CHANDAN MUKHERJEE

MTech (IT), BE (Computer Science)
SCJP (Java), Oracle (SQL) GLOBAL CERTIFIED
Microsoft Certified Innovative Educator
Corporate Trainer
chan.muk@gmail.com

Logging in Java

- logging is an important feature that helps developers to trace out the errors
- It provides a Logging API that was introduced in Java 1.4 version
- It provides the complete tracing information of the application.
- It records the critical failure if any occur in an application.

Log4j 2

- https://logging.apache.org/log4j/2.x/

Log4j 2

- Download log4j 2

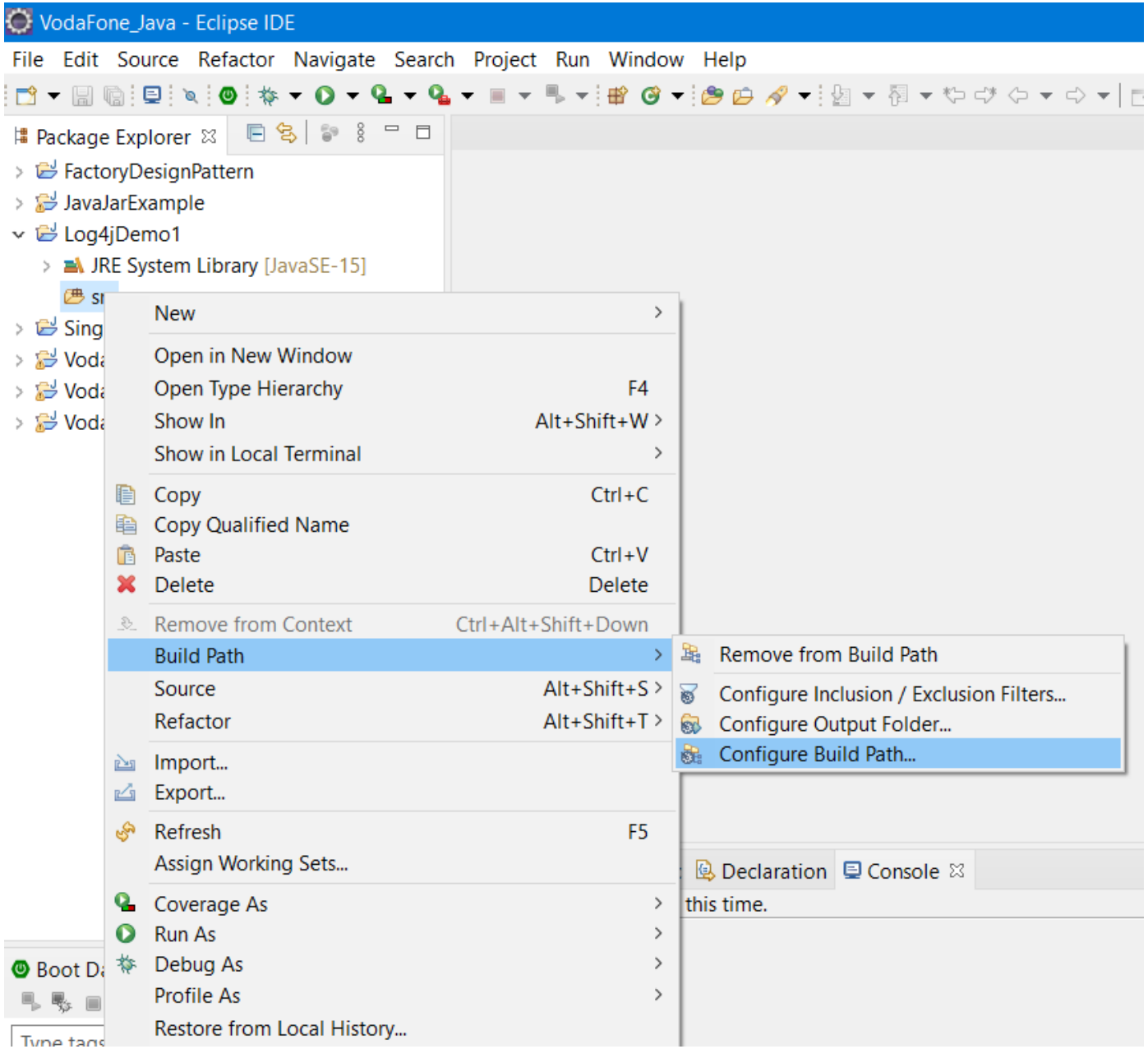
https://logging.apache.org/log4j/2.x/download.html

Using Log4j on your classpath

To use Log4j 2 in your application make sure that both the API and Core jars are in the application's classpath. Add the dependencies listed below to your classpath.

```
log4j-api-2.19.0.jar
log4j-core-2.19.0.jar
```

You can do this from the command line or a manifest file.


VodaFone_Java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
FactoryDesignPattern
JavaJarExample
Log4jDemo1
JRE System Library [JavaSE-15]
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Remove from Build Path
Configure Inclusion / Exclusion Filters...
Configure Output Folder...
Configure Build Path...
Declaration
Console
this time.

VodaFone_Java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- FactoryDesignPattern
- JavaJarExample
- Log4jDemo1
  - JRE System Library [JavaSE-15]
  - src
- SingletonDesignPattern
- VodafoneDay2
- VodaFoneJavaBasic
- VodaFoneJavaOOP

Boot Dashboard

Properties for Log4jDemo1

type filter text

- Resource
- Builders
- Coverage
- Java Build Path
- Java Code Style
- Java Compiler
- Javadoc Location
- Java Editor
- Namespaces
- Project Facets
- Project Natures
- Project References
- Run/Debug Settings
- Server
- Task Repository
- Task Tags
- Validation
- WikiText

Java Build Path

Source | Projects | Libraries | Order and Export | Module Dependencies

JARs and class folders on the build path:

- Modulepath
  - JRE System Library [JavaSE-15]
- Classpath

Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...

Apply

Apply and Close | Cancel

Properties for Log4jDemo1

type filter text

- Resource
- Builders
- Coverage
- Java Build Path
- Java Code Style
- Java Compiler
- Javadoc Location
- Java Editor
- Namespaces
- Project Facets
- Project Natures
- Project References
- Run/Debug Settings
- Server
- Task Repository
- Task Tags
- Validation
- WikiText

Java Build Path

Source | Projects | Libraries | Order and Export | Module Dependencies

JARs and class folders on the build path:

- Modulepath
  - JRE System Library [JavaSE-15]
- Classpath
  - log4j-api-2.19.0.jar - D:\VodaFone_Java\apache-log4j-2.19.0-bin
  - log4j-core-2.19.0.jar - D:\VodaFone_Java\apache-log4j-2.19.0-bin

Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...

Apply

Apply and Close | Cancel

- Log4jDemo1
  - src
  - JRE System Library [JavaSE-15]
  - Referenced Libraries
    - log4j-api-2.19.0.jar - D:\VodaFone_J
    - log4j-core-2.19.0.jar - D:\VodaFone_

FirstLoggingDemo.java

```java
package mypackage;
import org.apache.logging.log4j.*;
public class FirstLoggingDemo {

    private static Logger demologger =
                                LogManager.getLogger(FirstLoggingDemo.class.getName());
    public static void main(String[] args) {
        System.out.println("THIS WILL PRINT IN CONSOLE");
        demologger.fatal("This is FATAL Message");
        demologger.error("This is ERROR Message - Database Connection FAIL");
        demologger.warn("This is WARN Message");
        demologger.info("THIS IS INFO MESSAGE - Database Connection SUCCESS");
        demologger.debug("This is DEBUG Message");
        demologger.trace("This is TRACE Message");
    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> FirstLoggingDemo [Java Application] D:\eclipse_March21\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (18-Sep-2022, 11:18:25 pm – 11:18:26 pm)
THIS WILL PRINT IN CONSOLE
23:18:26.792 [main] FATAL mypackage.FirstLoggingDemo - This is FATAL Message
23:18:26.799 [main] ERROR mypackage.FirstLoggingDemo - This is ERROR Message - Database Connection FAIL
```

10. If no configuration file could be located the `DefaultConfiguration` will be used. This will cause logging output to go to the console.

https://logging.apache.org/log4j/2.x/manual/configuration.html

- https://logging.apache.org/log4j/2.x/manual/customloglevels.html

Standard log levels built-in to Log4J

Standard Level	intLevel
OFF	0
FATAL	100
ERROR	200
WARN	300
INFO	400
DEBUG	500
TRACE	600
ALL	`Integer.MAX_VALUE`

Log4j Configuration

https://logging.apache.org/log4j/2.x/manual/configuration.html

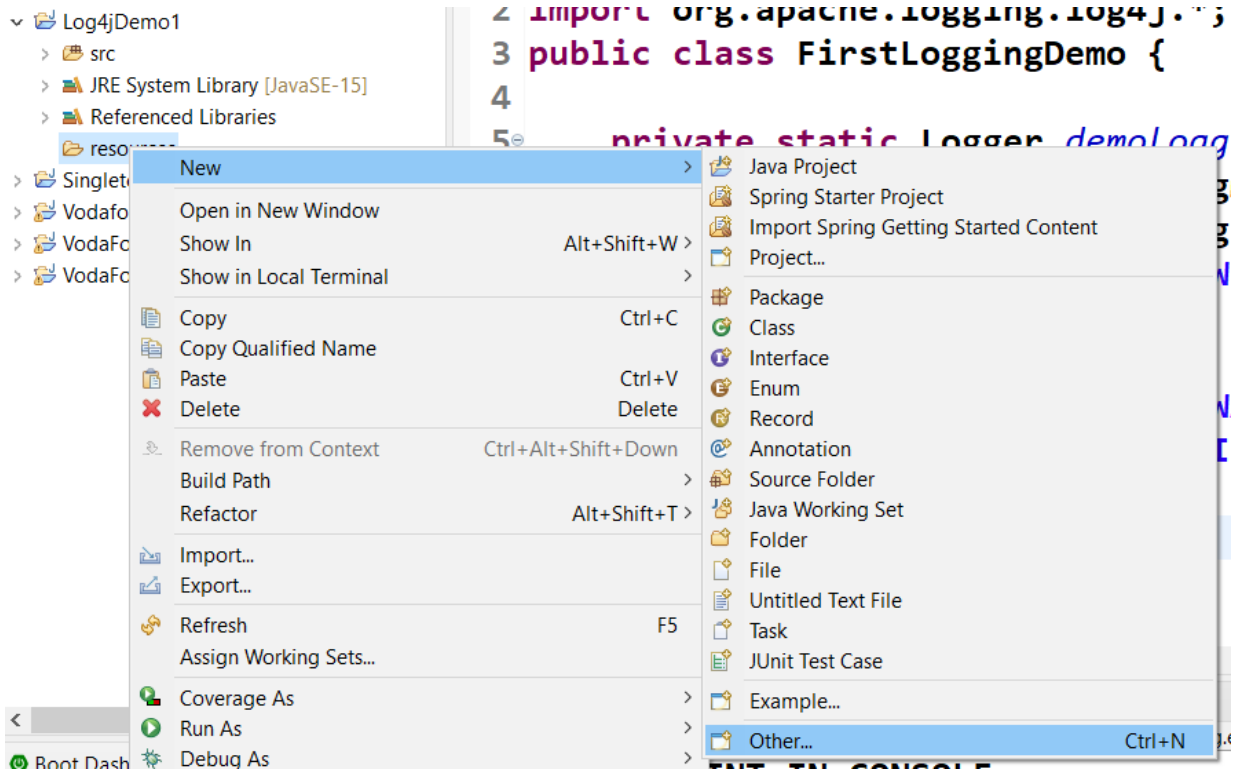
Configuration of Log4j 2 can be accomplished in 1 of 4 ways:

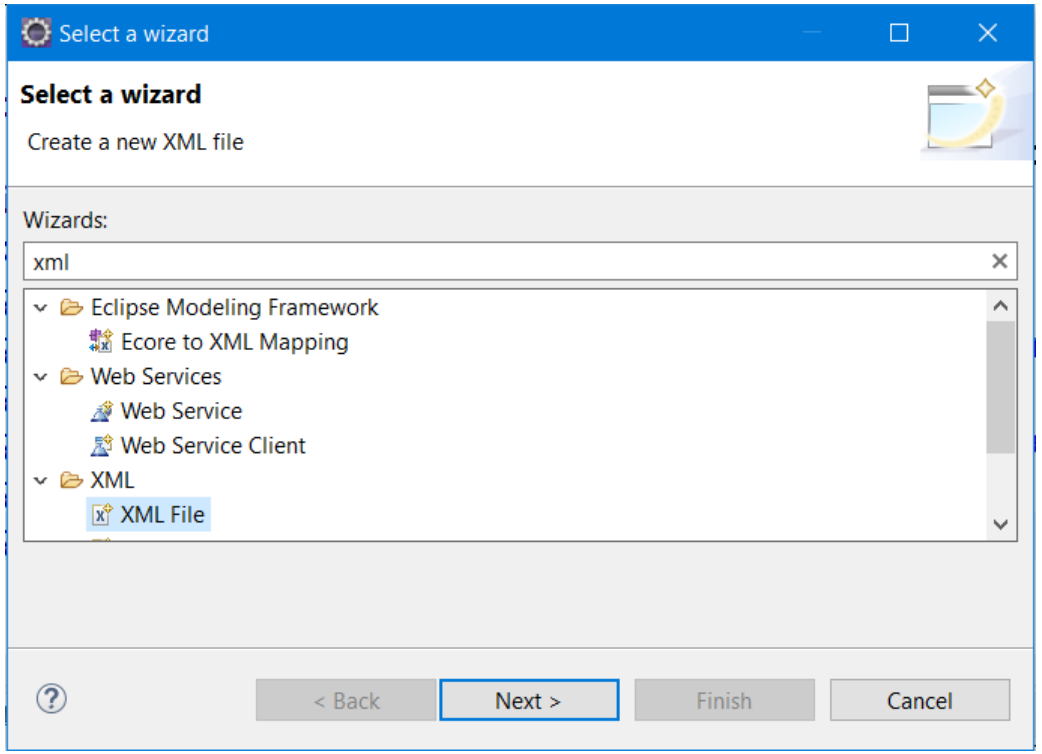
1. Through a configuration file written in XML, JSON, YAML, or properties format.
2. Programmatically, by creating a ConfigurationFactory and Configuration implementation.
3. Programmatically, by calling the APIs exposed in the Configuration interface to add components to the default configuration.
4. Programmatically, by calling methods on the internal Logger class.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

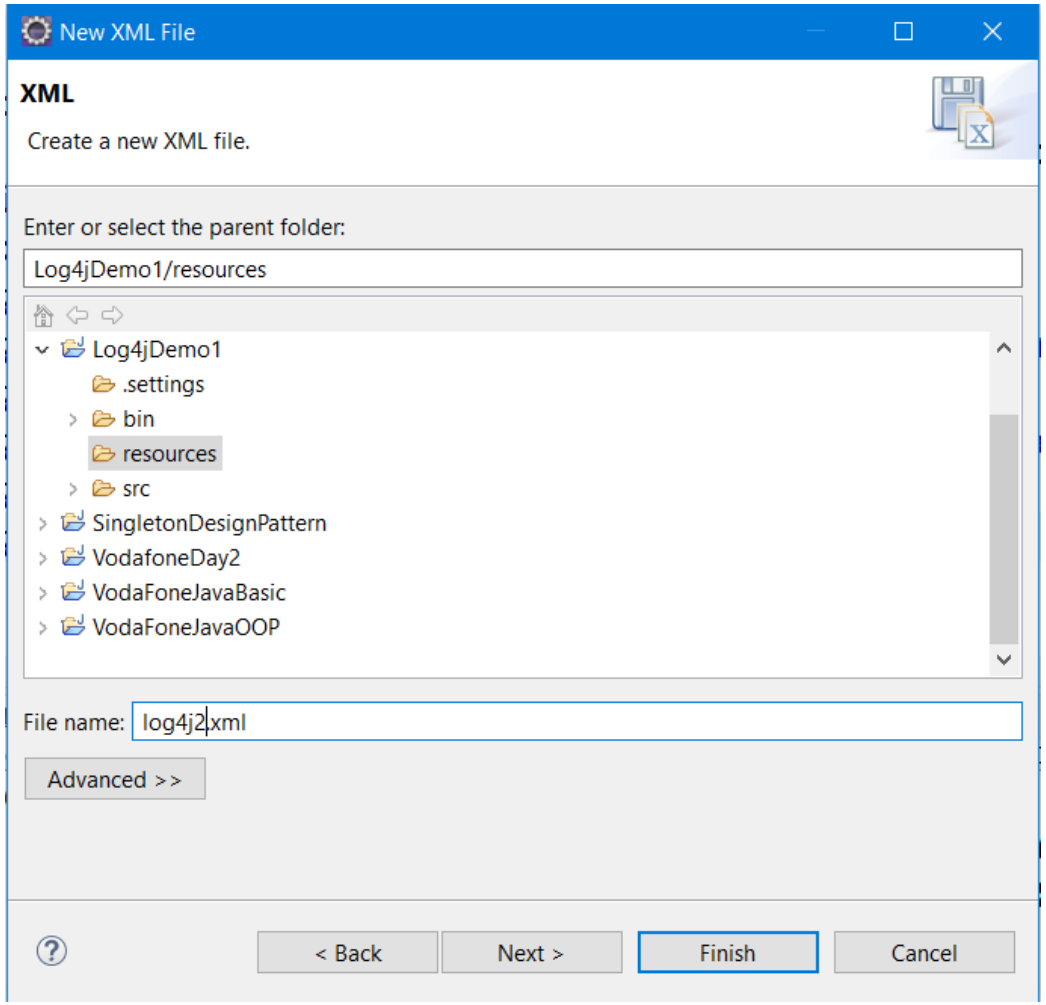
Log4j2 XML Configuration

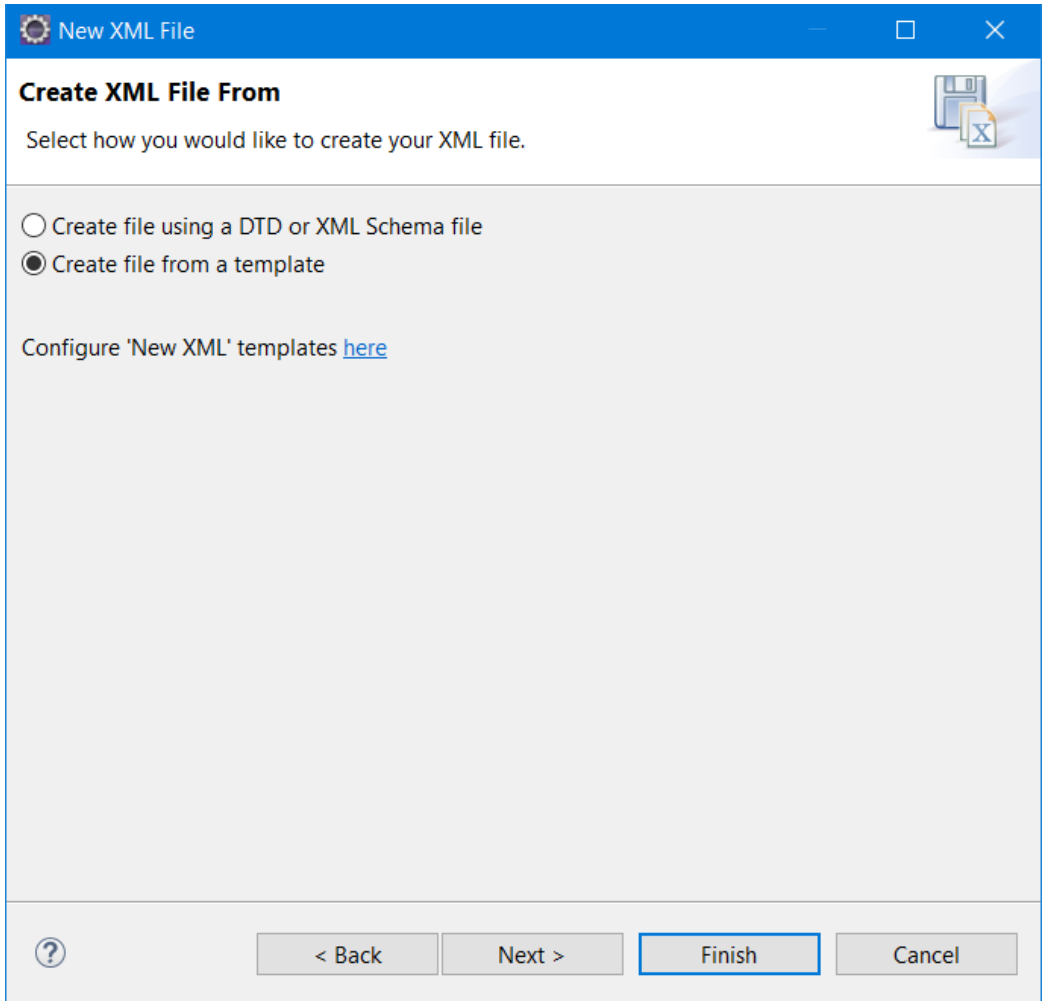
- Create new folder resources outside src folder
- Create XML file

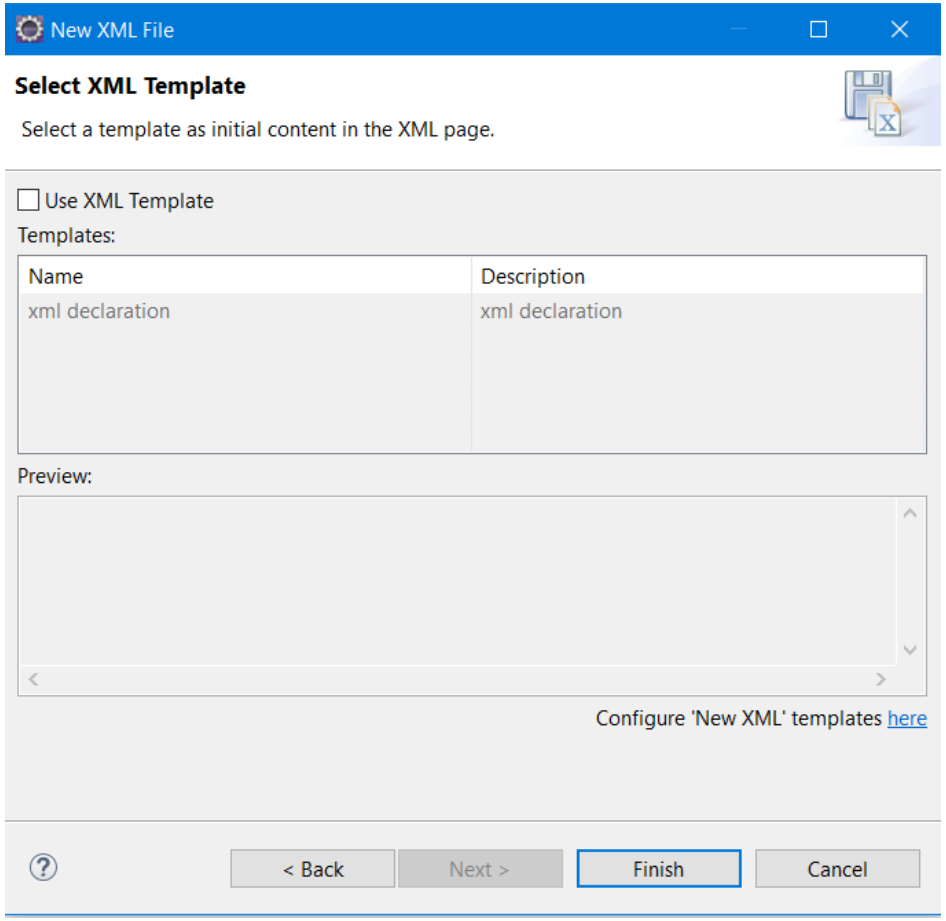




File Name Should be log4j2.xml





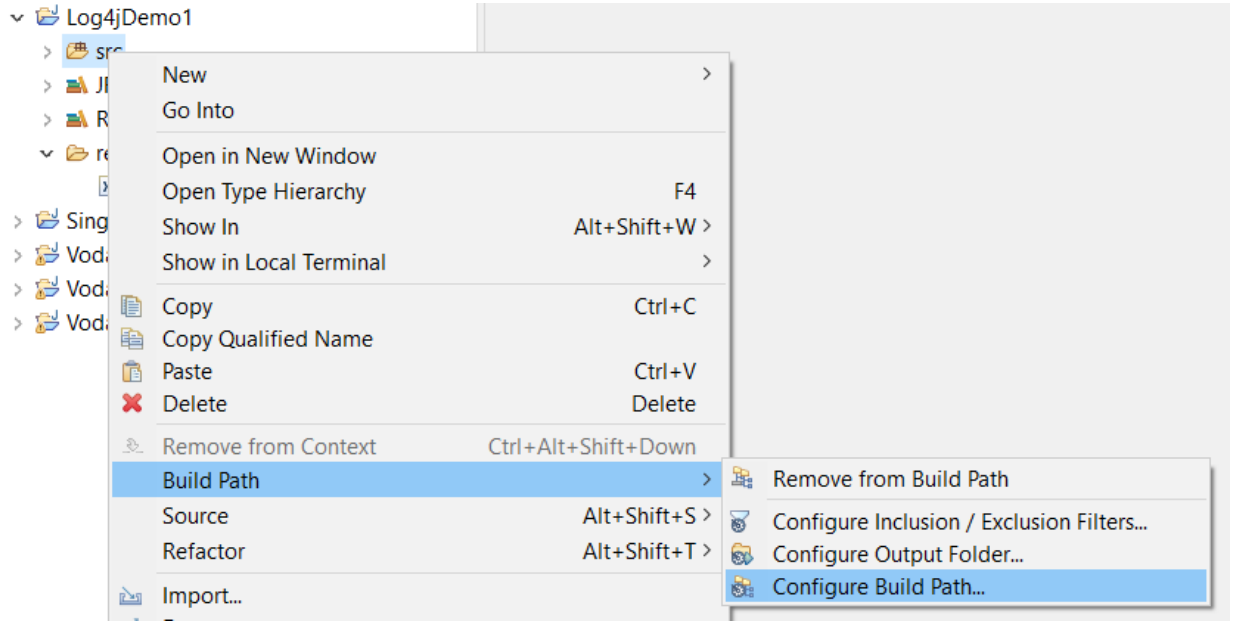


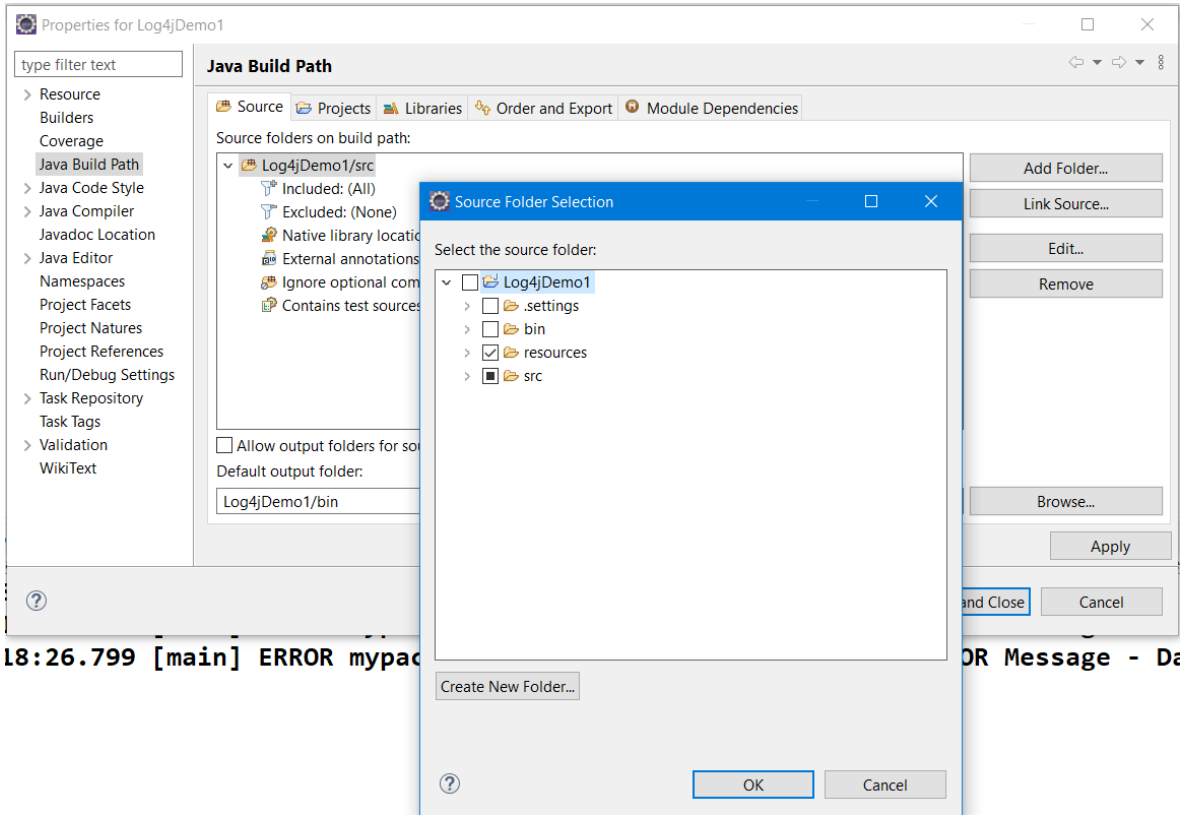
Change ROOT LEVEL TO TRACE So It will display all other below it level

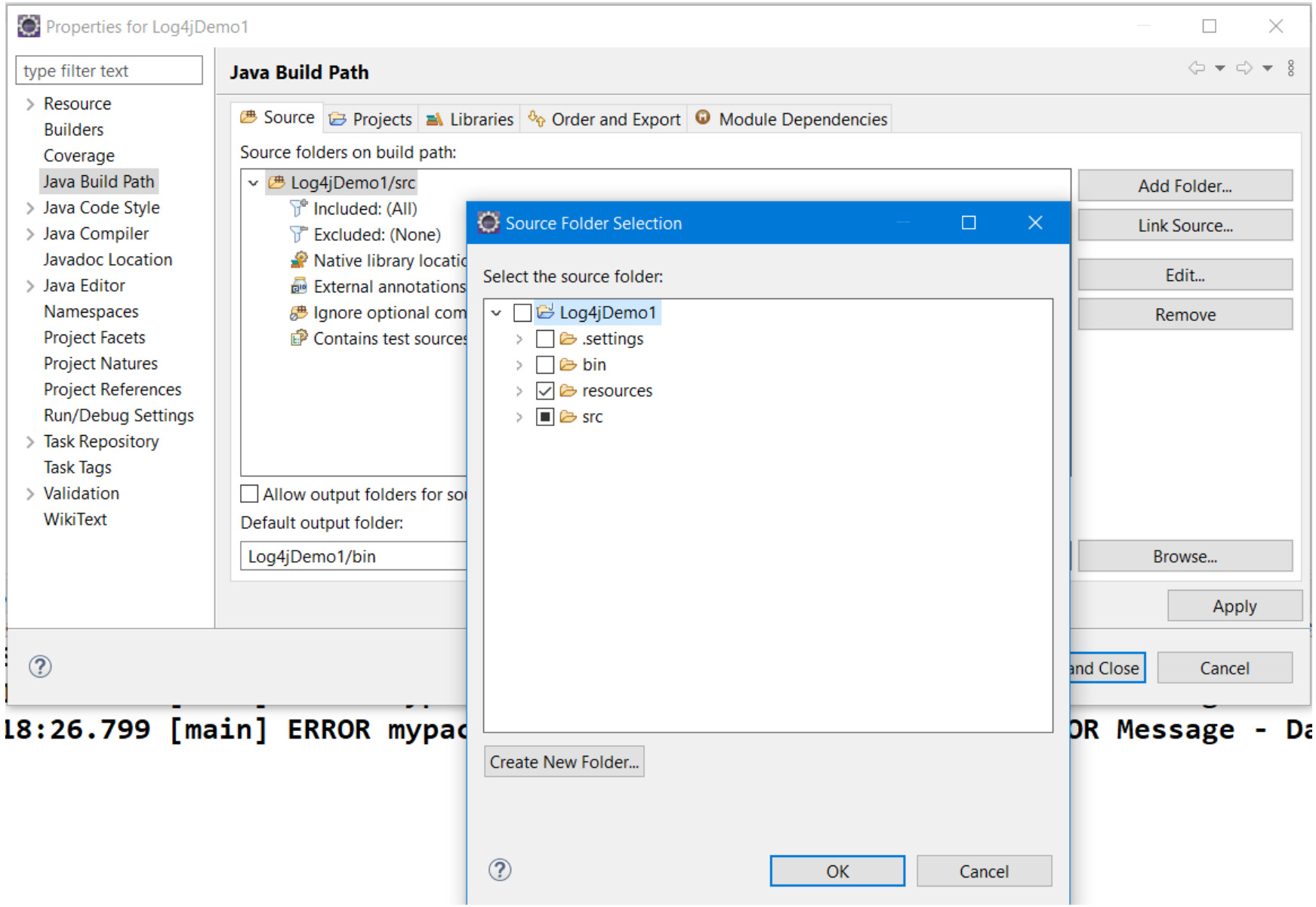
FirstLoggingDemo.java | log4j2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="trace">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

Add log4j2.XML FILE In build Path






Properties for Log4jDemo1
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Namespaces
Project Facets
Project Natures
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
Module Dependencies
Source folders on build path:
Log4jDemo1/src
Included: (All)
Excluded: (None)
Native library locatio
External annotations
Ignore optional com
Contains test sources
Allow output folders for so
Default output folder:
Log4jDemo1/bin
Add Folder...
Link Source...
Edit...
Remove
Browse...
Apply
and Close
Cancel
Source Folder Selection
Select the source folder:
Log4jDemo1
.settings
bin
resources
src
Create New Folder...
OK
Cancel
18:26.799 [main] ERROR mypac
OR Message - Da

```java
package mypackage;
import org.apache.logging.log4j.*;
public class FirstLoggingDemo {

    private static Logger demologger =
                                LogManager.getLogger(FirstLoggingDemo.class.getName());
    public static void main(String[] args) {
        System.out.println("THIS WILL PRINT IN CONSOLE");
        demologger.fatal("This is FATAL Message");
        demologger.error("This is ERROR Message - Database Connection FAIL");
        demologger.warn("This is WARN Message");
        demologger.info("THIS IS INFO MESSAGE - Database Connection SUCCESS");
        demologger.debug("This is DEBUG Message");
        demologger.trace("This is TRACE Message");
    }
```

Problems Javadoc Declaration Console

<terminated> FirstLoggingDemo [Java Application] D:\eclipse_March21\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (18-Sep-2022, 11:40:45 pm – 11:40:46 pm)

```
THIS WILL PRINT IN CONSOLE
23:40:46.104 [main] FATAL mypackage.FirstLoggingDemo - This is FATAL Message
23:40:46.106 [main] ERROR mypackage.FirstLoggingDemo - This is ERROR Message - Database Connection FAIL
23:40:46.106 [main] WARN  mypackage.FirstLoggingDemo - This is WARN Message
23:40:46.106 [main] INFO  mypackage.FirstLoggingDemo - THIS IS INFO MESSAGE - Database Connection SUCCESS
23:40:46.106 [main] DEBUG mypackage.FirstLoggingDemo - This is DEBUG Message
23:40:46.106 [main] TRACE mypackage.FirstLoggingDemo - This is TRACE Message
```

Change Log Level for Java Packages

Package Explorer

- FactoryDesignPattern
- JavaJarExample
- Log4jDemo1
  - src
    - mypack2
      - SecondLoggingDemo.java
    - mypackage
      - FirstLoggingDemo.java
  - resources
  - JRE System Library [JavaSE-15]
  - Referenced Libraries
- SingletonDesignPattern
- VodafoneDay2
- VodaFoneJavaBasic
- VodaFoneJavaOOP

SecondLoggingDemo.java

```java
package mypack2;
import org.apache.logging.log4j.*;
public class SecondLoggingDemo {

    private static Logger demologger =
                                LogManager.getLogger(SecondLoggingDemo.class.getName());
    public static void main(String[] args) {
        System.out.println("****************************");
        demologger.fatal("This is FATAL Message");
        demologger.error("This is ERROR Message - Database Connection FAIL");
        demologger.warn("This is WARN Message");
        demologger.info("THIS IS INFO MESSAGE - Database Connection SUCCESS");
        demologger.debug("This is DEBUG Message");
        demologger.trace("This is TRACE Message");
        System.out.println("****************************");
        demologger.fatal("This is FATAL Message");
        demologger.error("This is ERROR Message - Database Connection FAIL");
```

Change Log Level for Java Packages

- https://logging.apache.org/log4j/2.x/manual/configuration.html

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="com.foo.Bar" level="trace" additivity="false">
      <AppenderRef ref="Console"/>
    </Logger>
    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

Change Log Level for Java Packages

additivity="false" otherwise message will repeat twice

SecondLoggingDemo.java | FirstLoggingDemo.java | log4j2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
      <Logger name="mypack2" level="info" additivity="false">
         <AppenderRef ref="Console"/>
    </Logger>

    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

Change Log Level for Java Packages

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
     <Logger name="mypack2" level="info" additivity="false">
        <AppenderRef ref="Console"/>
    </Logger>

    <Root level="error">
```

Design | Source

Problems | Javadoc | Declaration | Console

<terminated> SecondLoggingDemo [Java Application] D:\eclipse_March21\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (19-Sep-2022, 12:15:42 am – 12:15:43 am)

```
****************************
00:15:43.034 [main] FATAL mypack2.SecondLoggingDemo - This is FATAL Message
00:15:43.036 [main] ERROR mypack2.SecondLoggingDemo - This is ERROR Message - Database Connection FAIL
00:15:43.036 [main] WARN  mypack2.SecondLoggingDemo - This is WARN Message
00:15:43.036 [main] INFO  mypack2.SecondLoggingDemo - THIS IS INFO MESSAGE - Database Connection SUCCESS
****************************
00:15:43.036 [main] FATAL mypack2.SecondLoggingDemo - This is FATAL Message
00:15:43.036 [main] ERROR mypack2.SecondLoggingDemo - This is ERROR Message - Database Connection FAIL
00:15:43.036 [main] WARN  mypack2.SecondLoggingDemo - This is WARN Message
00:15:43.036 [main] INFO  mypack2.SecondLoggingDemo - THIS IS INFO MESSAGE - Database Connection SUCCESS
```

Create Log File

- https://logging.apache.org/log4j/2.x/manual/configuration.html

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="debug" name="RoutingTest" packages="org.apache.logging.log4j.test">
  <Properties>
    <Property name="filename">target/rolling1/rollingtest-$${sd:type}.log</Property>
  </Properties>
  <ThresholdFilter level="debug"/>

  <Appenders>
    <Console name="STDOUT">
      <PatternLayout pattern="%m%n"/>
      <ThresholdFilter level="debug"/>
    </Console>
    <Routing name="Routing">
      <Routes pattern="$${sd:type}">
        <Route>
          <RollingFile name="Rolling-${sd:type}" fileName="${filename}"
                       filePattern="target/rolling1/test1-${sd:type}.%i.log.gz">
            <PatternLayout>
              <pattern>%d %p %c{1.} [%t] %m%n</pattern>
            </PatternLayout>
            <SizeBasedTriggeringPolicy size="500" />
          </RollingFile>
        </Route>
        <Route ref="STDOUT" key="Audit"/>
      </Routes>
    </Routing>
  </Appenders>
```

- PROGRAM

Best Practices in Java Coding

Use Proper Naming Conventions

- Self-explanatory
- Meaningful distinctions
- Pronounceable

✓ Class and interface names should be nouns, starting with an uppercase letter.
For example: `Student`, `Car`, `Rectangle`, `Painter`, etc.

✓ Variable names should be nouns, starting with a lowercase letter.
For example: `number`, `counter`, `birthday`, `gender`, etc.

✓ Method names should be verbs, starting with a lowercase letter.
For example: `run`, `start`, `stop`, `execute`, etc.

✓ Constant names should have all UPPERCASE letters and words are separated by underscores.
For example: `MAX_SIZE`, `MIN_WIDTH`, `MIN_HEIGHT`, etc.

✓ Using `camelCase` notation for names.
For example: `StudentManager`, `CarController`, `numberOfStudents` etc.

Class Members must be accessed privately

```
public class Teacher {
    private String name;
    private String subject;

    public void setName(String name) {
        this.name = name;
    }
    public void setSubject(String subject)
        this.subject = subject;
    }
}
```

Ordering Class Members by Scopes

```java
public class StudentManager {
    protected List<Student> listStudents;
    public int numberOfStudents;
    private String errorMessage;
    float rowHeight;
    float columnWidth;
    protected String[] columnNames;
    private int numberOfRows;
    private int numberOfColumns;
    public String title;

}
```

```java
public class StudentManager {
    private String errorMessage;
    private int numberOfColumns;
    private int numberOfRows;


    float columnWidth;
    float rowHeight;

    protected String[] columnNames;
    protected List<Student> listStudents;

    public int numberOfStudents;
    public String title;

}
```

Use Underscores in lengthy Numeric Literals

```java
int maxUploadSize = 20971520;
long accountBalance = 1000000000000L;
float pi = 3.141592653589F;
```

```java
int maxUploadSize = 20_971_520;
long accountBalance = 1_000_000_000_000L;
float pi = 3.141_592_653_589F;
```

Using Enums or Constant Class instead of

```java
public enum Color {
    BLACK, WHITE, RED
}
```

```java
public class AppConstants {
    public static final String TITLE = "Application Name";

    public static final int VERSION_MAJOR = 2;
    public static final int VERSION_MINOR = 4;

    public static final int THREAD_POOL_SIZE = 10;

    public static final int MAX_DB_CONNECTIONS = 50;

    public static final String ERROR_DIALOG_TITLE = "Error";
    public static final String WARNING_DIALOG_TITLE = "Warning";
    public static final String INFO_DIALOG_TITLE = "Information";
}
```

- **Use StringBuilder or StringBuffer for String Concatenation**

```
String sql = "Insert Into Users (name, email, pass, address)";
sql += " values ('" + user.getName();
sql += "', '" + user.getEmail();
sql += "', '" + user.getPass();
sql += "', '" + user.getAddress();
sql += "')";
```

```
StringBuilder sbSql
    = new StringBuilder("Insert Into Users (name, email, pass, address)");

sbSql.append(" values ('").append(user.getName());
sbSql.append("', '").append(user.getEmail());
sbSql.append("', '").append(user.getPass());
sbSql.append("', '").append(user.getAddress());
sbSql.append("')");

String sql = sbSql.toString();
```

- **Proper handling of Null Pointer Exceptions**

Using enhanced for loops instead of for loops with counter

```
String[] names = {"Sam", "Mike", "John"};
for (int i = 0; i < names.length; i++) {
    method1(names[i]);
}
```

```
For (String Name1 : names) {
    Method1(Name1);
}
```

- **Return Empty Collections instead of returning Null elements**

- **Proper Commenting**

- **Use of single quotes and double quotes**

```
public class classA {
    public static void main(String args[]) {
    System.out.print("A" + "B");
    System.out.print('C' + 'D');
    }
}
```

AB135

• Using Interface References to Collections

ArrayList<Integer> alist = new ArrayList<Integer>();

List<Integer> alist = new ArrayList<Integer>();

• Avoid Redundant Initializations

it is not encouraged to initialize member variables with the values: like 0, false and null. These values are already the default initialization values of member variables in Java

Redundant Initializations

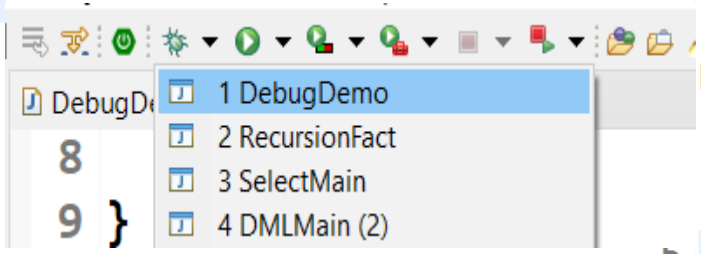
```
public class Person {
    private String name;
    private int age;
    private boolean;

    public Person() {
        String name = null;
        int age = 0;
        boolean isGenius = false;
    }
}
```

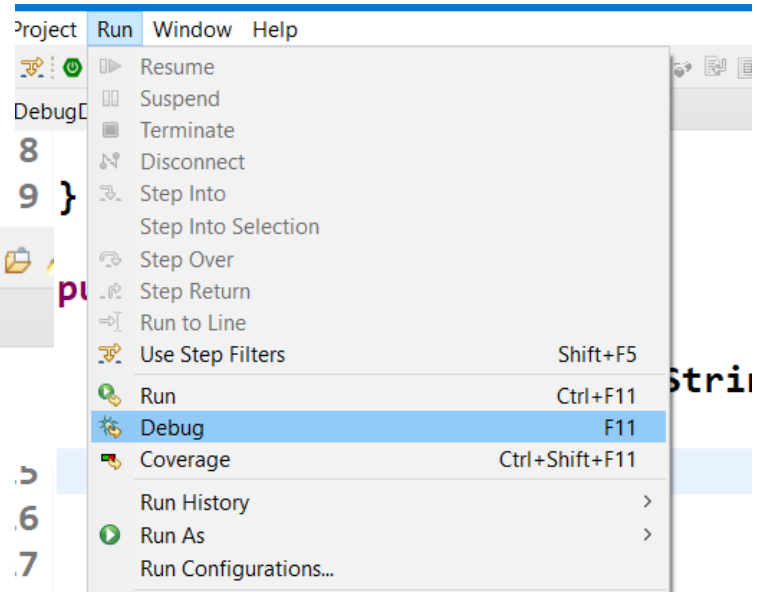
Debugging

- CREATE DEBUG POINT

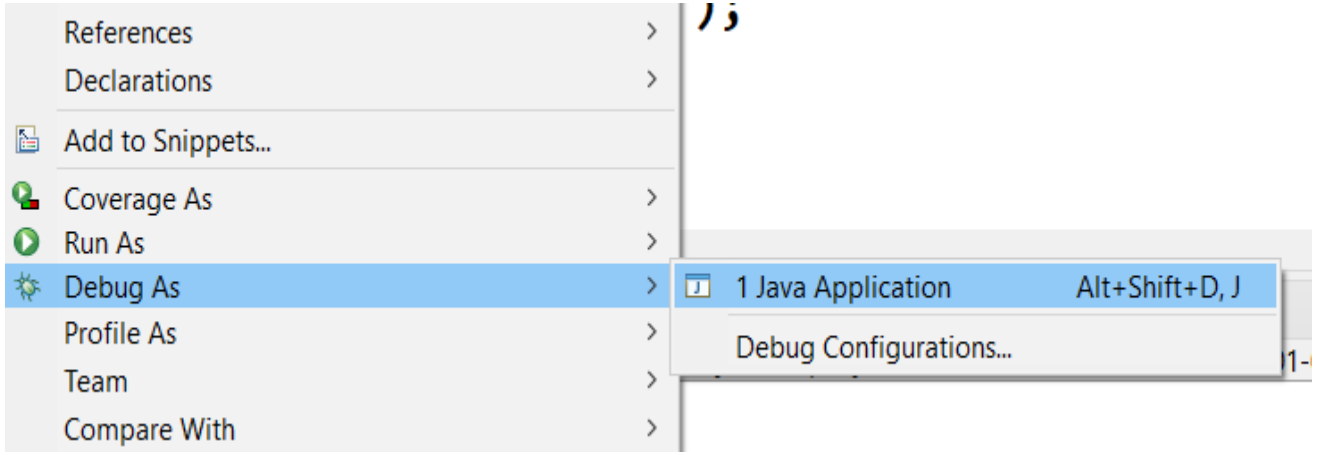
```
10
11 public class DebugDemo {
12
13     public static void main(String[] args) {
14         int a = 25;
15         int b = 35;
16
17         Add ob = new Add();
           d(a, b);
           intln(k);

           t.println("HI");
```

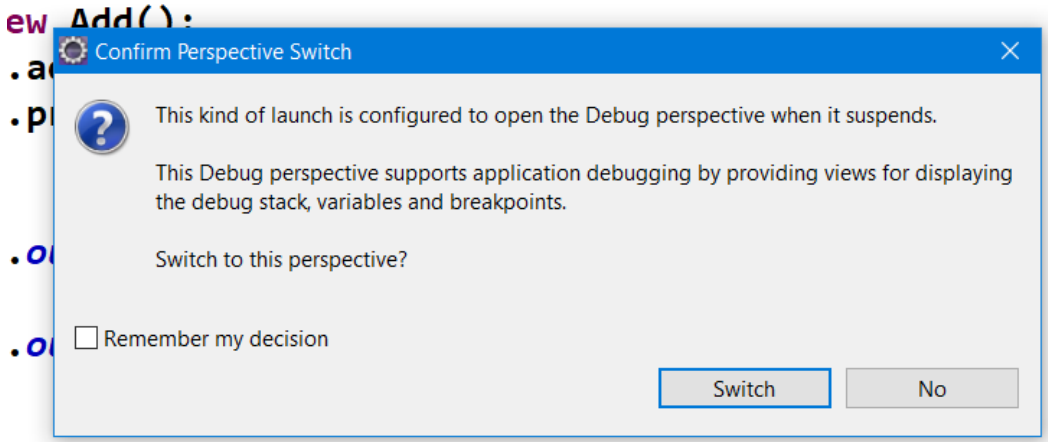
Toggle Breakpoint	Ctrl+Shift+B
Disable Breakpoint	Shift+Double Click
Toggle Tracepoint	
Run to Line	Ctrl+Alt+Click
Go to Annotation	Ctrl+1
Validate	
Revert Block	
Delete Added Line	
Add Bookmark...	
Add Task...	
Show Quick Diff	Ctrl+Shift+Q
Show Line Numbers	
Folding	›
Preferences...	
Breakpoint Properties...	Ctrl+Double Click

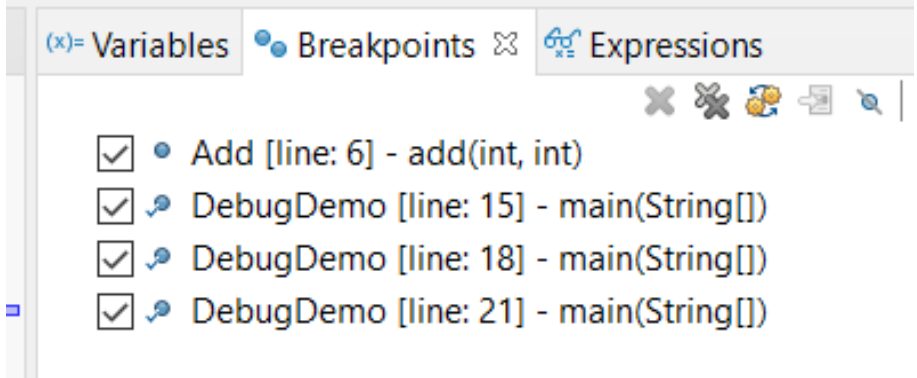


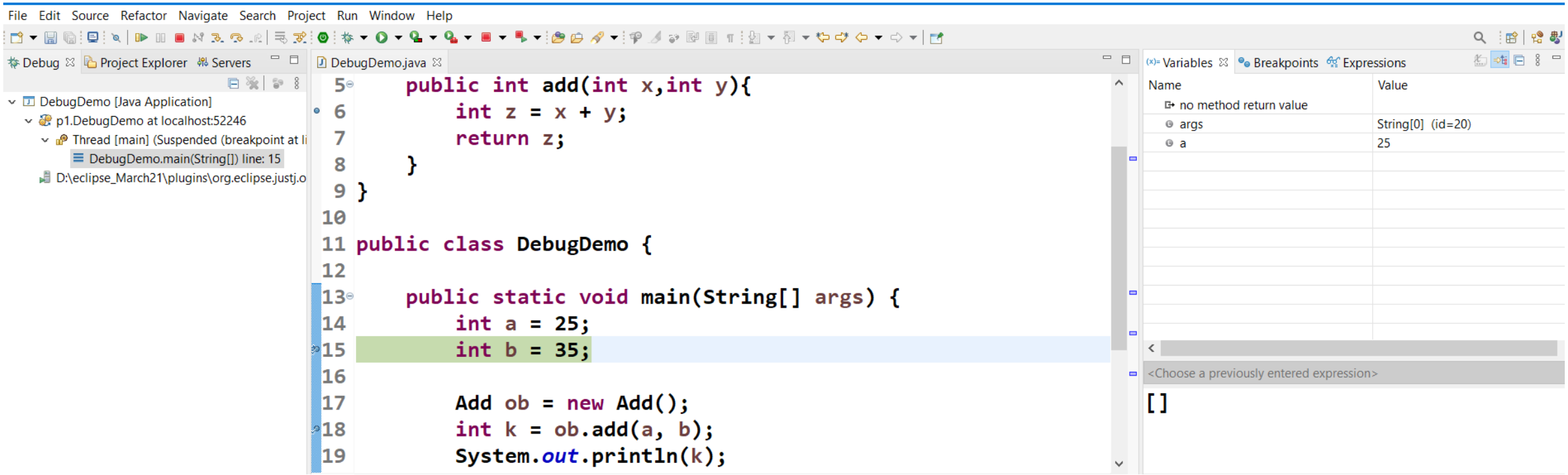


RIGHT CLICK AND SELECT DEBUG




Confirm Perspective Switch
This kind of launch is configured to open the Debug perspective when it suspends.
This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.
Switch to this perspective?
Remember my decision
Switch
No


Variables
Breakpoints
Expressions
Add [line: 6] - add(int, int)
DebugDemo [line: 15] - main(String[])
DebugDemo [line: 18] - main(String[])
DebugDemo [line: 21] - main(String[])


File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
Servers
DebugDemo [Java Application]
p1.DebugDemo at localhost:52246
Thread [main] (Suspended (breakpoint at li
DebugDemo.main(String[]) line: 15
D:\eclipse_March21\plugins\org.eclipse.justj.o
DebugDemo.java
public int add(int x,int y){
int z = x + y;
return z;
}
}
public class DebugDemo {
public static void main(String[] args) {
int a = 25;
int b = 35;
Add ob = new Add();
int k = ob.add(a, b);
System.out.println(k);
Variables
Breakpoints
Expressions
Name
Value
no method return value
args
String[0] (id=20)
a
25
<Choose a previously entered expression>
[]

Key	Description
F5	Executes the currently selected line and goes to the next line in your program. If the selected line is a method call the debugger steps into the associated code.
F6	F6 steps over the call, i.e. it executes a method without stepping into it in the debugger.
F7	F7 steps out to the caller of the currently executed method. This finishes the execution of the current method and returns to the caller of this method.
F8	F8 tells the Eclipse debugger to resume the execution of the program code until is reaches the next breakpoint or watchpoint.

The following picture displays the buttons and their related keyboard shortcuts.

Design Pattern

- Design Patterns are already defined and provides industry standard approach to solve a recurring problem, so it saves time if we sensibly use the design pattern. There are many java design patterns that we can use in our java based projects.

- Using design patterns promotes reusability that leads to more robust and highly maintainable code. It helps in reducing total cost of ownership (TCO) of the software product.

- Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily.

Design Patterns In JAVA

23 GoF(Gang Of Four) Design Patterns

Creational

1. Singleton
2. Factory
3. Abstract Factory
4. Builder
5. Prototype

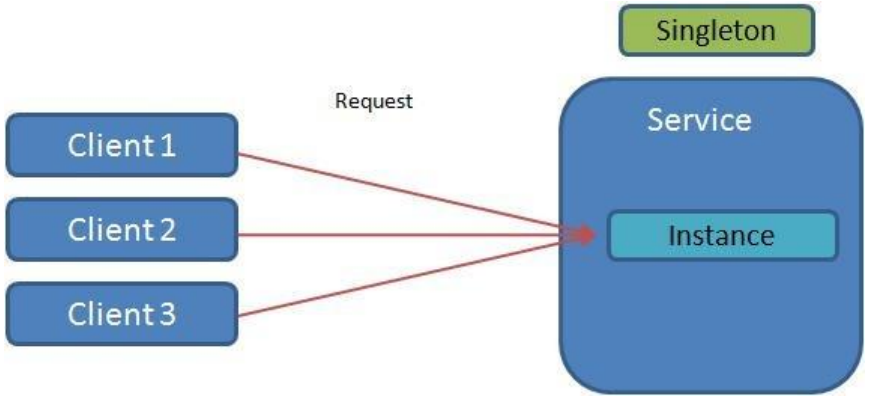
Structural

6. Adapter
7. Composite
8. Proxy
9. Flyweight
10. Facade
11. Bridge
12. Decorator

Behavioral

13. Template Method
14. Mediator
15. Observer
16. Strategy
17. Command
18. State
19. Visitor
20. Iterator
21. Interpreter
22. Memento
23. Chain Of Responsibility

Singleton Pattern



- ✓ Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
- ✓ The singleton class must provide a global access point to get the instance of the class.
- ✓ Singleton pattern is used for logging, drivers objects, caching and thread pool.
- ✓ Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.
- ✓ Singleton design pattern is used in core java classes also, for example java.lang.Runtime, java.awt.Desktop.

Factory Design Pattern / Factory Method Design Pattern

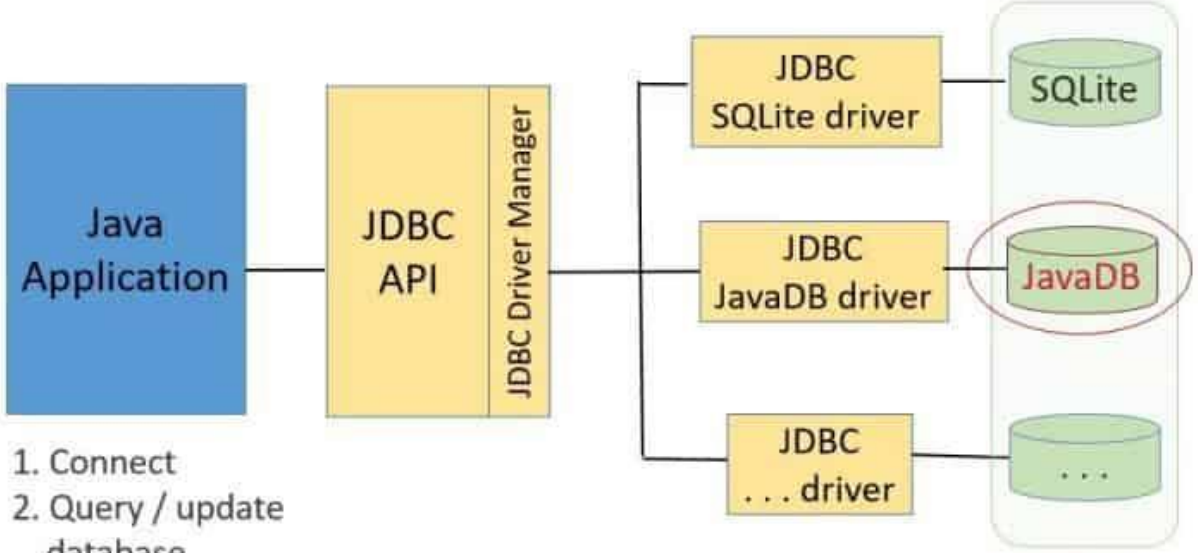
- The factory design pattern is used when we have a superclass with multiple sub-classes and based on input, we need to return one of the sub-class.

- This pattern takes out the responsibility of the instantiation of a class from the client program to the factory class.

- Program

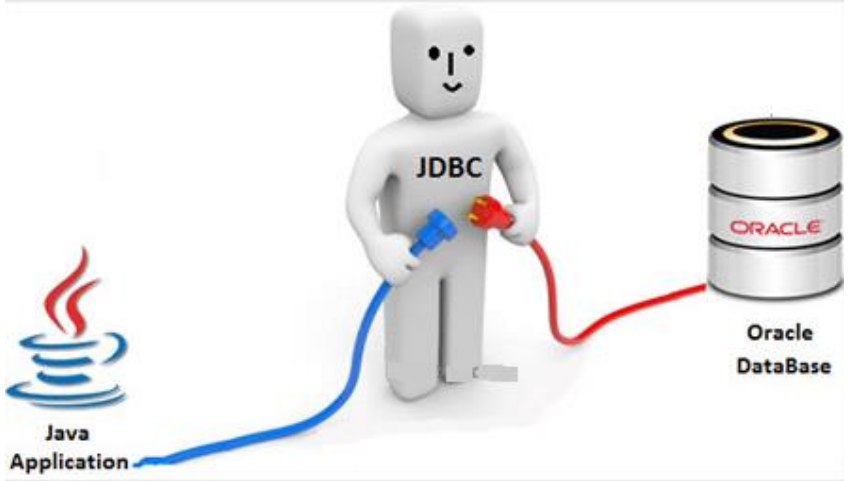
JDBC

JDBC - Java Database Connectivity

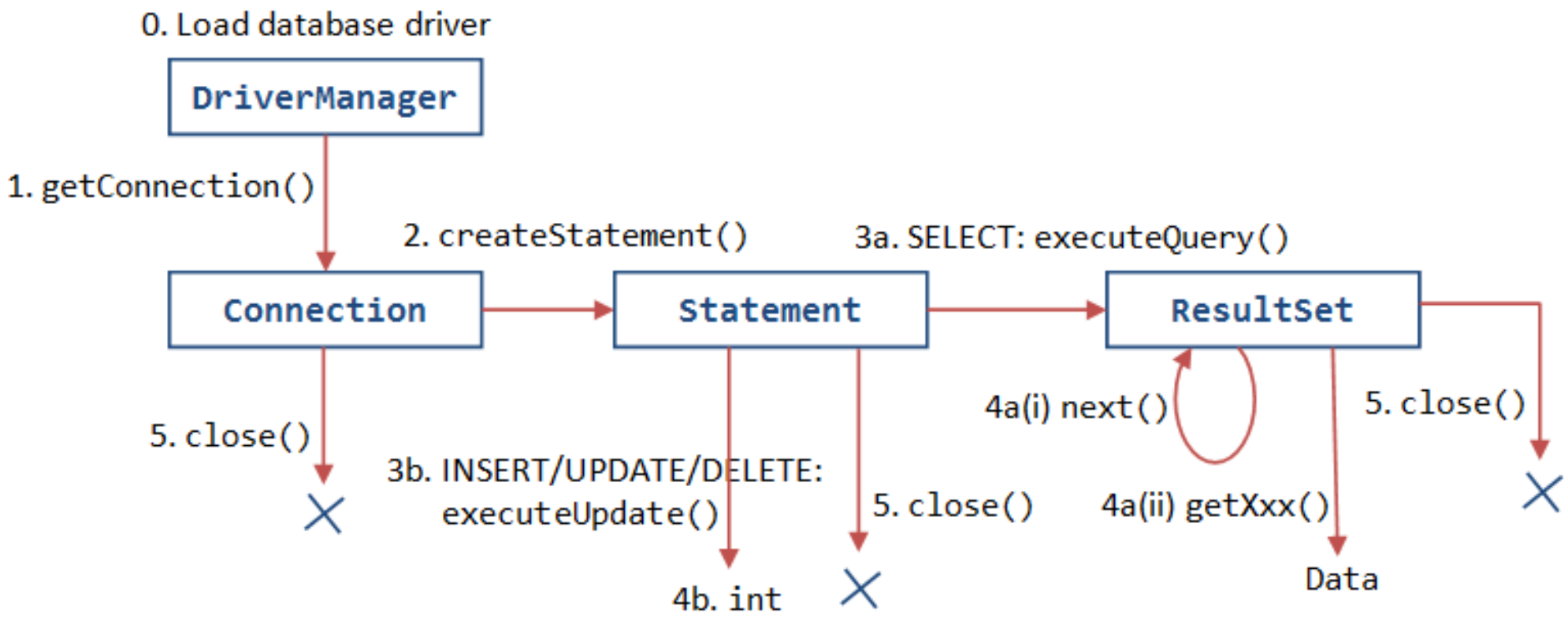


1. Connect
2. Query / update database
3. Receive results

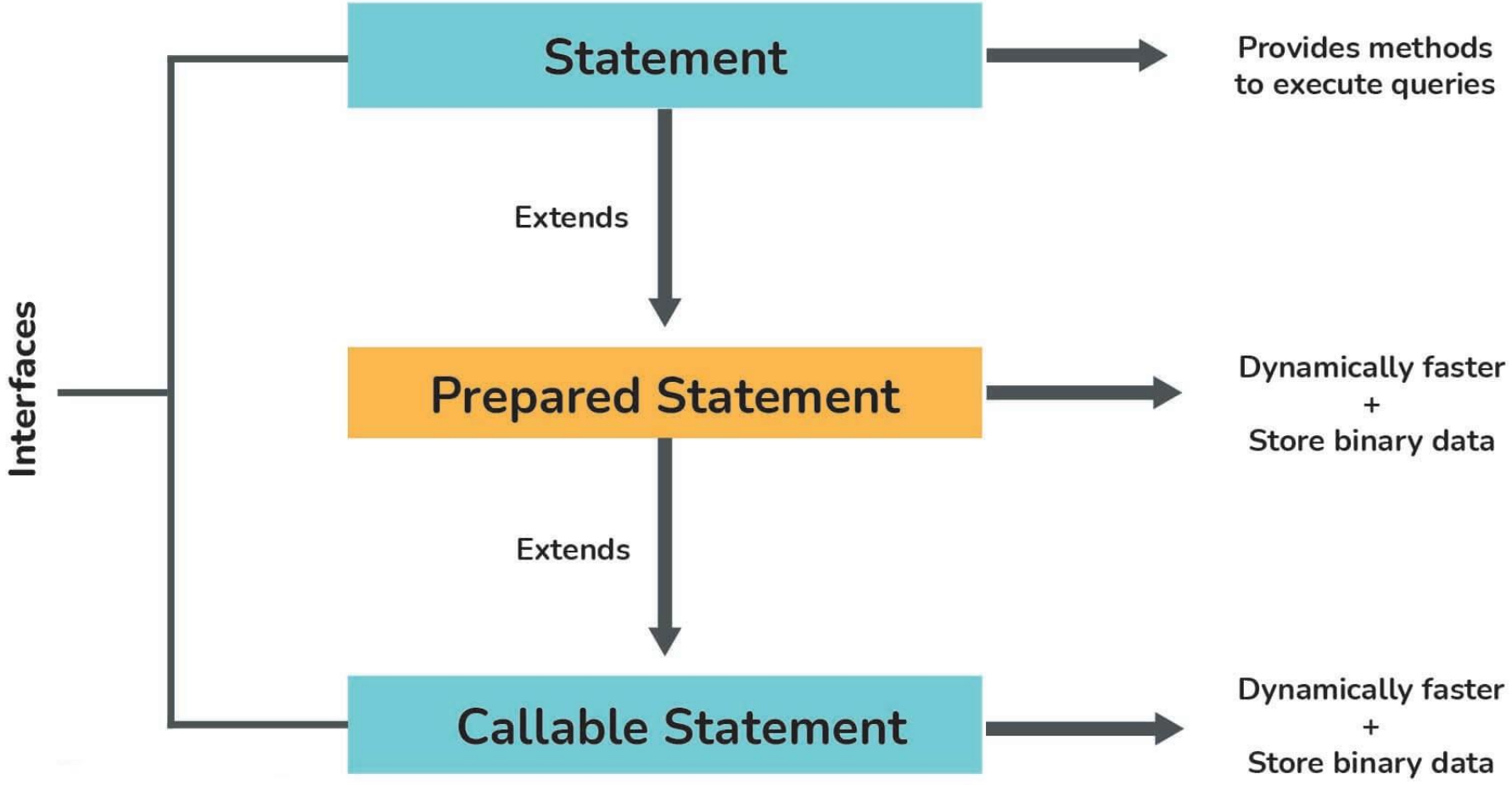
DBMS
Database Management System

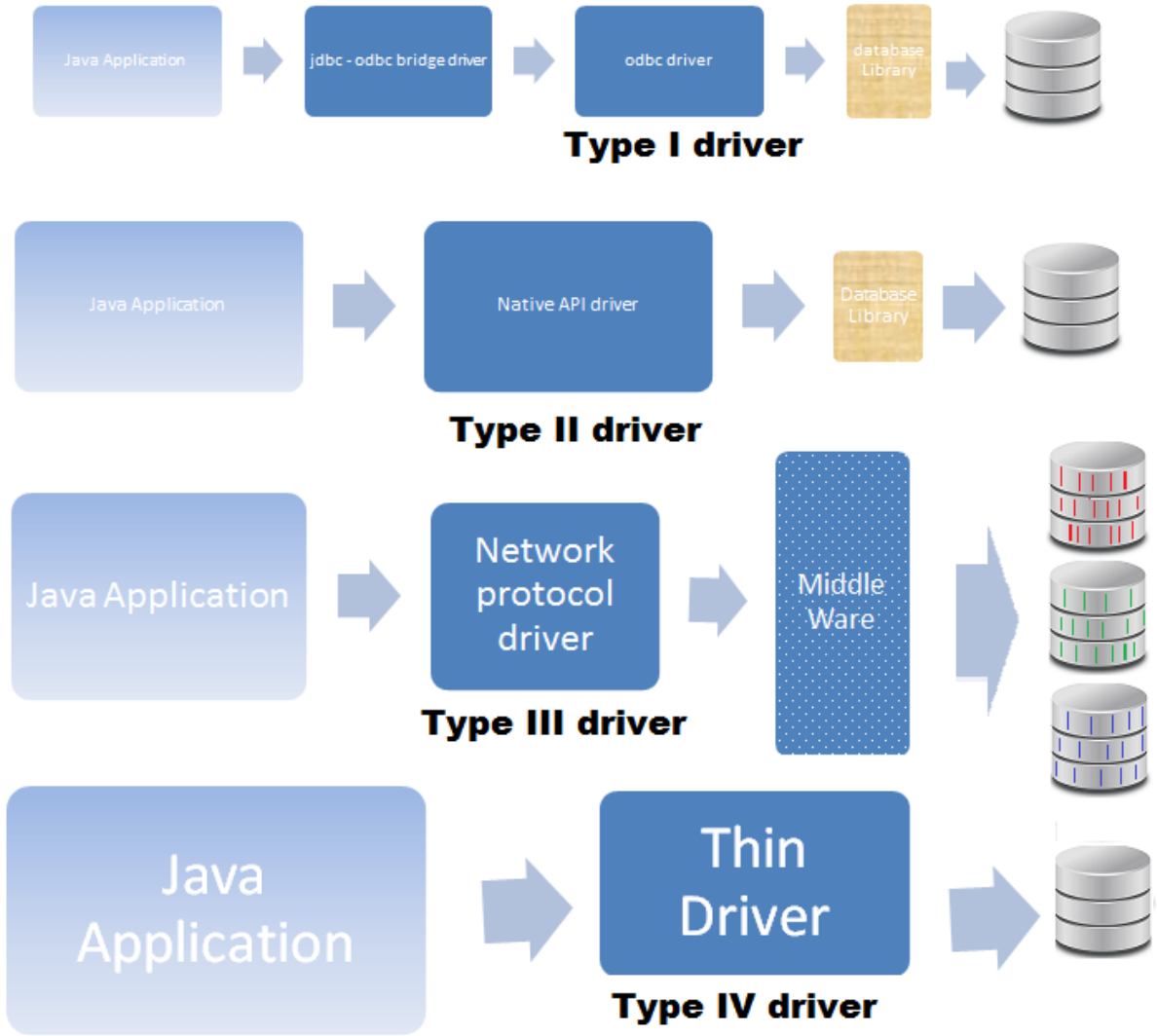


1. Import Packages
2. Load Driver
3. Establish Connection
4. Create and execute statement
5. Retrieve Results
6. Close connection


0. Load database driver
DriverManager
1. getConnection()
2. createStatement()
3a. SELECT: executeQuery()
Connection
Statement
ResultSet
5. close()
3b. INSERT/UPDATE/DELETE:
executeUpdate()
5. close()
4a(i) next()
4a(ii) getXxx()
5. close()
4b. int
Data

Types Of Statements In JDBC





Types of JDBC drivers

Property	Type-1	Type-2	Type-3	Type-4
Conversion	From JDBC calls to ODBC calls	From JDBC calls to native library calls	From JDBC calls to middle-wear specific calls	From JDBC calls to Data Base specific calls
Implemented-in	Only java	Java + Native language	Only java	Only java
Architecture	Follow 2-tier architecture	Follow 2-tier architecture	Follow 3-tier architecture	Follow 2-tier architecture
Platform-independent	NO	NO	YES	YES
Data Base independent	YES	NO	YES	NO
Thin or Thick	Thick	Thick	Thick	Thin