CS 525 - ASD

# Advanced Software Development

## MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

# Advanced Software Development



Maharishi University
OF MANAGEMENT

# Lesson 2

L1: ASD Introduction
L2: Strategy, Template method
L3: Observer pattern
L4: Composite pattern, iterator pattern
L5: Command pattern
L6: State pattern
L7: Chain Of Responsibility pattern

Midterm

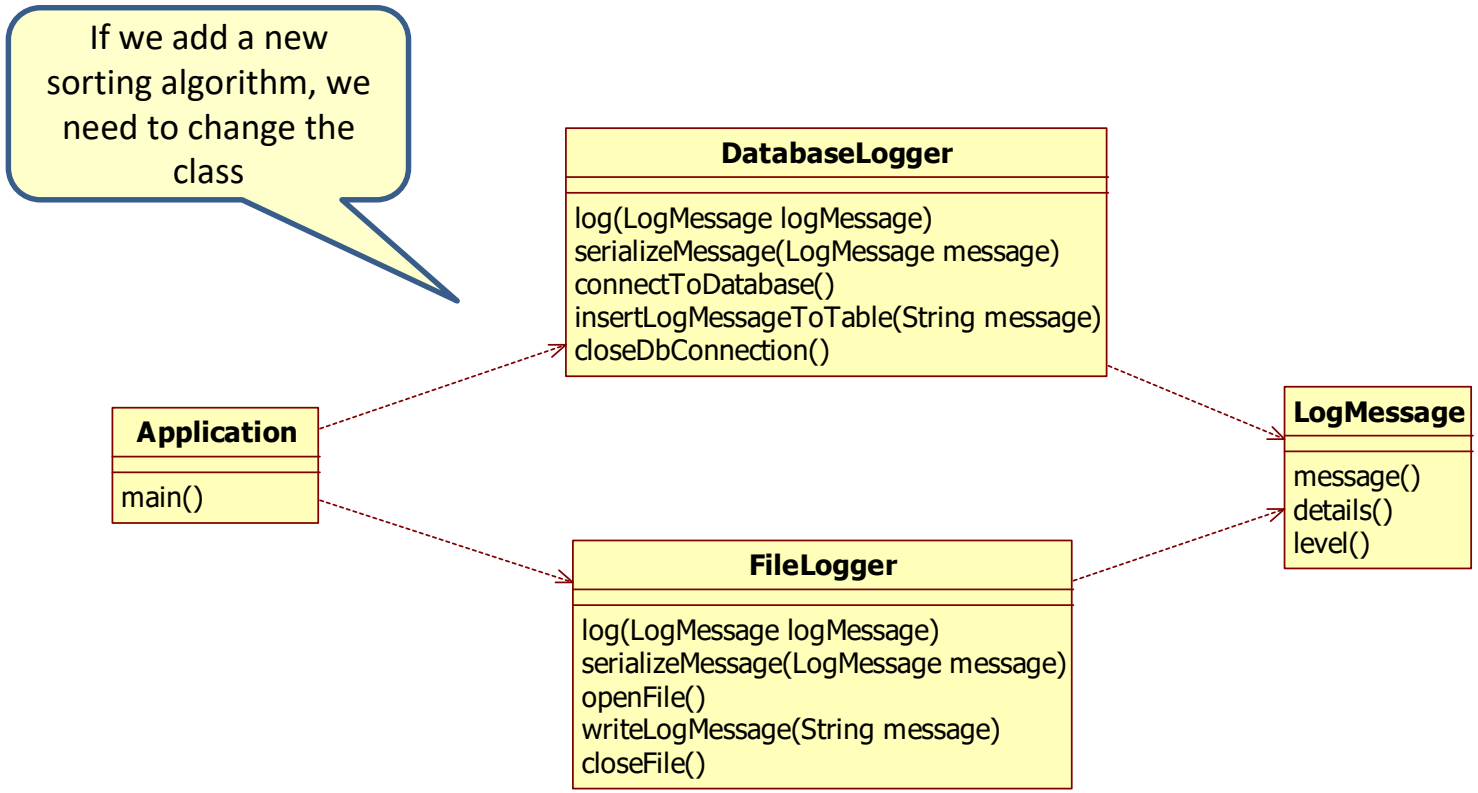
L8: Proxy, Adapter, Mediator
L9: Factory, Builder, Decorator, Singleton
L10: Framework design
L11: Framework implementation
L12: Framework example: Spring framework
L13: Framework example: Spring framework

Final

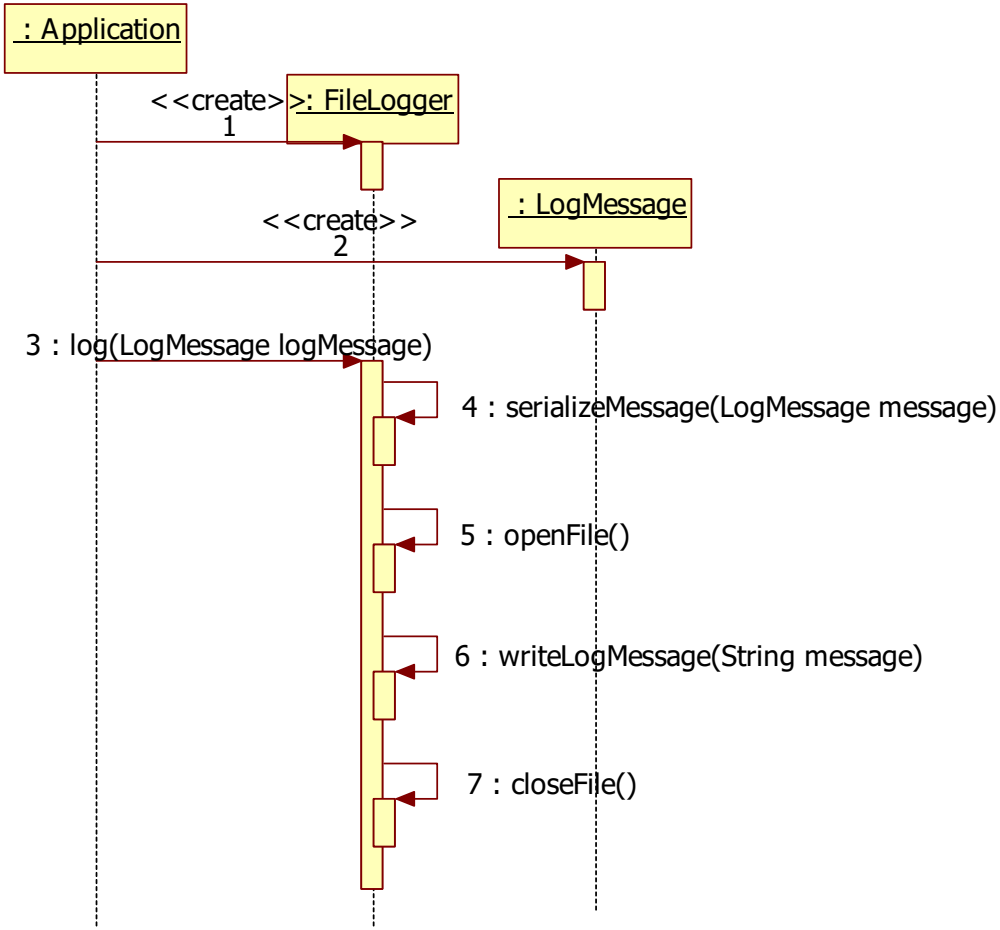
# Template method

- The template method pattern defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure

# Logging to different sources



# Using the FileLogger



# DatabaseLogger

```
public class DatabaseLogger {
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    connectToDatabase();
    insertLogMessageToTable(messageToLog);
    closeDbConnection();
  }
  private String serializeMessage(LogMessage message) {
    System.out.println("Serializing message");
    return message.toString();
  }
  private void connectToDatabase() {
    System.out.println("Connecting to Database.");
  }
  private void insertLogMessageToTable(String message) {
    System.out.println("Inserting Log Message to DB table : " + message);
  }
  private void closeDbConnection() {
    System.out.println("Closing DB connection.");
  }
}
```

# FileLogger

```
public class FileLogger{
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    openFile();
    writeLogMessage(messageToLog);
    closeFile();
  }
  private String serializeMessage(LogMessage message) {
    System.out.println("Serializing message");
    return message.toString();
  }
  private void openFile() {
    System.out.println("Opening File.");
  }
  private void writeLogMessage(String message) {
    System.out.println("Appending Log message to file : " + message);
  }
  private void closeFile() {
    System.out.println("Close File.");
  }
}
```

# LogMessage

```
public class LogMessage {
  enum LogLevel {
    WARNING,
    INFO,
    ERROR
  }

  private String message;
  private String details;
  private LogLevel level;

  public LogMessage(String message, String details, LogLevel level) {
    this.message = message;
    this.details = details;
    this.level = level;
  }

  @Override
  public String toString() {
    return "LogMessage "+LocalDate.now()+" - "+LocalTime.now()+" [message=" +
            message + ", details=" + details + ", level=" + level + "]";
  }
}
```

# Application

```
public class Application {

  public static void main(String[] args) {
    FileLogger fileLogger = new FileLogger();
    LogMessage message = new LogMessage("cannot send email", "smpt server
                smtp.acme.com cannot be reached", LogLevel.ERROR);
    fileLogger.log(message);

    System.out.println("------------------------");

    DatabaseLogger databaseLogger = new DatabaseLogger();
    LogMessage message2 = new LogMessage("subject is empty", "this email has no
               subject, emails should have a subject", LogLevel.INFO);
    databaseLogger.log(message2);
  }
}
```

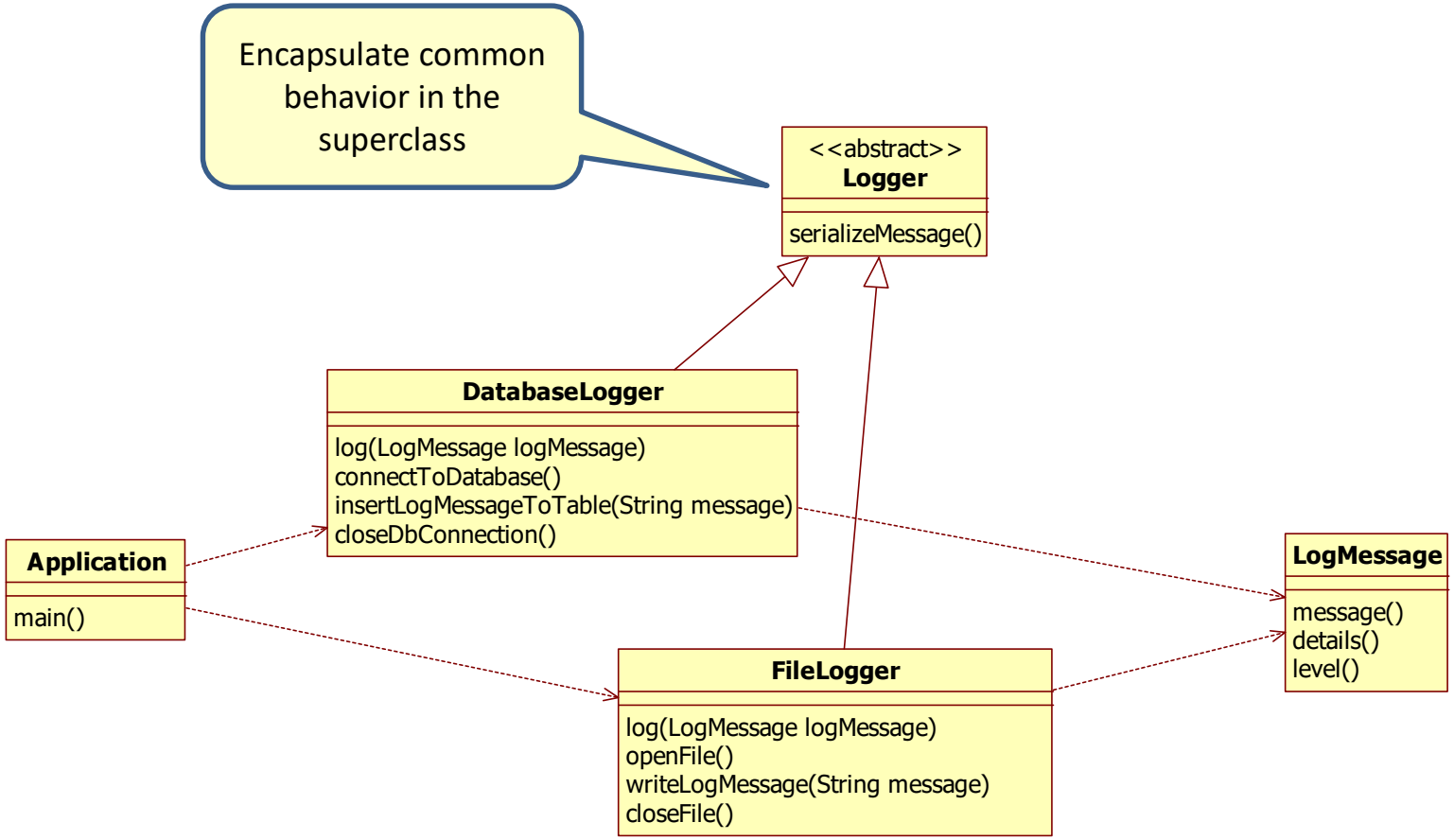
# Common behavior

```
public class FileLogger{
  public void log(LogMessage message) {...}

  private String serializeMessage(LogMessage message) {
    System.out.println("Serializing message");
    return message.toString();
  }
  private void openFile() {...}
  private void writeLogMessage(String message) {...}
  private void closeFile() {...}
}
```

```
public class DatabaseLogger {
  public void log(LogMessage message) {...}

  private String serializeMessage(LogMessage message) {
    System.out.println("Serializing message");
    return message.toString();
  }
  private void connectToDatabase() {...}
  private void insertLogMessageToTable(String message) {...}
  private void closeDbConnection() {...}
}
```

Serializing the LogMessage is the same for all loggers

# Inheritance



# DatabaseLogger

```
public abstract class Logger {
    protected String serializeMessage(LogMessage message) {
        System.out.println("Serializing message");
        return message.toString();
    }
}
```

```
public class DatabaseLogger extends Logger {
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    connectToDatabase();
    insertLogMessageToTable(messageToLog);
    closeDbConnection();
  }
  private void connectToDatabase() {
    System.out.println("Connecting to Database.");
  }
  private void insertLogMessageToTable(String message) {
    System.out.println("Inserting Log Message to DB table : " + message);
  }
  private void closeDbConnection() {
    System.out.println("Closing DB connection.");
  }
}
```

# FileLogger

```
public abstract class Logger {
    protected String serializeMessage(LogMessage message) {
        System.out.println("Serializing message");
        return message.toString();
    }
}
```

```
public class FileLogger extends Logger {
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    openFile();
    writeLogMessage(messageToLog);
    closeFile();
  }
  private void openFile() {
    System.out.println("Opening File.");
  }
  private void writeLogMessage(String message) {
    System.out.println("Appending Log message to file : " + message);
  }
  private void closeFile() {
    System.out.println("Close File.");
  }
}
```

# A common algorithm

```
public class DatabaseLogger extends Logger {
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    connectToDatabase();
    insertLogMessageToTable(messageToLog);
    closeDbConnection();
  }
  ...
}
```

Open/connect to repository

Write log message to repository

Close repository

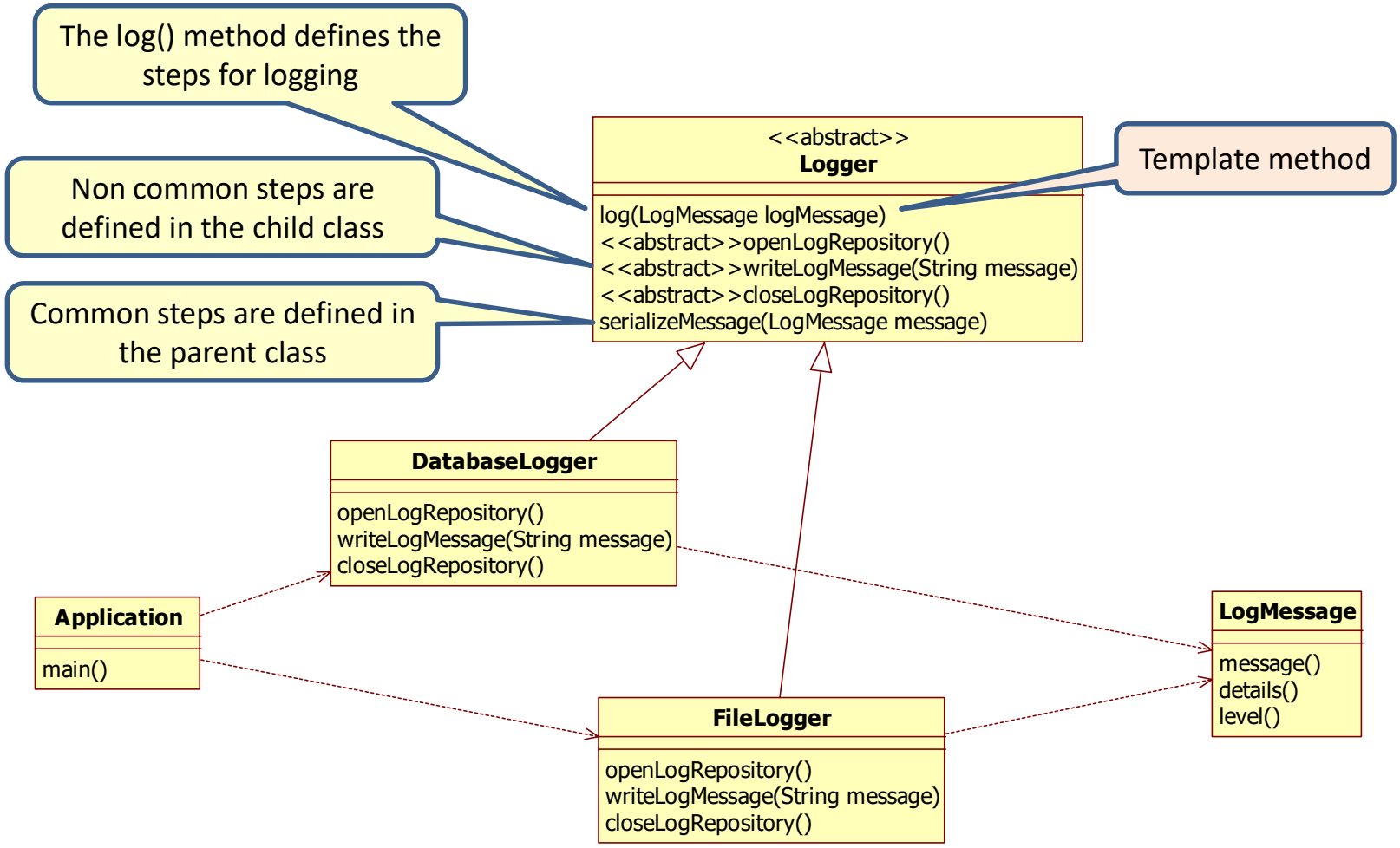
```
public class FileLogger extends Logger {
  public void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    openFile();
    writeLogMessage(messageToLog);
    closeFile();
  }
  ...
}
```

Open/connect to repository

Write log message to repository

Close repository

# Template method pattern



# Logger

```
public abstract class Logger {
  protected void log(LogMessage message) {
    String messageToLog = serializeMessage(message);
    openLogRepository();
    writeLogMessage(messageToLog);
    closeLogRepository();
  }
  protected abstract void openLogRepository() ;
  protected abstract void writeLogMessage(String message);
  protected abstract void closeLogRepository();

  protected String serializeMessage(LogMessage message) {
    System.out.println("Serializing message");
    return message.toString();
  }

}
```

Template method

The log() method defines the steps for logging

Non common steps are defined in the child class

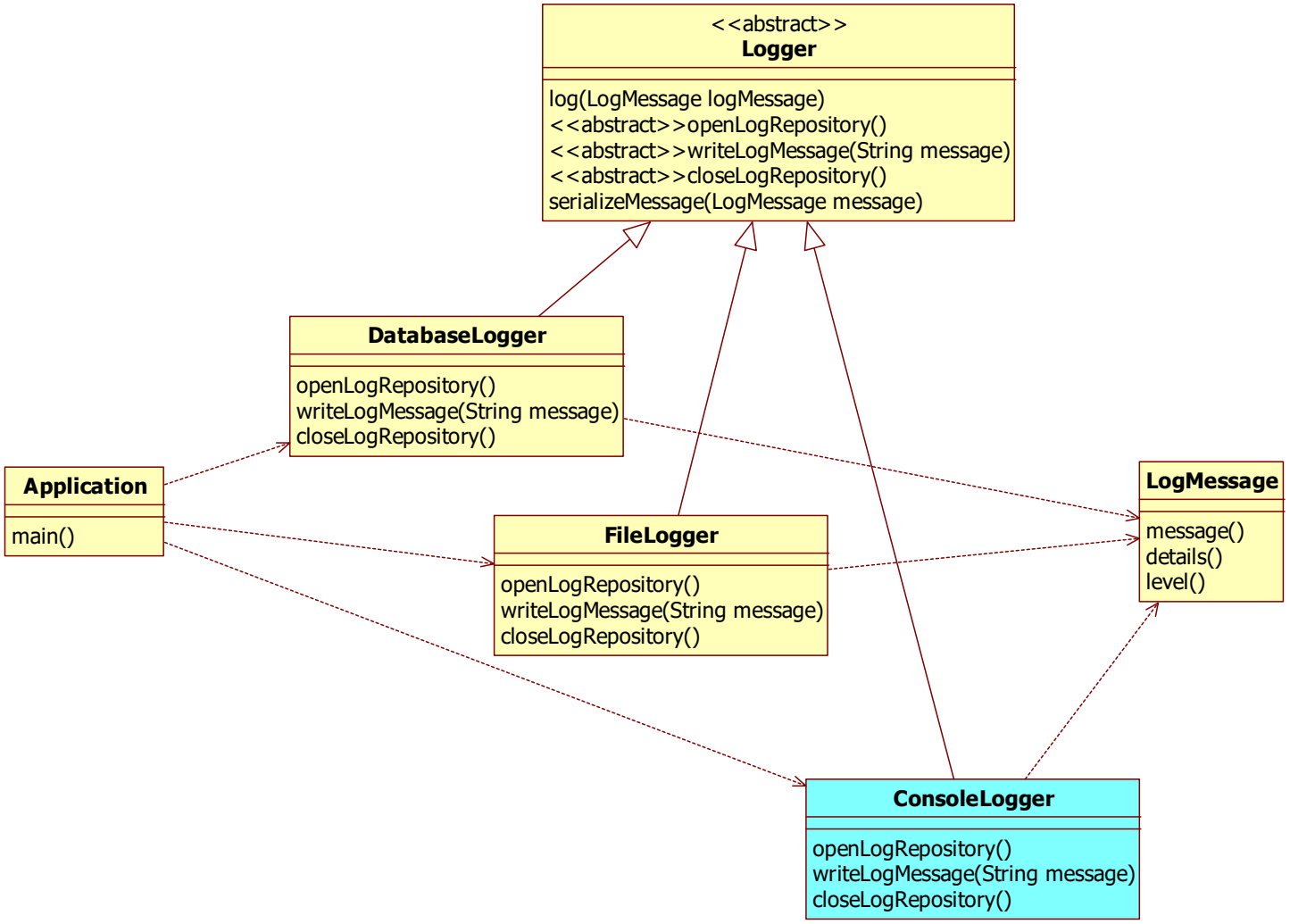
Common steps are defined in the parent class

# Concrete loggers

```
public class DatabaseLogger extends Logger{

  protected void openLogRepository() {
    System.out.println("Connecting to Database.");
  }
  protected void writeLogMessage(String message) {
    System.out.println("Inserting Log Message to DB table : " + message);
  }
  protected void closeLogRepository() {
    System.out.println("Closing DB connection.");
  }
}
```

```
public class FileLogger extends Logger{

  protected void openLogRepository() {
    System.out.println("Opening File.");
  }
  protected void writeLogMessage(String message) {
    System.out.println("Appending Log message to file : " + message);
  }
  protected void closeLogRepository() {
    System.out.println("Close File.");
  }
}
```

# Let’s add a ConsoleLogger



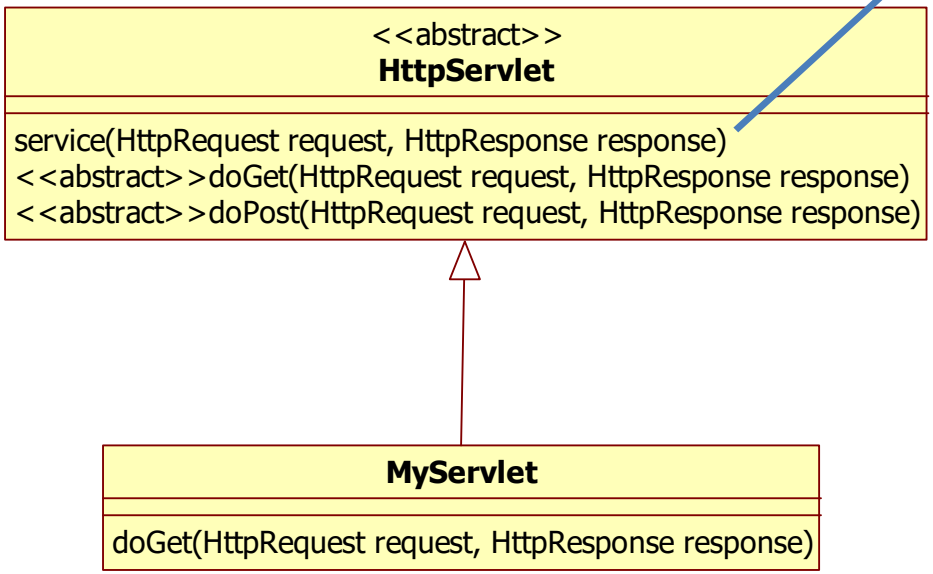
# ConsoleLogger

```
public class ConsoleLogger extends Logger {

  protected void openLogRepository() {}
  protected void writeLogMessage(String message) {
   System.out.println("Console logger: "+message);
  }
  protected void closeLogRepository() {}
}
```

```
ConsoleLogger consoleLogger = new ConsoleLogger();
LogMessage message3 = new LogMessage("this email has multiple receivers", "try
       the mailmerge functionality", LogLevel.WARNING);
consoleLogger.log(message3);
```

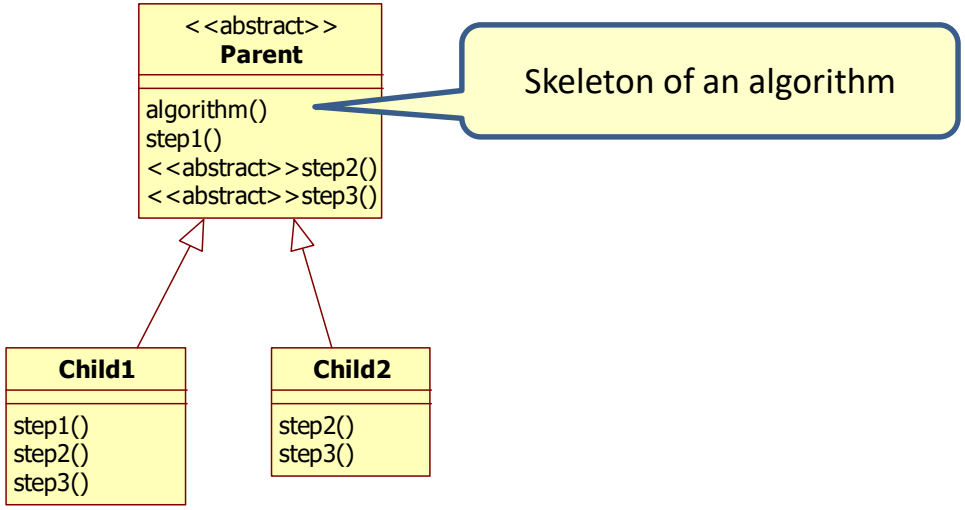
# Example of template method

```
if (request.getType().equals("GET"))
  doGet(request, response);
else if (request.getType().equals("POST"))
  doPost(request, response);
```



# Template method pattern

- What problem does it solve?
  - Whenever you have an algorithm with different steps that is used in different situations, then define this algorithm in one place (parent class) and let child classes implement the concrete steps.

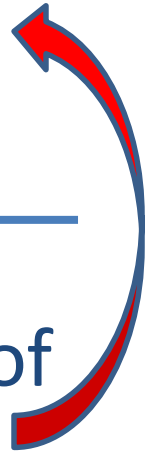


# Main point

- The template method defines an algorithm in the parent class and the different steps can be implemented in the child class(es)
- Every human being has free will to decide how to live your life within the boundaries of the laws of nature.

# Connecting the parts of knowledge with the wholeness of knowledge

1. Strategies are encapsulated algorithms that can be plugged in at runtime.
2. The template method can be used when you have a stable algorithm whose individual steps may vary.



3. **Transcendental consciousness** is the home of all the laws of nature.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that everything is an expression of the same Unified Field.