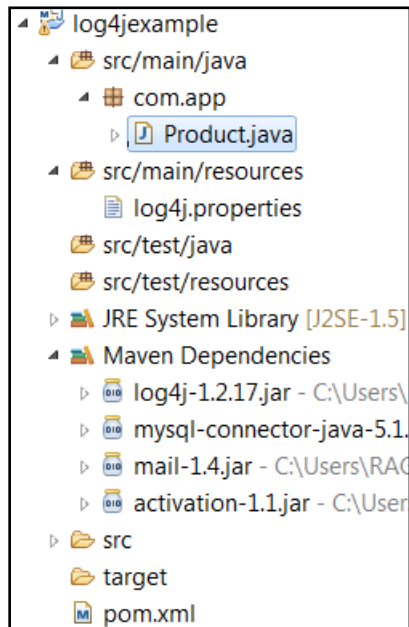
Example #1 Console Appender Example

Step#1 : Define one Test class with example code with log methods.

```java
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                  log.debug("Hello");
                  log.info("Hello");
                  log.warn("Hello");
                  log.error("Hello");
                  log.fatal("Hello");

        }
}
```

Step#2 Create log4j.properties file with below code

log4j.propertie:

```properties
# Root logger option
log4j.rootLogger=DEBUG, stdout
# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd } %p %c:%L - %m%n
```

Step#3 Run above Test class Product.java (ctrl+F11) to see output.

Folder Structure:-

Example#2 File Appender Example

Step#1 : Define one Test class with example code with log methods.

```java
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                    log.debug("Hello");
                    log.info("Hello");
                    log.warn("Hello");
                    log.error("Hello");
                    log.fatal("Hello");

        }
}
```

Step#2 Create log4j.properties file with below code

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, file
# Redirect log messages to a log file
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=logs/myapp.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %p %c:%L - %m%n
```

Step#3 Run above Test class Product.java (ctrl+F11) to see output.
_____

Example#3 JDBC Appender Example

Step#1 : Define one Test class with example code with log methods.

```java
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                log.debug("Hello");
                log.info("Hello");
                log.warn("Hello");
                log.error("Hello");
                log.fatal("Hello");

        }
}
```

Step#2 Create this table before you run example, all the logs will be stored in below table

```sql
CREATE TABLE LOGS (
        METHOD VARCHAR(20)        NOT NULL,
        DATED   DATETIME          NOT NULL,
        LOGGER  VARCHAR(50)        NOT NULL,
        LEVEL   VARCHAR(10)        NOT NULL,
        MESSAGE VARCHAR(1000)     NOT NULL
);
```

Step#3 Create log4j.properties file with below code

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, db

# Define the Jdbc appender
log4j.appender.db=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.db.driver=com.mysql.jdbc.Driver
log4j.appender.db.URL=jdbc:mysql://localhost:3306/test
log4j.appender.db.user=root
log4j.appender.db.password=root
log4j.appender.db.layout=org.apache.log4j.PatternLayout
log4j.appender.db.sql=INSERT INTO LOGS VALUES ('%M', now() ,'%C','%p','%m')
```

Step#4 Run above Test class Product.java (ctrl+F11) to see output.

_____

Example#4 SMTP Appender Example

**Step#1 : Define one Test class with example code with log methods.**

```java
package com.app;
import org.apache.log4j.Logger;
public class Product {
        private static Logger log=Logger.getLogger(Product.class);
          public static void main(String[] srs){
                log.debug("Hello");
                log.info("Hello");
                log.warn("Hello");
                log.error("Hello");
                log.fatal("Hello");

        }
}
```

Step#2 Create log4j.properties file with below code

**** Modify user Name and password before you run this application**

```
# Root logger option
log4j.rootLogger=DEBUG, email
#configuring the SMTP appender
log4j.appender.email=org.apache.log4j.net.SMTPAppender
log4j.appender.email.SMTPHost=smtp.gmail.com
log4j.appender.email.SMTPUsername=raghusirjava@gmail.com
log4j.appender.email.SMTPPassword=abdeyhfk@33
log4j.appender.email.From=raghusirjava@gmail.com
log4j.appender.email.To=<your email id>@gmail.com
log4j.appender.email.Subject=Log of messages
log4j.appender.email.Threshold=DEBUG
log4j.appender.email.layout=org.apache.log4j.PatternLayout
log4j.appender.email.layout.ConversionPattern= %m %n
```

Step#3 Configure this VM Argument before you run application
Right click in Editor and Choose "Run As" => "Run Configuration"
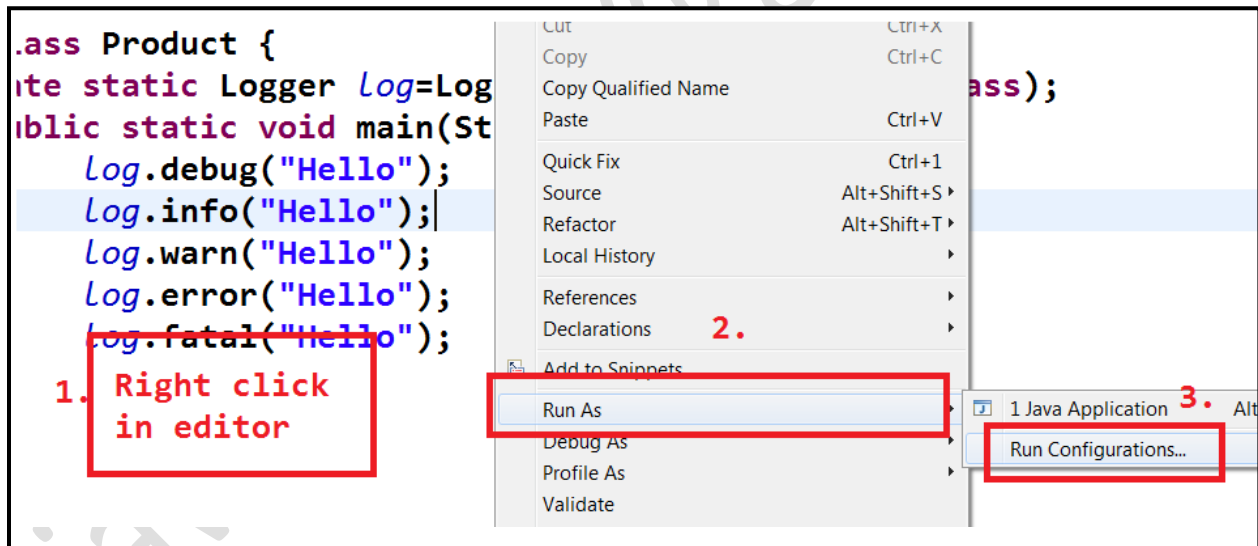
Enter this key=value in VM arguments in below screen

> -Dmail.smtp.starttls.enable=true
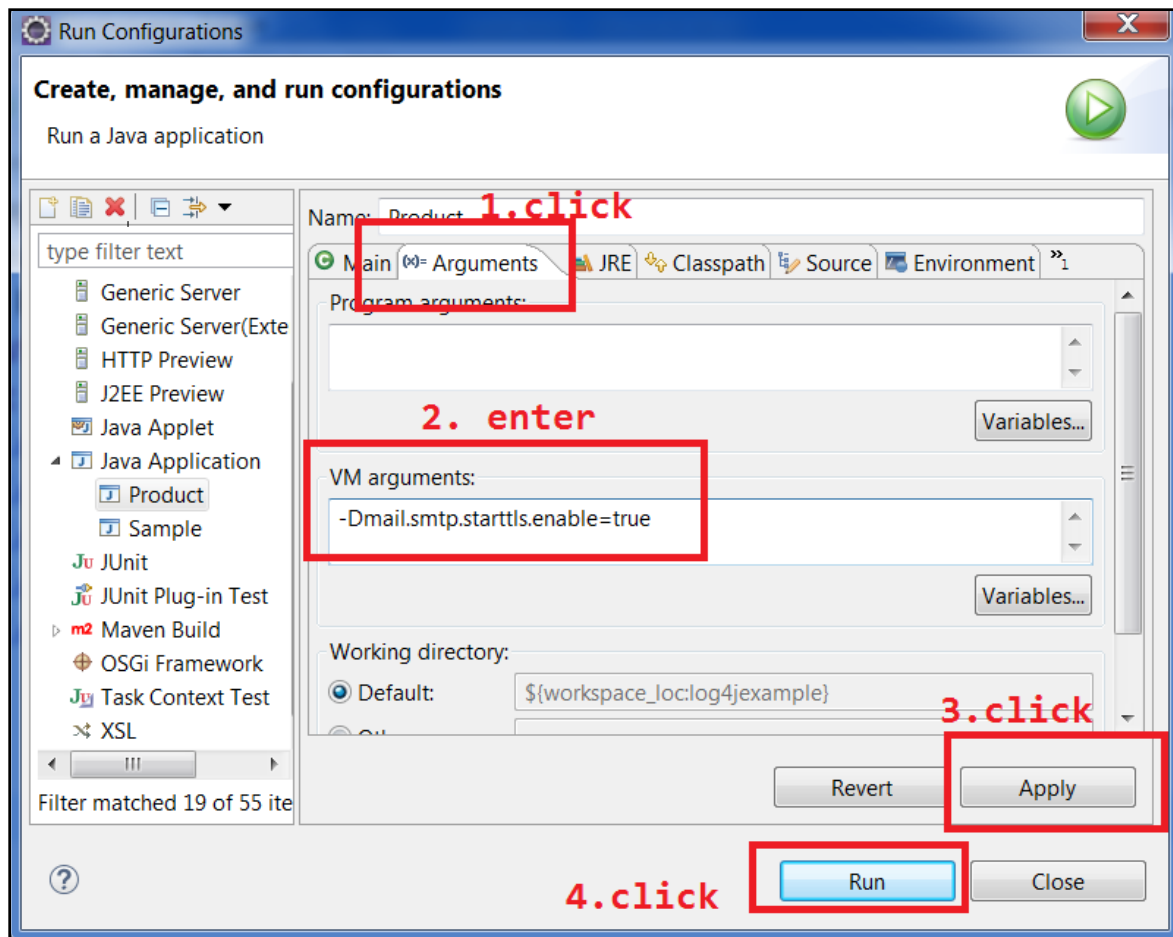
Step#4 Run above Test class Product.java (ctrl+F11) to see output. Or Click on Run Option shown in above screen

Facebook Group: https://www.facebook.com/groups/thejavatemple