BLG 411E – Software Engineering

Recitation Session 4

Log4j, Hibernate

Beyza Eken, Bilge S. Akkoca Gazioğlu, Müge Erel Özçevik

21.11.2017

Outline

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Introduction

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Introduction

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Introduction

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Introduction

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Introduction

Definition

Recording the events which happen during a software execution.

- Logging allows to:
  - audit important items,
  - analyze application use,
  - trace errors.
- Performance is the most important consideration.

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Pros & Cons

- Logging is useful when it is well written:
  - Quick debugging
  - Easy maintenance
  - Structured storage of an application's runtime information
- If not:
  - If too verbose, it can cause scrolling blindness
  - it can slow down an application

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Logging Levels

OFF	No logging
FATAL	Only errors that cause the application to abort
ERROR	Critical errors that prevent the use case from continuing
WARN	Other errors and exceptions where processes may still continue, i.e. “Current data unavailable, using cached values”
INFO	Life cycle events, completion of use cases, i.e. “[Who] booked ticket from [Where] to [Where]”
DEBUG	All other (sub) events that helps developers in application debugging
ALL	All method invocations with parameter and return values

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Application Logging
Tips

- Apply logging levels appropriately
- Avoid side effects (no impact on the application's behavior)
- Log for easy reading and easy parsing
- Include both data and description, no magic numbers or characters (i.e. &&&)
- Use a logging pattern (i.e. %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)
- Maximize logging for external systems
- Maximize logging when in trouble, not otherwise
- Avoid excessive string concatenation, use parametrized logging methods
- Do not log sensitive information

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Log4j
Introduction

- Three main components
  - **Loggers**
    - Responsible for capturing logging information
  - **Appenders**
    - Responsible for publishing logging information to various preferred destinations (database, file, console, etc.)
  - **Layouts**
    - Responsible for formatting logging information in different styles
    - %d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n)

Definition

A Java-based logging utility which provides performance improvements and advanced filtering.

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloWorld {
    private static final Logger l
                    = LogManager.getLogger("HelloWorld");
    public static void main(String[] args) {
        l.info("Hello, World!");
        l.error("Mission failed!");
    }
}
```

Definition

A Java-based logging utility which provides performance improvements and advanced filtering.

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloWorld {
    private static final Logger l
                    = LogManager.getLogger("HelloWorld");
    public static void main(String[] args) {
        l.info("Hello, World!");
        l.error("Mission failed!");
    }
}
```

Definition

A Java-based logging utility which provides performance improvements and advanced filtering.

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloWorld {
    private static final Logger l
                    = LogManager.getLogger("HelloWorld");
    public static void main(String[] args) {
        l.info("Hello, World!");
        l.error("Mission failed!");
    }
}
```

Log4j
Sample


```
2
3  import org.apache.log4j.Level;
4  import org.apache.log4j.Logger;
5  import org.apache.log4j.PropertyConfigurator;
6
7  public class Log4jHelloWorld {
8
9      static final Logger logger = Logger.getRootLogger();
10
11
12     public static void main(String[] args) {
13
14         PropertyConfigurator.configure("log4j.properties");
15
16         logger.setLevel(Level.WARN);
17
18         logger.debug("Sample debug message");
19         logger.info("Sample info message");
20         logger.warn("Sample warn message");
21         logger.error("Sample error message");
22         logger.fatal("Sample fatal message");
23     }
24
25 }
```

Problems @ Javadoc Declaration Console

```
<terminated> Log4jHelloWorld [Java Application] /Library/Java/JavaVirtualMachines/jdk1
2016-11-29 10:31:44 WARN  root:20 - Sample warn message
2016-11-29 10:31:44 ERROR root:21 - Sample error message
2016-11-29 10:31:44 FATAL root:22 - Sample fatal message
```

Object-Relational Mapping
Definitions

- A visualization [1]

[1] https://www.tutorialspoint.com/hibernate/hibernate_quick_guide.htm

Object-Relational Mapping
Definitions

Persistence Application's data to outlive the applications process, state of objects to live beyond the scope of the JVM so that the same state is available later

DMSs Although ODBMSs are emerging, RDBMSs are still the most popular ones.

Paradigm mismatch Object models (e.g. Java) and relational models (e.g. SQL) do not work very well together. See the next slide for the details.

ORM A programming technique for converting data between the two incompatible type systems

Object-Relational Mapping
Definitions

Persistence Application's data to outlive the applications process, state of objects to live beyond the scope of the JVM so that the same state is available later

DMSs Although ODBMSs are emerging, RDBMSs are still the most popular ones.

Paradigm mismatch Object models (e.g. Java) and relational models (e.g. SQL) do not work very well together. See the next slide for the details.

ORM A programming technique for converting data between the two incompatible type systems

Object-Relational Mapping
Definitions

Persistence Application's data to outlive the applications process, state of objects to live beyond the scope of the JVM so that the same state is available later

DMSs Although ODBMSs are emerging, RDBMSs are still the most popular ones.

Paradigm mismatch Object models (e.g. Java) and relational models (e.g. SQL) do not work very well together. See the next slide for the details.

ORM A programming technique for converting data between the two incompatible type systems

Object-Relational Mapping
Definitions

Persistence Application's data to outlive the applications process, state of objects to live beyond the scope of the JVM so that the same state is available later

DMSs Although ODBMSs are emerging, RDBMSs are still the most popular ones.

Paradigm mismatch Object models (e.g. Java) and relational models (e.g. SQL) do not work very well together. See the next slide for the details.

ORM A programming technique for converting data between the two incompatible type systems

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

Granularity	Object model usually has more classes than the number of corresponding tables in the database.
Subtypes	Inheritance in OOP does not exist in RDBMSs.
Identity	OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).
Association	References vs. foreign keys
Navigation	In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

  Granularity Object model usually has more classes than the number of corresponding tables in the database.

  Subtypes Inheritance in OOP does not exist in RDBMSs.

  Identity OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).

  Association References vs. foreign keys

  Navigation In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

  Granularity Object model usually has more classes than the number of corresponding tables in the database.

  Subtypes Inheritance in OOP does not exist in RDBMSs.

  Identity OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).

  Association References vs. foreign keys

  Navigation In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

Granularity	Object model usually has more classes than the number of corresponding tables in the database.
Subtypes	Inheritance in OOP does not exist in RDBMSs.
Identity	OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).
Association	References vs. foreign keys
Navigation	In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

Granularity	Object model usually has more classes than the number of corresponding tables in the database.
Subtypes	Inheritance in OOP does not exist in RDBMSs.
Identity	OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).
Association	References vs. foreign keys
Navigation	In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Object-Relational Mapping

Object-relational impedance mismatch

- RDBMSs represent data in a **tabular format** where object-oriented languages represent it as an **inter-connected graph** of objects. Mismatch problems:

Granularity	Object model usually has more classes than the number of corresponding tables in the database.
Subtypes	Inheritance in OOP does not exist in RDBMSs.
Identity	OOP supports identity ($a == b$) and equality ($a.equals(b)$) while RDBMSs only support identity (the primary key).
Association	References vs. foreign keys
Navigation	In OOP, navigation is from one association to another, walking the object network. That is inefficient in RDBMSs, so JOINs are used.

Definition

Hibernate is an ORM library for Java that solves object-relational impedance mismatch problems by replacing direct persistence-related DB access with high-level object handling functions.

- Allows developer to concentrate on Java code instead of complex SQL statements.
- Reduces the lines of code, makes the system more understandable and easier to maintain.
- Abstracts the application away from the underlying SQL database and dialect, fosters portability.

Definition

Hibernate is an ORM library for Java that solves object-relational impedance mismatch problems by replacing direct persistence-related DB access with high-level object handling functions.

- Allows developer to concentrate on Java code instead of complex SQL statements.
- Reduces the lines of code, makes the system more understandable and easier to maintain.
- Abstracts the application away from the underlying SQL database and dialect, fosters portability.

Definition

Hibernate is an ORM library for Java that solves object-relational impedance mismatch problems by replacing direct persistence-related DB access with high-level object handling functions.

- Allows developer to concentrate on Java code instead of complex SQL statements.
- Reduces the lines of code, makes the system more understandable and easier to maintain.
- Abstracts the application away from the underlying SQL database and dialect, fosters portability.

Definition

Hibernate is an ORM library for Java that solves object-relational impedance mismatch problems by replacing direct persistence-related DB access with high-level object handling functions.

- Allows developer to concentrate on Java code instead of complex SQL statements.
- Reduces the lines of code, makes the system more understandable and easier to maintain.
- Abstracts the application away from the underlying SQL database and dialect, fosters portability.

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                  column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                  column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                  column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                  column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                  column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Mapping file

- Class–Table
- Property–Id
- Property–Column

- Types and column names are optional.

```
<hibernate-mapping>
    <class name="Product" table="PRODUCTS">
        <id name="productId" column="pid" >
            <generator class="assigned" />
        </id>
        <property name="proName" column="pname" />
        <property name="price"/>
        <property name="date" type="timestamp"
                column="creation_date"/>
    </class>
</hibernate-mapping>
```

Hibernate
Configuration file

- JDBC connection information (connection.driver_class, connection.username, connection.password, connection.url)
- Number of connections (connection.pool_size)
- SQL variant (dialect)
- DB schema generation (hbm2ddl.auto)
- Mapping file name (mapping)

Hibernate
Configuration file

- JDBC connection information (connection.driver_class, connection.username, connection.password, connection.url)
- Number of connections (connection.pool_size)
- SQL variant (dialect)
- DB schema generation (hbm2ddl.auto)
- Mapping file name (mapping)

Hibernate
Configuration file

- JDBC connection information (connection.driver_class, connection.username, connection.password, connection.url)
- Number of connections (connection.pool_size)
- SQL variant (dialect)
- DB schema generation (hbm2ddl.auto)
- Mapping file name (mapping)

Hibernate
Configuration file

- JDBC connection information (connection.driver_class, connection.username, connection.password, connection.url)
- Number of connections (connection.pool_size)
- SQL variant (dialect)
- DB schema generation (hbm2ddl.auto)
- Mapping file name (mapping)

Hibernate
Configuration file

- JDBC connection information (connection.driver_class, connection.username, connection.password, connection.url)
- Number of connections (connection.pool_size)
- SQL variant (dialect)
- DB schema generation (hbm2ddl.auto)
- Mapping file name (mapping)

Hibernate
Configuration file

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/javawebtutor</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">123</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="hbm2ddl.auto">create</property>
    <mapping resource="com/jwt/hibernate/student.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate
Sample

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class SimpleTest {

    public static void main(String[] args) {

        Configuration cfg = new Configuration();
        cfg.configure("hibernate.cfg.xml");

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();
        Student student = new Student();
        student.setName("Ali");
        student.setNo("101");
        student.setPhone("8888");
        student.setDegree("M.S");

        Transaction tx = session.beginTransaction();
        session.save(student);
        System.out.println("Object saved successfully!");
        tx.commit();
        session.close();
        factory.close();
    }
}
```

Hibernate
Sample

```
<hibernate-mapping>
  <class name="com.jwt.hibernate.Student" table="STUDENT">
    <id column="ID" name="id" type="long"/>
    <property column="STUDENT_NAME" name="name" type="string"/>
    <property column="DEGREE" name="degree" type="string"/>
    <property column="ROLL" name="roll" type="string"/>
    <property column="PHONE" name="phone" type="string"/>
  </class>
</hibernate-mapping>
```

```
mysql> select * from student;
+----+--------------+--------+------+-------+
| ID | STUDENT_NAME | DEGREE | NO   | PHONE |
+----+--------------+--------+------+-------+
|  0 | Ali          | M.S    | 101  | 8888  |
+----+--------------+--------+------+-------+
```

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm

References and Further Reading

- http://www.javacodegeeks.com/2011/01/10-tips-proper-application-logging.html
- http://blogg.kantega.no/logging-in-java-with-users-in-mind/
- http://logging.apache.org/log4j/2.x/
- http://www.agiledata.org/essays/impedanceMismatch.html
- http://hibernate.org/orm/
- http://www.java4s.com/hibernate/hibernate-hello-world-program-saving-an-objcet/
- https://www.tutorialspoint.com/log4j/log4j_configuration.htm
- http://www.allapplabs.com/log4j/log4j.htm