TEL-RAN

Java Course-Part 2

Dr. Yuri Granovsky

Content

User Datagram Protocol (UDP)

- UDP provides an unreliable packet delivery system built on top of the IP protocol
- Connectionless communication
  - No relation between packets
  - No state machine
- Speed & no overhead
- Datagram packet has IP address and Port number

UDP Client

- Datagram packet creation for sending to a server

  DatagramPacket spacket = new DatagramPacket(<array of bytes>, <number of bytes>, <InetAdress object>, <port number>);
- UDP Socket creation

  DatagramSocket socket = new DatagramSocket();
- Sending packet

  socket.send(spacket);
- Datagram packet creation for receiving response from a server

  DatagramPacket rpacket = new DatagramPacket(<array of bytes>, <number of bytes>);
- Receiving packet

  socket.receive (rpacket);
  - It blocks running thread until response comes
  - To set socket on timeout mode allowing the waiting no more than pointed time in milliseconds. After that time ***InterruptedIOException*** will be thrown

  socket.setSoTimeout(<time in milliseconds>);

UDP Server

- Creation of socket listening to packets from a client on determined port

  DatagramSocket socket = new DatagramSocket(<port number>);
- Creation of a packet for receiving

  DatagramPacket rpacket = new DatagramPacket(<array of bytes>, <number of bytes>);
- Block on receive

  socket.receive(rpacket);
- Find out where packet came from so we can reply to the same host/port

  InetAddress remoteHost = rpacket.getAddress();

  int remotePort = rpacket.getPort();
- Extract the packet data and packet real length

  byte[] data = packet.getData();

  Int length=packet.getLength();
- Sending response to a client

  DatagramPacket spacket = new DatagramPacket (<array containing response>, <length of array with response>, remoteHost, remotePort);

  Socket.send(spacket);

UDP Client/Server

- Implement simple Echo protocol
  - Client gets string from the console and sends it to server
  - Server responses a message containing the same string concatenated with “response from server”
  - Client displays the received message on the console
- Implement the following protocol Server has array of the “clever” messages like “don’t forget drink water”
  - Client may send the following requests
    - “getNumberOfMessages”
    - “getMessage”
  - Server on the request “getNumberOfMessages” sends the client the length of the messages array
  - Server on the request “getMessage” sends the client random message
  - Server on an unknown requests closes connection with the client
  - Using that protocol write client printing out all non duplicated “clever” messages the server owns

Application Layer – Class URL

- URL – Uniform Resource Locator http://www.google.com – Protocol & Resource
  - Creation of URL object
    - 6 constructors with meaning of a URL string definition and throwing ***MalformedURLException*** in the case a protocol is not supported

    URL url = new URL(<url string>);
  - Retrieving Data from URL

```
public final InputStream openStream(  ) throws IOException
public URLConnection openConnection(  ) throws IOException
public final Object getContent(  ) throws IOException
```

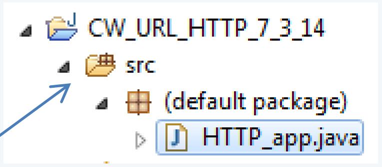
- URLConnection class has method ***getInputStream()*** and ***getOutputStream***

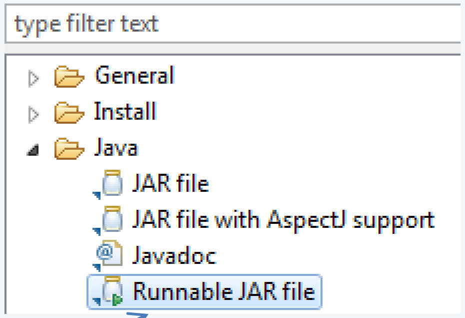
Exercise- HTTP Protocol

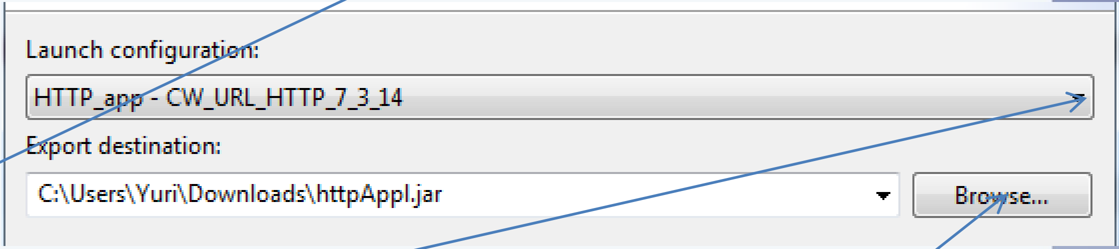
1. Display on console the content of the initial HTML page from
   - The site name shall be entered from the console
2. Display on console selected lines of the initial HTML page
   - The site name shall be entered from the console
   - Only lines containing the string that shall be entered from the console

Runnable JAR - Creation

- Right click on the class containing main method
  - Select Java
    - Select Runnable JAR file

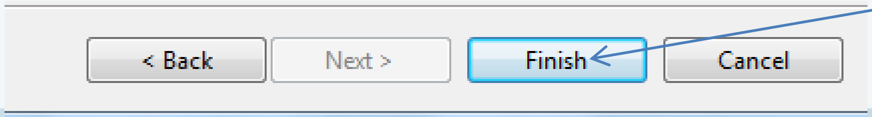






Launch configuration:
HTTP_app - CW_URL_HTTP_7_3_14
Export destination:
C:\Users\Yuri\Downloads\httpAppl.jar

- Select launch configuration
- Select export destination
- Press "Finish"

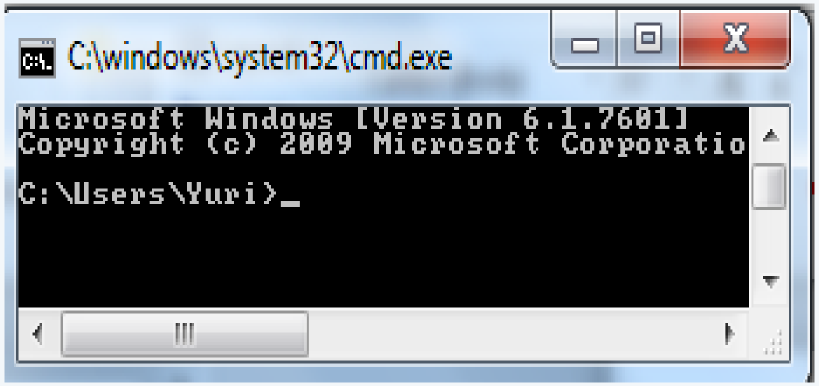


Runnable JAR - Running

- Launching windows console
- Run command

java –version

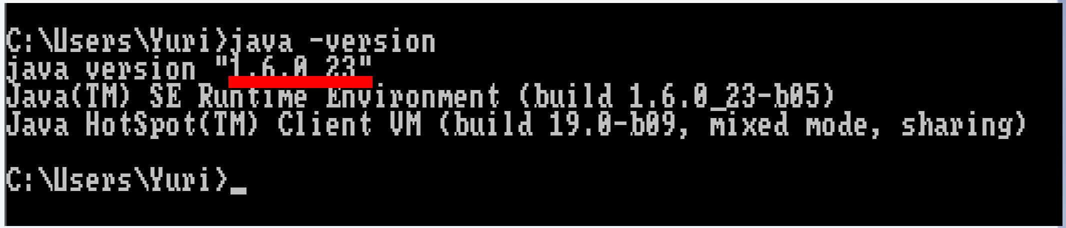
- – JRE Version mismatches JDK version or command ***java*** doesn't exist – see next slide
  - inserting path of the java run time environment in the system path variable – see next slide (162-163)
- – JRE Version matches JDK version
  - Running runnable JAR (see slide 164)



C:\windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporatio

C:\Users\Yuri>_
```

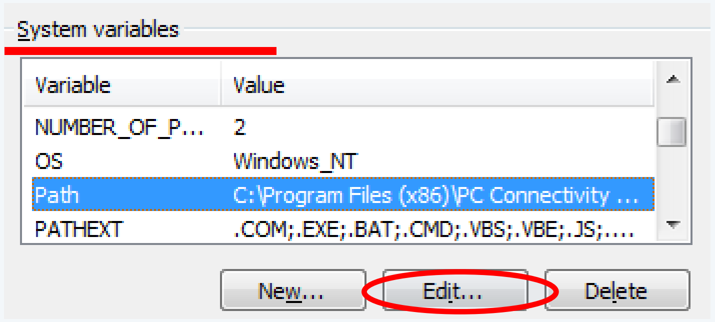


```
C:\Users\Yuri>java -version
java version "1.6.0_23"
Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)

C:\Users\Yuri>_
```

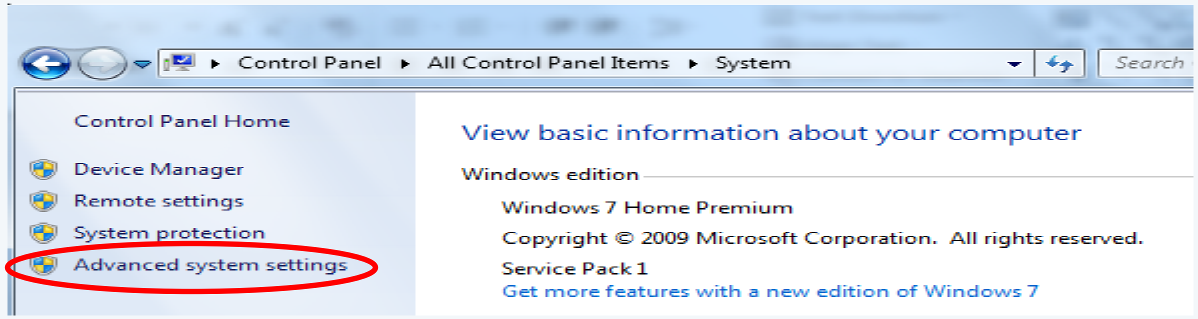
JRE Path Insertion (Windows 7) Access to Editing System Variable *Path*

- Control Panel->All Control Panel Items-> System
  - Advanced system setting
  - Environment Variables
    - System variables
      - Select ***Path*** variable
      - Press ***Edit***



System variables

Variable	Value
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\Program Files (x86)\PC Connectivity ...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....



View basic information about your computer

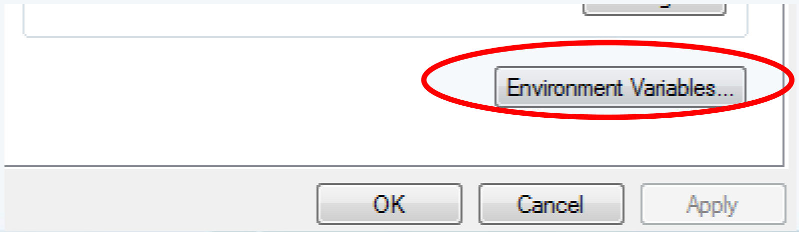
Windows edition

Windows 7 Home Premium

Copyright © 2009 Microsoft Corporation. All rights reserved.

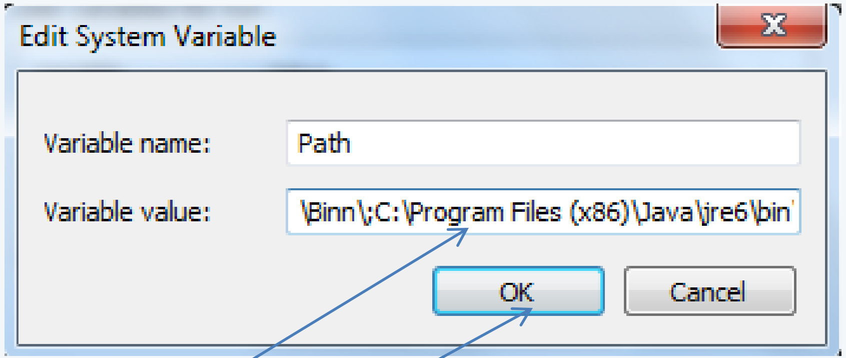
Service Pack 1

Get more features with a new edition of Windows 7



JRE Path Insertion (Windows 7) Update System Variable ***Path***

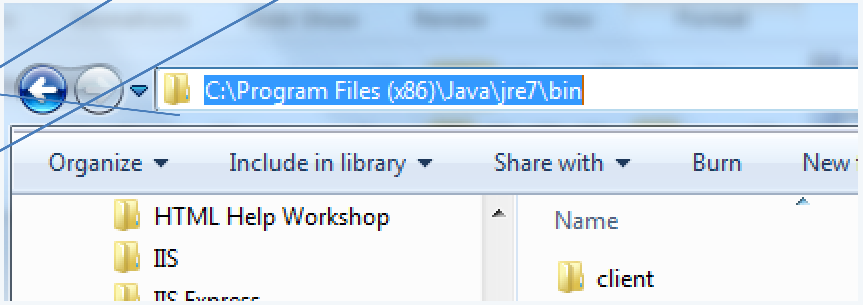
- Either update existing path to JRE bin directory (if the path exists) or add after ';' new path
- How to get the path of JRE bin
  - Find and enter into directory bin of JRE
    - Copy to clipboard path to bin directory
      - Paste from clipboard to the proper part of the Variable value (be careful not to damage existing paths)
- After update/insert press OK on all open screens of the "Control Panel" dialogs
- Close cmd console and launch it again (slide 161)



Edit System Variable

Variable name: Path

Variable value: \Binn\;C:\Program Files (x86)\Java\jre6\bin

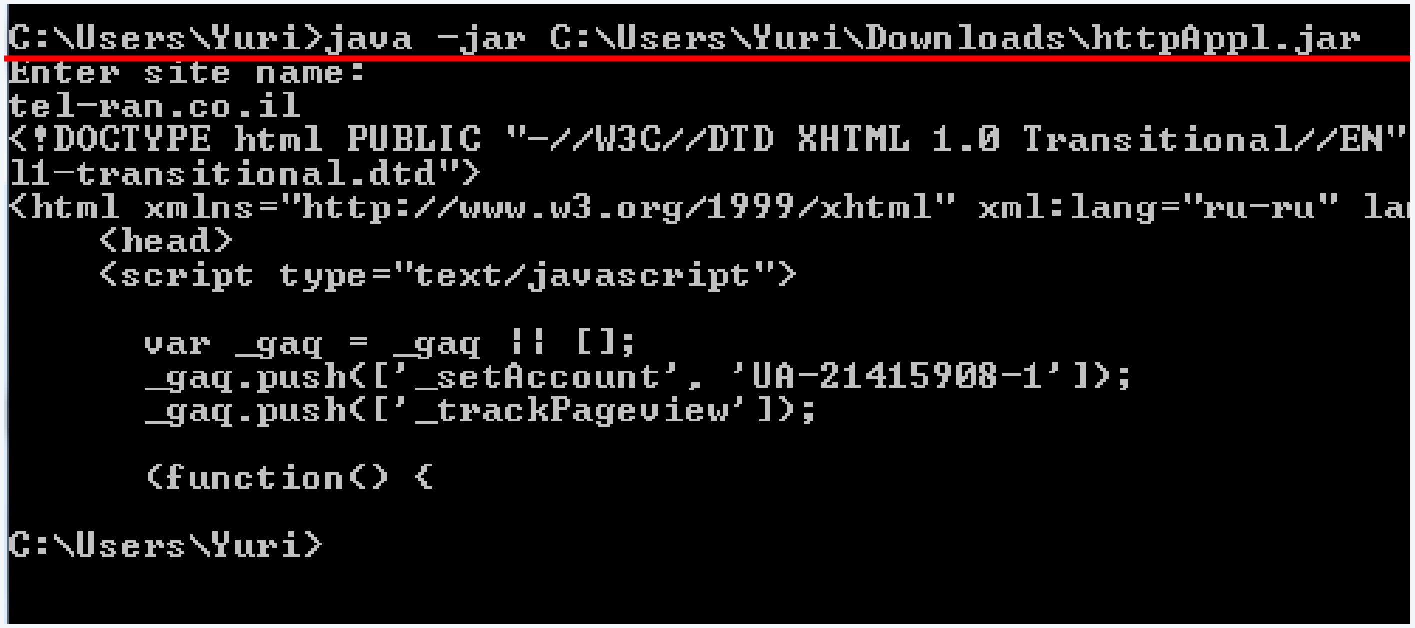


C:\Program Files (x86)\Java\jre7\bin

Running Runnable JAR

- Running command

java –jar <full path to the runnable jar>



```
C:\Users\Yuri>java -jar C:\Users\Yuri\Downloads\httpAppl.jar
Enter site name:
tel-ran.co.il
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
11-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru-ru" la
    <head>
    <script type="text/javascript">

      var _gaq = _gaq || [];
      _gaq.push(['_setAccount', 'UA-21415908-1']);
      _gaq.push(['_trackPageview']);

      (function() {

C:\Users\Yuri>
```

Exercise – Check a Site Existence

- Write Java application and create runnable jar for checking whether a site exists
  - Command Line Arguments:
    - Name of site (for example “Tel-Ran”)
    - Name of the configuration file containing names of the domains (for example “co.il”)
      - Each domain name resides in the separate line of the configuration file
  - Output of the application
    - If the site exists the output shall contain name(s) of domain(s) of the site
    - If the site doesn’t exist the output shall contain a proper message

Thread Creation

- *Class implementing Runnable Interface*

```
class X implements Runnable {
        ………..
        @Override
        public void run(){
        ……
        }
}
class Appl {
static public main(String args[]) {
X r1=new X();
Thread t1=new Thread(r1);
t1.start();                    }
}
```

- *Class extending standard class Thread with overriding method run()*

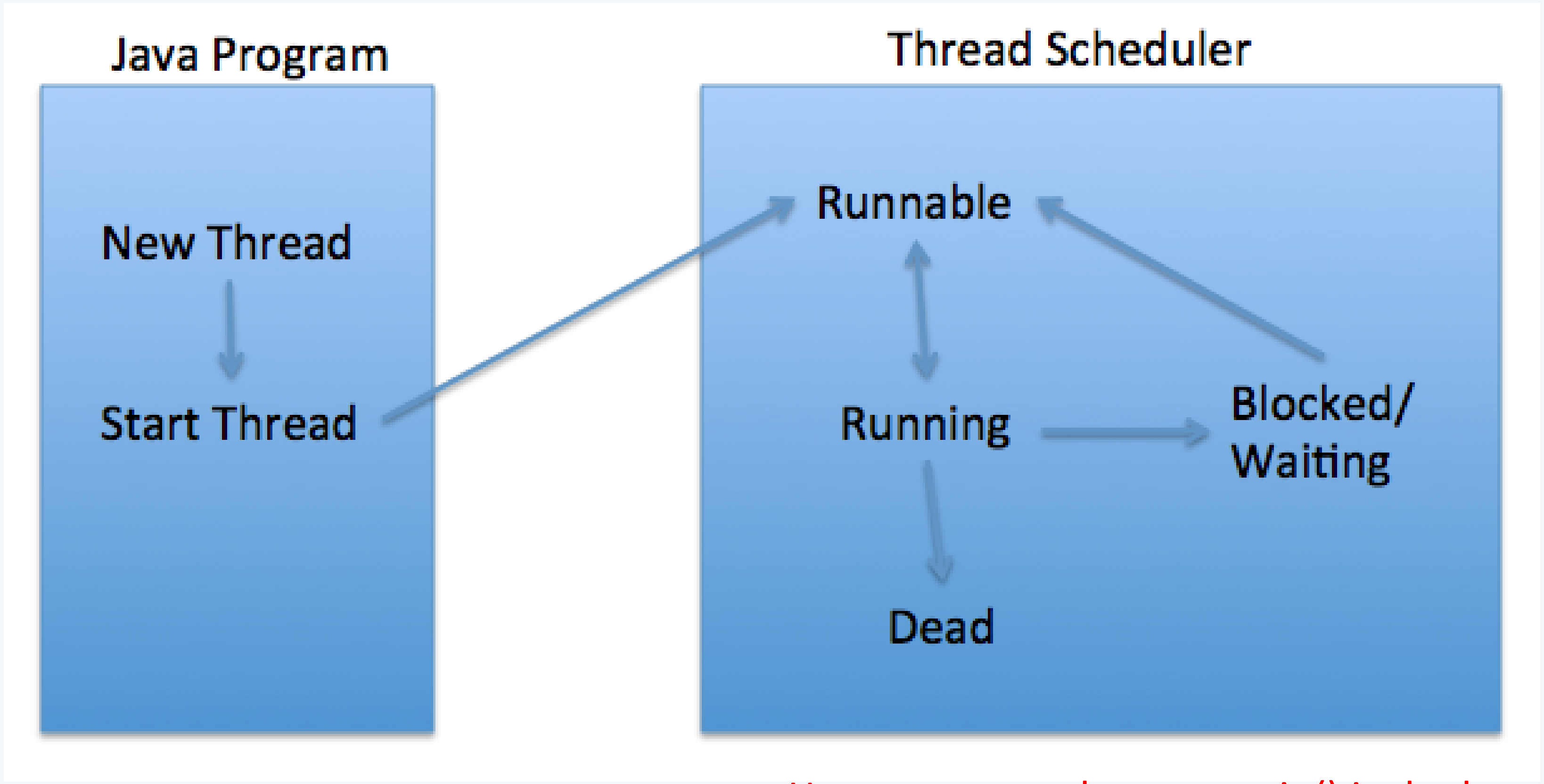
```
        class X extends Thread {
                ………..
                @Override
                public void run(){
                ……
                }
        }

class Appl {
        static public main(String args[]) {
        X t1=new X();
        t1.start();
                }
        }
```

User Thread and Daemon Thread

- When we create a Thread in java program, it’s known as user thread
  - When there are no user threads running, JVM shutdown the program and quits.
- A daemon thread runs in background and doesn’t prevent JVM from terminating
  - Before calling method ***start()*** the method ***setDaemon(true)*** should be called
  - A child thread created from daemon thread is also a daemon thread.

Thread Lifecycle

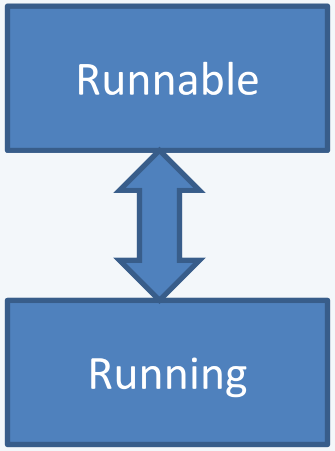


Can we call ***run()*** method of a Thread class?
Yes we can but there will not be multithreading

How can we make sure main() is the last thread to finish in Java Program?
Method join for thread will wait for thread finishing

Thread Scheduling

- Thread priority
  - setPriority(int priority);
    - Thread.MAX_PRIORITY
    - Thread.MIN_PRIORITY
    - Thread.NORMAL_PRIORITY
- Time slicing
- Application scheduling
  - yield();interrupt();notify()
  - sleep(); wait()



How can we pause the execution of a Thread for specific time?

Static method sleep with pointing time in the milliseconds

How can we may interrupt thread?

Method interrupt moves thread in the InterruptedException catch

Exercise-Thread Control

- The main thread starts the child thread
- Each second the child thread prints on console any symbol from string given by the main thread (15 symbols on the line)
- Each time period defined through main argument the main thread sends the child thread the interruption for printing another symbol
- After 15 interruptions the main and the child threads should be finished

Exercise – Threads Scheduling

- **Write the following code:**
  - **One thread object prints number 1 for 100000 times**
  - **Second thread object prints number 2 for 100000 times**
  - **After printing some portion, for example ,10 numbers ,the “running” should be moved to another thread**
- **For advanced students: number of threads is undefined and each thread prints its number by portions with determined amount of numbers**

Notes:

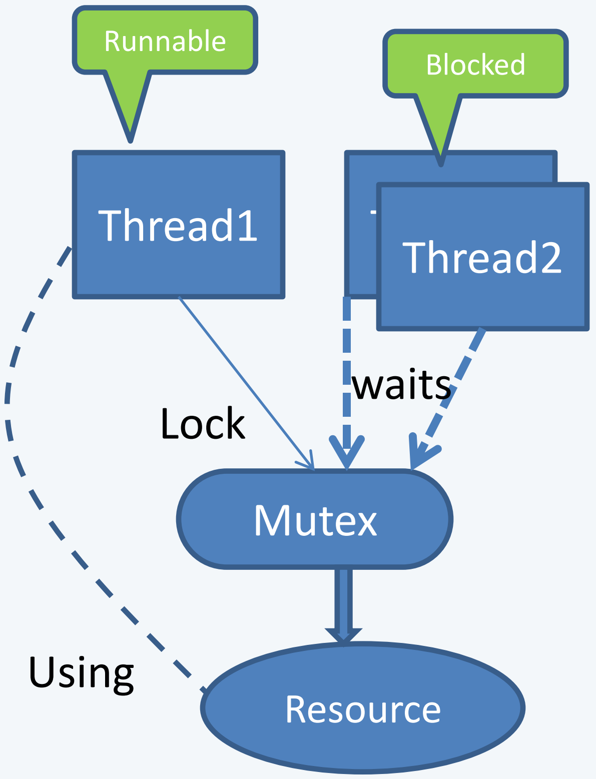
- **One thread object contains reference to other**
- **Method run of the class extending Thread class performs the following**
  1. **Prints the given portions (for example 10) of the given number (for example either 1 or 2)**
  2. **Interrupts other object thread**
  3. **Sleep for 10 milliseconds**

Output example:

```
1111111111
2222222222
1111111111
2222222222
…………………
1111111111
2222222222
```

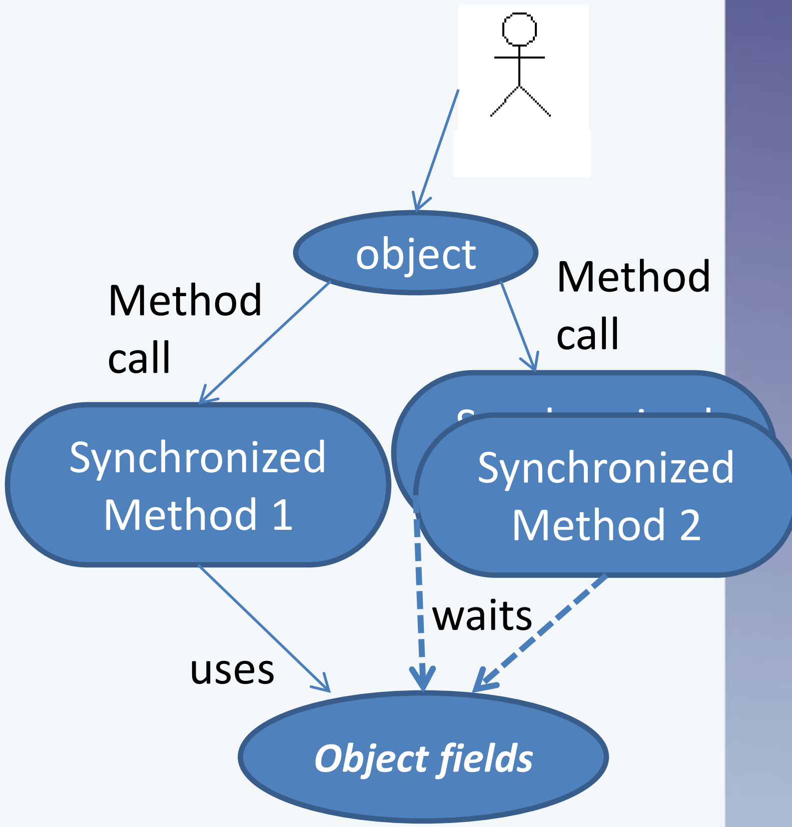
Monitors

- Mutually Exclusive Locking Mechanism (mutex)
- Synchronized method
  - Exclusive Lock for all methods of "this" object until returning or throwing exception from the method
- Synchronized block / statement
  - More flexible for performing simultaneously noncurrent code
- Lock interface
  - Lock implementations provide additional functionality over the use of synchronized methods and statements by providing a non-blocking attempt to acquire a lock (***tryLock()),*** an attempt to acquire the lock that can be interrupted (***lockInterruptibly(),*** and an attempt to acquire the lock that can timeout (***tryLock(long, TimeUnit)).***
  - ***ReentrantLock*** , ***ReentrantReadWriteLock*** – Lock implementations
- Atomic operations (AtomicInteger, Atomic…)



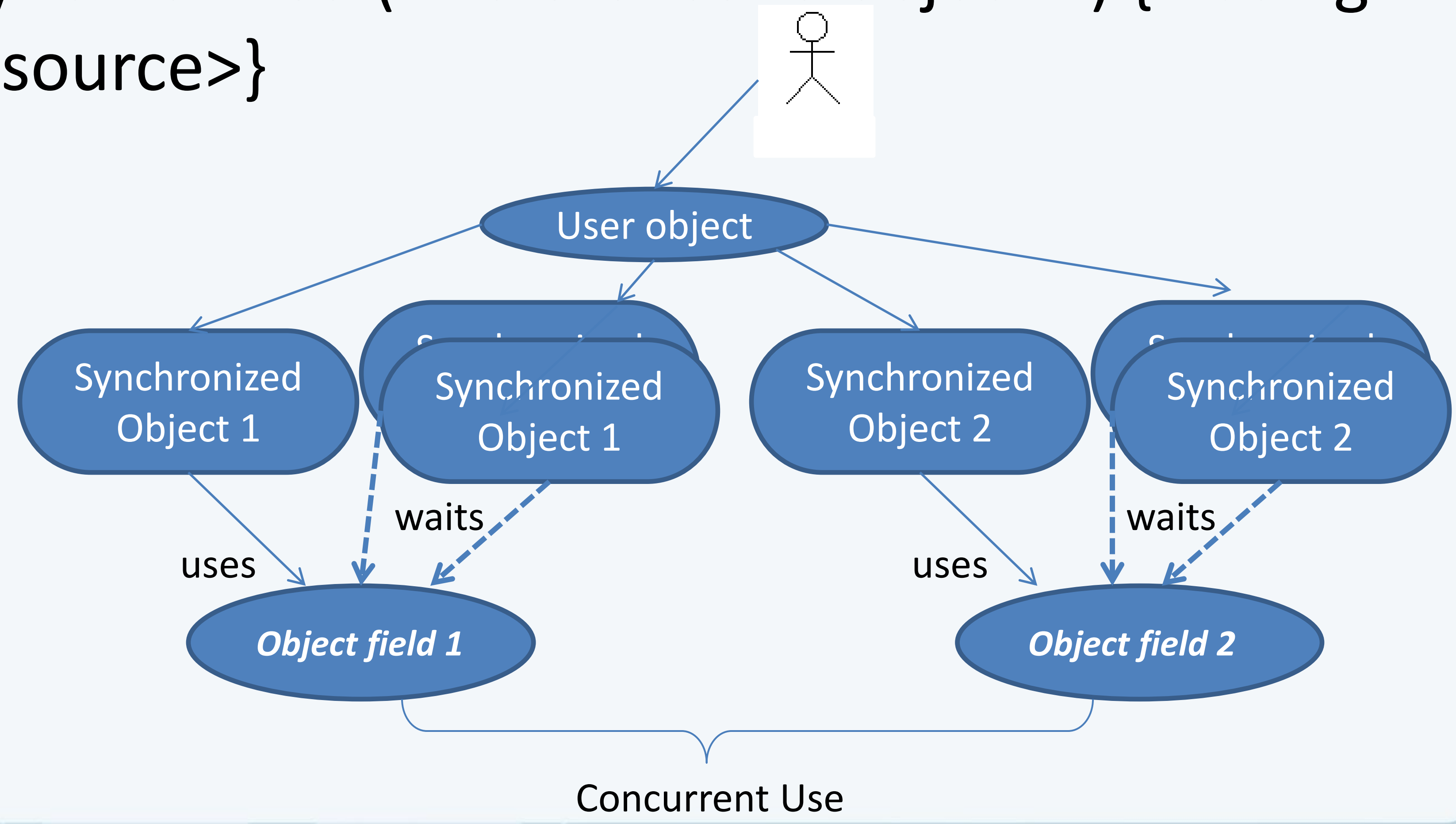
Synchronized Method

```
public class SynchronizedCounter
{
        private int c = 0;
        public synchronized void increment()
        {
                c++;
        }
        public synchronized void decrement()
        {
                c--;
        }
        public synchronized int value()
        {
                return c;
        }
}
```



Synchronized Block/Statement

synchronized (<reference to object>) { <using resource>}



Atomic Operations

- Operations that can’t be interrupted
- Atomic operations are as follows
  - Reading (return) and writing (assignment) primitives and references to objects except ***long*** and ***double*** types
  - Operations with Atomic data types
- Atomic data types
  - ***AtomicBoolean***
  - ***AtomicInteger***
  - ***AtomicLong***
  - ***AtomicIntegerArray***
  - ***AtomicLongArray***

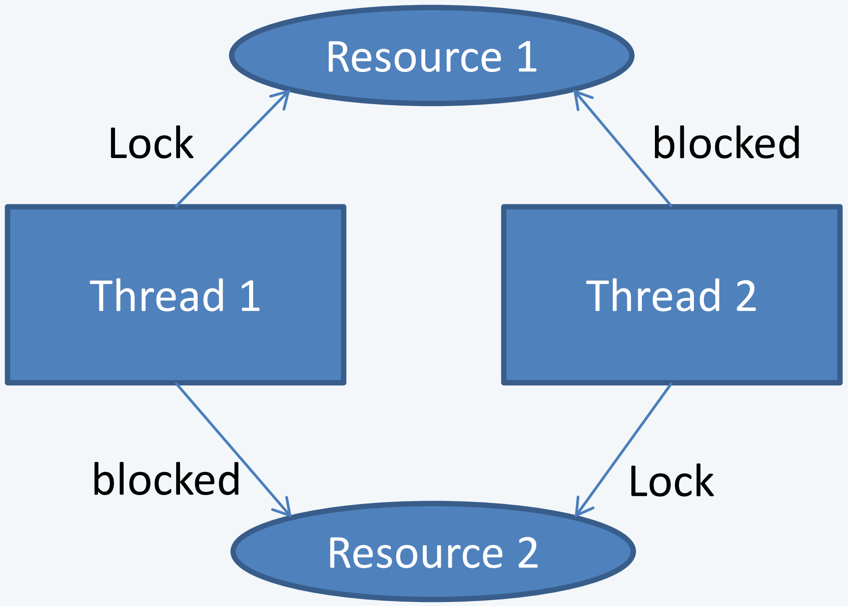
Exercise – TCP Messaging Client/Server Concurrent Server

- Update of the Server part of the clever messaging client/server described on the slide #150
  - Possibility of establishing several connections for serving several clients simultaneously
  - Request from a client “getNumberConnections”
  - Response from the server on request “getNumberConnections” containing number of the established connections
- Update of the Client part of the clever messaging client/server described on the slide #150
  - Add possibility of establishing several connections for serving several clients simultaneously in the existing code
  - Write new client application allowing getting number of the clients being served on server

TEL-RAN

DeadLock

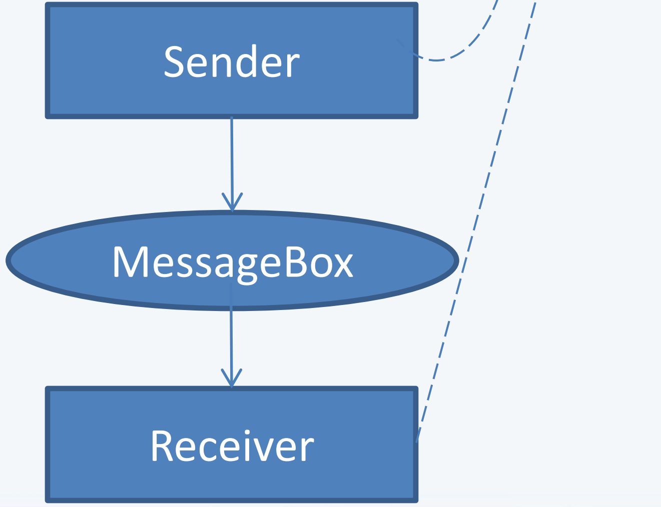
- Deadlock is a programming situation where two or more threads are blocked forever
- Deadlock may happen with at least two threads and two or more resources.



Write the example of a code according to the picture illustrating a deadlock .

Communication between Threads

```
public class MessageBox {
private String message =   null;
public synchronized void
putMessage(String msg) {
            message = msg;
            notify ();
}
```



```
public synchronized String
getMessage() {
            String msg = null;
try {
            while(message==null)
                        wait();
            msg = message;
            message=null;
}
catch(Exception ex){
       ex.printStackTrace();
}
return msg;
}
```

Questions – notify, notifyAll,wait

- Which Java class has notify and wait methods?
  - Class Object, that is all classes have these methods
- What's difference between notify and notifyAll ?
  - ***notify*** method will only notify one Thread and ***notifyAll*** method will notify all Threads which are waiting on that monitor
- Why 'synchronized' mandatory ?
  - A wait() only makes sense when there is also a notify() from other thread

```
synchronized(lock){
        while(!condition)
        {
            lock.wait();
        }
 }
```

Exercise – QueueMessageBox Only one Receiver may process message

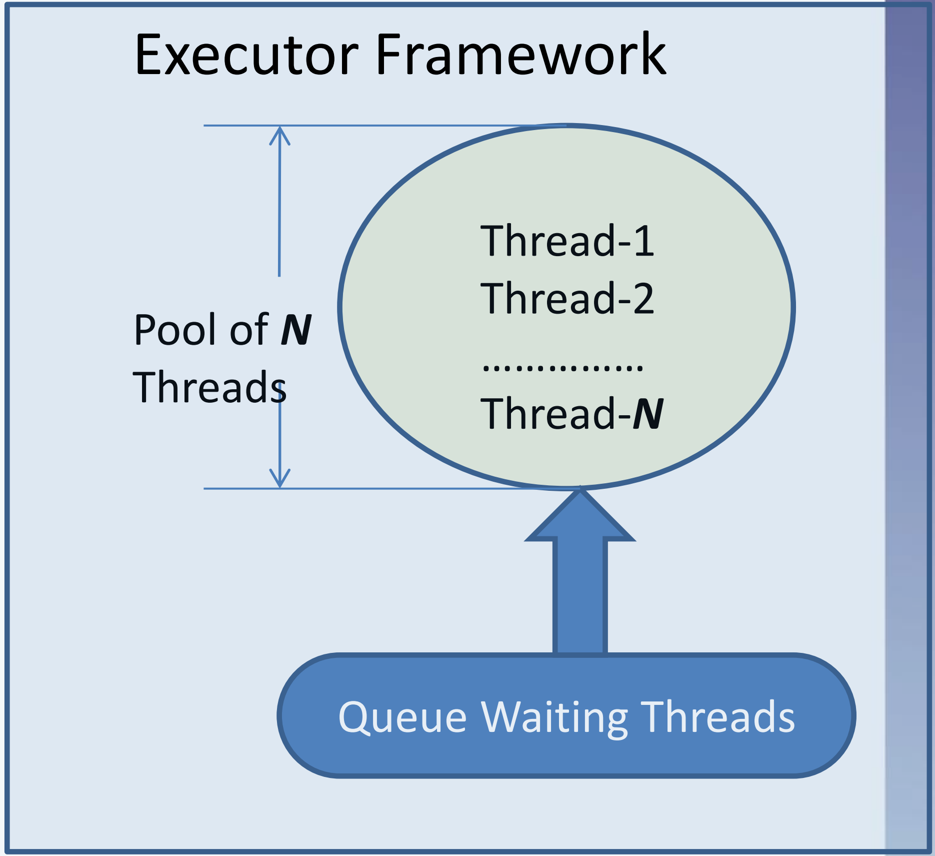
- Write class QueueMessageBox with methods putMessage and getMessage
  - putMessage puts string into a box and sends notification
  - getMessage returns string from a box just in the case if the box contains new message, otherwise it waits
- Write class Sender putting messages into an object of QueueMessageBox class
  - Gets messages from the console
- Write class Receiver getting messages from object of QueueMessageBox class
  - Writing message to a file with thread identifier
- Requirements:
  - No one messages may be lost, even if no one receiver is running

Exercise-TopicMessageBox All Receivers process message

- The same methods as for QueueMessageBox but all receivers that are running should get a message
- Receivers that were started after a sender had sent a message will not get the message

Executor Framework

- Thread pool
  - ***N*** executing threads
- Queue
  - Awaiting threads
- Executor Services
  - Methods of the interface ***ExecutorService***



TEL-RAN

Executor Services

- Create Fixed Thread Pool

  ExecutorService executor = Executors.newFixedThreadPool(<N>);
- Submit runnable for executing

  executor. execute(<Runnable>);
- Stop executor

  executor.shutdown ();
- Waiting for termination of all tasks

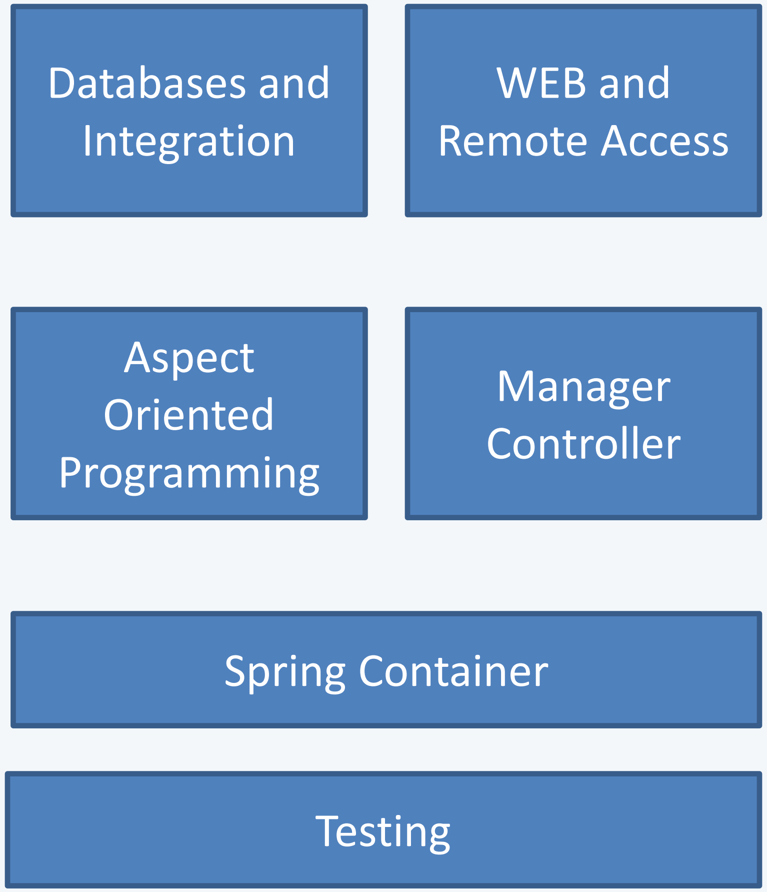
  executor.awaitTermination(<number time units>, <Time Unit>);
  TimeUnit.SECONDS/MILLISECONDS/HOURS/…

Exercise – Messaging Server with Thread Pool

- Update Messaging server with applying Executor Framework
- Optimization of the Messaging Server
  - Imitate 1000 clients
  - Each client gets 1000 messages under TCP
  - Find optimal size of Thread Pool

SPRING

Spring Framework High Level Architecture



- Database and Integration
  - Connection to any Databases
  - Integration with different frameworks, social networks and clouding services
- WEB and Remote Access
  - MVC approach for WEB applications development
  - WEB services
- AOP
  - Aspects applying and development
- Manager
  - Application Context Management and controlling
- Spring Container
  - Registration of the application context components
- Testing
  - Tools for creation and running Application context tests

TEL-RAN

Spring JAR's

- Dropbox Link to the required Spring JAR's
    - https://www.dropbox.com/sh/hnrwffq6q4d75vf/AACmbrUpf4z2Amz5FE3dnSVra
  - Spring-4.0.3.zip – ZIP file containing the Spring Framework JAR's
  - Hibernate-4.2.1.2.zip – ZIP file containing the Hibernate Framework JAR's
  - Jpa-2.zip –ZIP file containing the JPA Framework JAR's
- Installation
  - Create the following folders on the local PC: spring, hibernate, jpa
  - Unzip the above archives into the proper folders

TEL-RAN

Adding Spring JAR’s into Eclipse Projects

- Create Java project (for example, Spring_start)
- Right click on the project->Properties->Java Build Path->Libraries->Add Library->User Library->next
  - If there is not the required user library
    - Press User Libraries button
    - Press New and type user library name (for example, spring) and OK
    - Press Add External JAR’s, select all jars from the folder and open them, press OK and Finish
  - If there is the required user library
    - Select the library

XML File – Basic Terms 1

- (Unicode) character
  - XML document is a string of unicode characters

  <?xml version="1.0" encoding="UTF-8"?>

  <俄语>данные</俄语>
- Processor and application
  - The *processor* analyzes the markup and passes structured information to an *application*.
  - The processor (as the specification calls it) is often referred to as an *XML parser*
- Markup and content
  - *markup*
    - strings that constitute markup begin with the character < and end with >
  - *content*
    - Strings of characters that are not markup are content

XML File – Basic Terms 2

- Tag
  - *start-tags*; for example: <bean>
  - *end-tags*; for example: </bean>
  - *empty-element tags*; for example: <property />
- Element
  - A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.
  - The characters between the start- and end-tags, if any, are the element's *content*, and may contain markup, including other elements, which are called *child elements*
- Attribute
  - A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag

```
<bean id="emf"
    class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="springHibernate" />
</bean>
<bean
```

XML File – XML Schema

- XML Schema Definition XSD
  - File with the extension .xsd containing description of the markup's, elements, attributes and values
  - Attribute xmlns defines a prefix name and a name space of the schema
  - Attribute xsi:schemaLocation defines the location of the XSD files for each schema's namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:tx="http://www.springframework.org/schema/tx"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd ">
```

XML File Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:tx="http://www.springframework.org/schema/tx"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans spring-beans-4.0.xsd
  http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd ">
<bean id="emf“ class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="springHibernate" />
</bean>
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
<bean id="person" class="tel_ran.spring.db.PersonDB"/>
<tx:annotation-driven transaction-manager="transactionManager"/>
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="emf"/>
</bean>
  </beans>
```

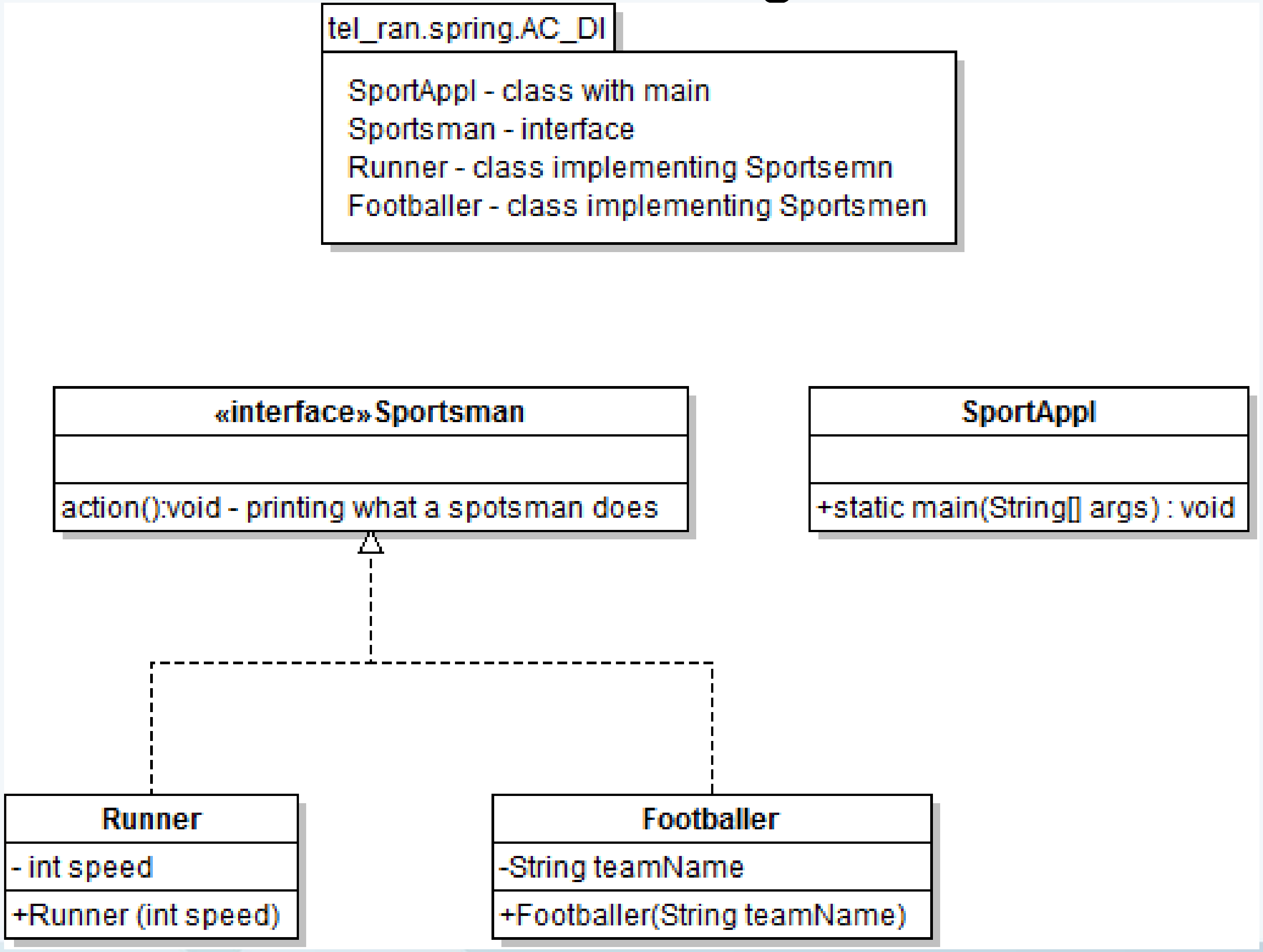
Application Context and Dependency Injection

- Application Context
  - XML file containing all application's components (beans)
  - Connection between Java class and AC component
  - Creation

    AbstractApplicationContext ctx = new FileSystemXmlApplicationContext("beans1.xml");
  - Closing

    ctx.close();
- Dependency Injection (DI)
  - Creation and configuration of the objects based on the interfaces and XML file configuration

Example – AC and DI
UML Class Diagram


tel_ran.spring.AC_DI
SportAppl - class with main
Sportsman - interface
Runner - class implementing Sportsemn
Footballer - class implementing Sportsmen
«interface» Sportsman
action():void - printing what a spotsman does
SportAppl
+static main(String[] args) : void
Runner
- int speed
+Runner (int speed)
Footballer
-String teamName
+Footballer(String teamName)

Example – AC and DI Function ***main***

```
package tel_ran.spring.AC_DI;
import org.springframework.beans.BeansException;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class Appl {
public static void main(String[] args) throws BeansException  {
AbstractApplicationContext ctx = new
FileSystemXmlApplicationContext(“beans1.xml");
Sportsman spotsman=(Sportsman)ctx.getBean(“sporttsman”);
spotsman.action();
ctx.close();
}
}
```

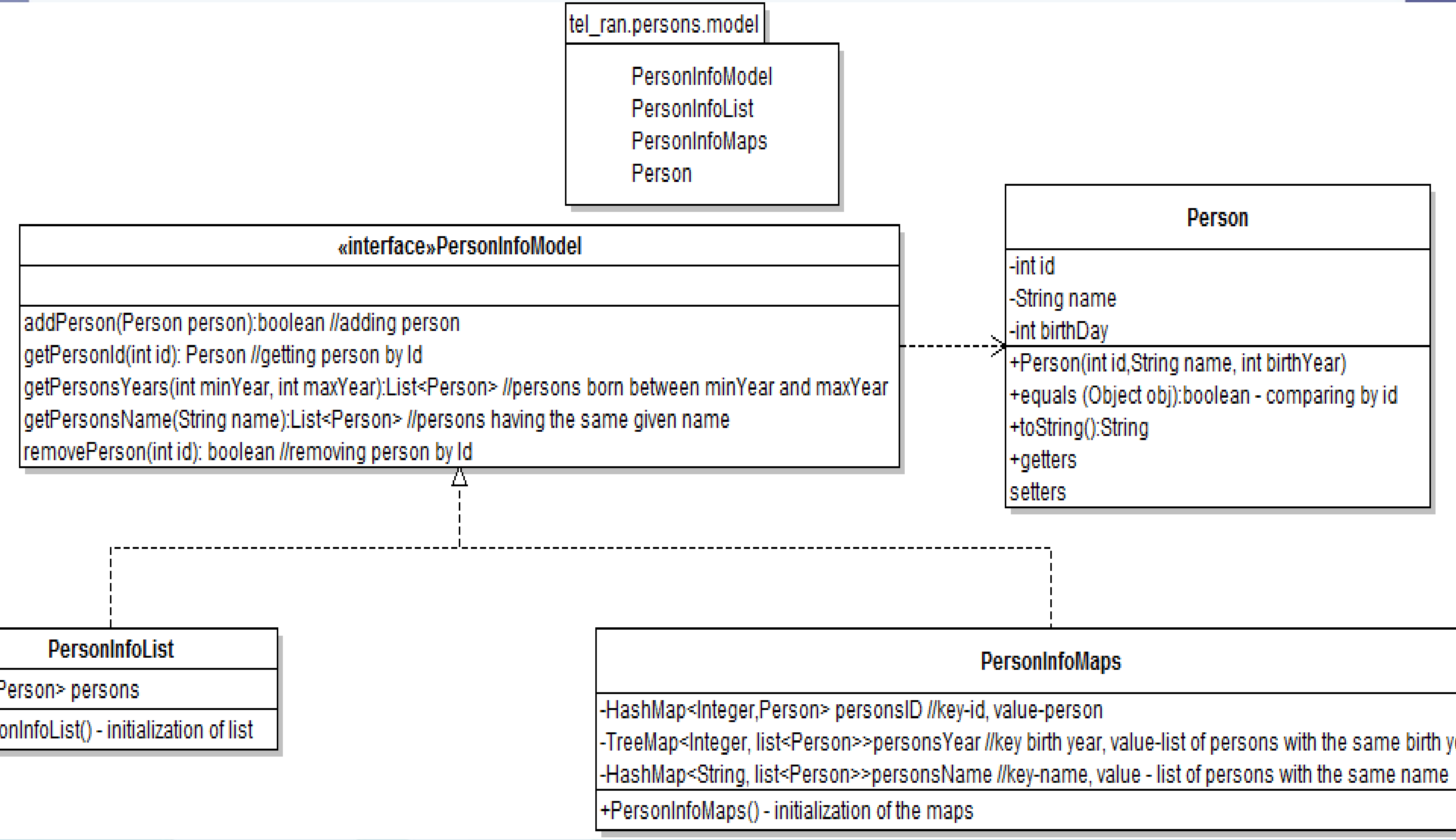
Example – AC and DI XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd ">
 <bean id="sportsman" class = "tel_ran.spring.AC_DI.Runner" >
  <constructor-arg><value>50</value></constructor-arg>
  </bean>
  </beans>
```

Exercise -Spring Person Application

- Implement Person application running under Spring framework
- Classes implementing interfaces ***PersonInfoModel, PersonInfoView*** and ***Requester*** should be created as Spring components-beans
- Diagrams for Model, View and Controller are presented on the next slides
- File containing the requests should be placed in the project directory

Person Application Model



«interface»PersonInfoModel

```
addPerson(Person person):boolean //adding person
getPersonId(int id): Person //getting person by Id
getPersonsYears(int minYear, int maxYear):List<Person> //persons born between minYear and maxYear
getPersonsName(String name):List<Person> //persons having the same given name
removePerson(int id): boolean //removing person by Id
```

Person

```
-int id
-String name
-int birthDay
+Person(int id,String name, int birthYear)
+equals (Object obj):boolean - comparing by id
+toString():String
+getters
setters
```

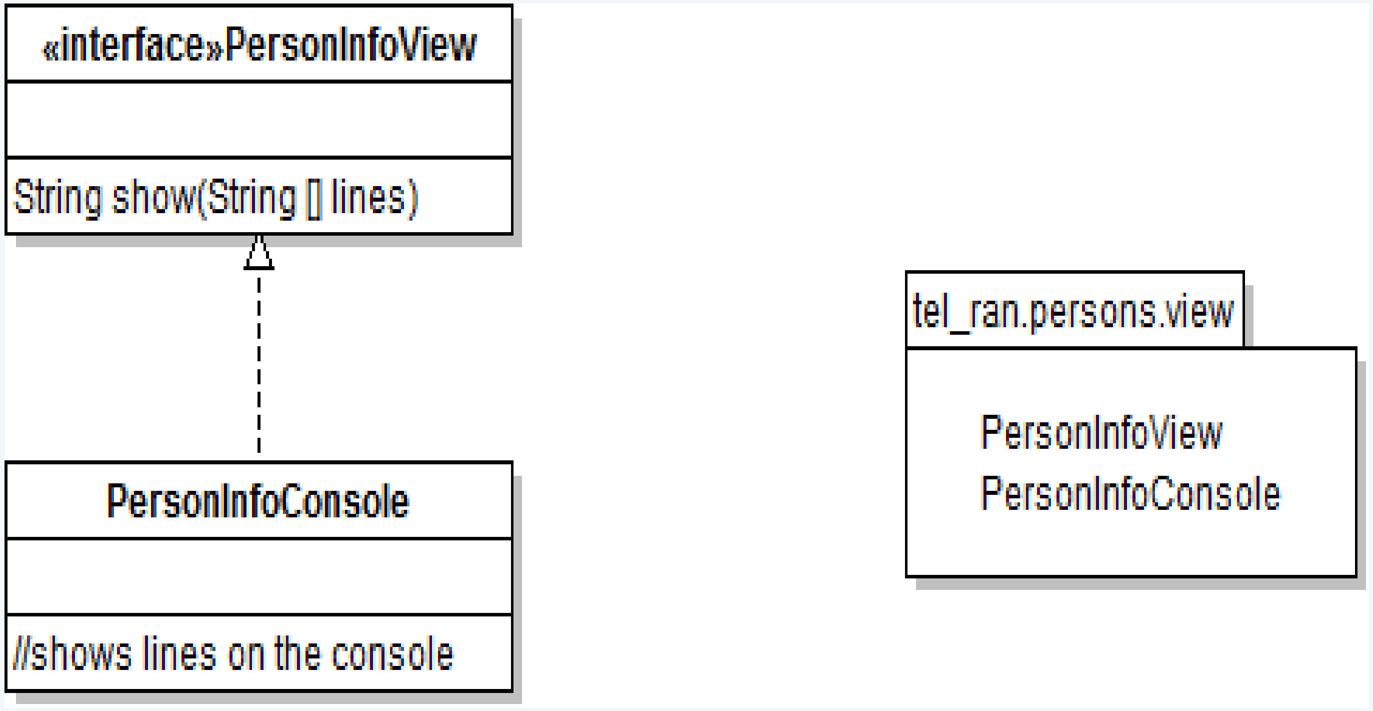
PersonInfoList

```
erson> persons
onInfoList() - initialization of list
```

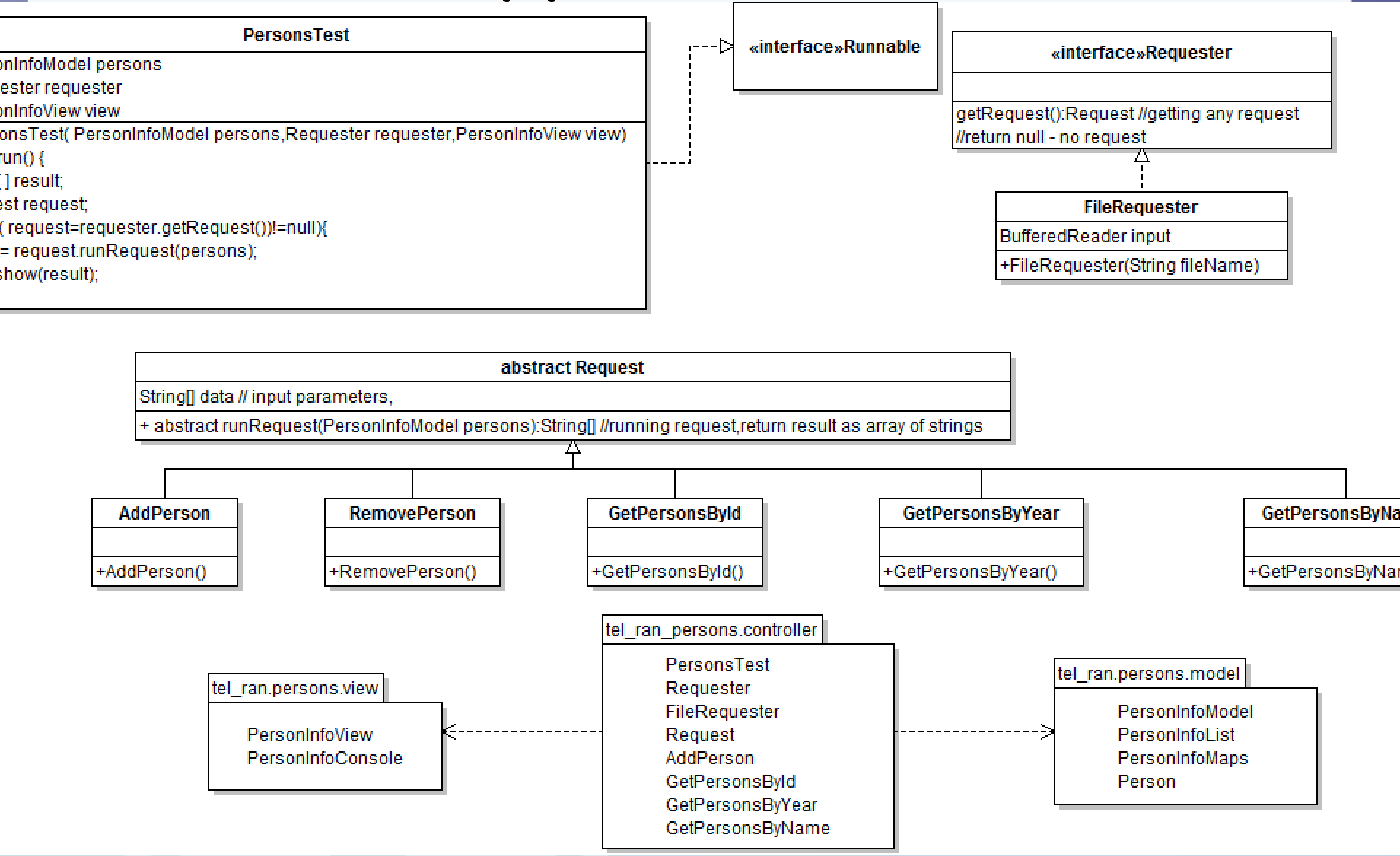
PersonInfoMaps

```
-HashMap<Integer,Person> personsID //key-id, value-person
-TreeMap<Integer, list<Person>>personsYear //key birth year, value-list of persons with the same birth y
-HashMap<String, list<Person>>personsName //key-name, value - list of persons with the same name
+PersonInfoMaps() - initialization of the maps
```

Person Application View


«interface»PersonInfoView
String show(String [] lines)
PersonInfoConsole
//shows lines on the console
tel_ran.persons.view
PersonInfoView
PersonInfoConsole

Person Application Controller

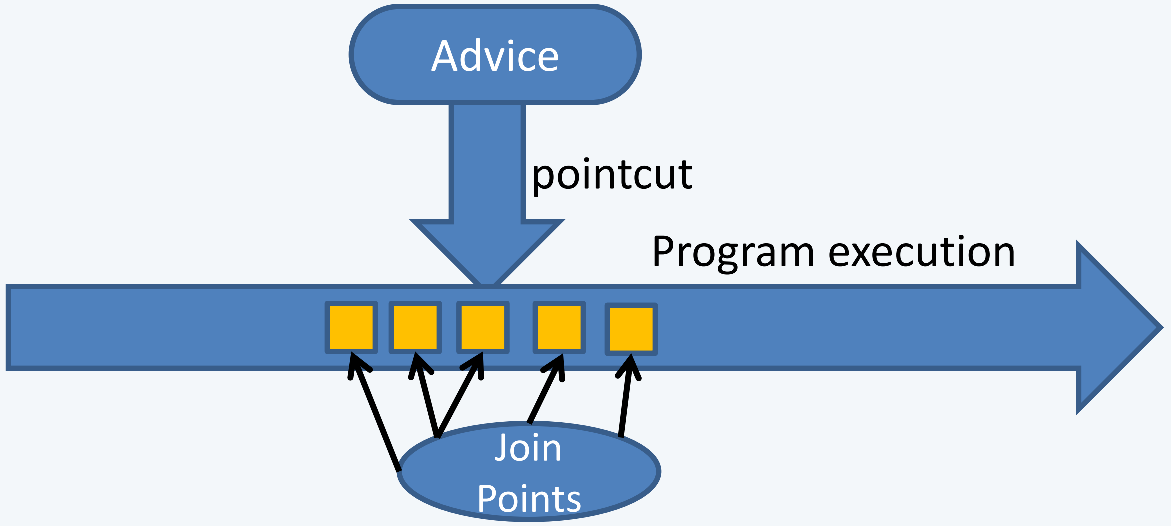


Aspect Oriented Programming

- AOP is a programming paradigm, like OOP
- OOP fails in the modularization of cross-cutting aspects
  - Security
  - Tracing
  - Benchmarking
  - Validation
  - Others
- AOP completes OOP in the cross-cutting aspects

AOP Concepts

- Join Point
  - *Point during the execution of a program, such as the execution of a method or the handling of an exception*
- *Advice*
  - Action taken by an aspect at a particular join point
- *Pointcut*
  - Predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut



Advice Types

- Before advice
  - Advice that executes before a join point
- After returning advice
  - Advice to be executed after a join point completes normally
- After throwing advice
  - Advice to be executed if a method exits by throwing an exception
- **Around advice**
  - **Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.**

Pointcuts Expression Examples

- Execution of any public method:

  execution(public * *(..))
- Execution of any method with a name beginning with "set":

  execution(* set*(..))
- Execution of any method defined by the AccountService interface:

  execution(* com.xyz.service.AccountService.*(..))
- Execution of any method defined in the service package:

  execution(* com.xyz.service.*.*(..))
- Execution of any method defined in the service package or a sub-package:

  execution(* com.xyz.service..*.*(..))

AOP XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:aop="http://www.springframework.org/schema/aop"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/sche
ma/beans
http://www.springframework.org/schema/beans/spring-
beans-4.0.xsd
   http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-
4.0.xsd">
```

TEL-RAN

AOP XML Example

```
<bean id="bench" class="tel_ran.spring.aop.Bench">
</bean>
<aop:aspectj-autoproxy></aop:aspectj-autoproxy>
<aop:config>
<aop:aspect ref="bench">
<aop:pointcut id="timing" expression="execution(public *
*(..))" />
<aop:around pointcut-ref="timing"
method="watchPerformance"/>
</aop:aspect>
</aop:config>
  </beans>
```

Example of the Around Advice

```
public class Bench {
public void watchPerformance(ProceedingJoinPoint point) throws
Throwable{
Signature sign=point.getSignature();
String name=sign.getName();
System.out.println("method invoked is "+name);
long time1=System.currentTimeMillis();
point.proceed();
long time2=System.currentTimeMillis();
System.out.println("time of running method "+name+" is "+(time2-
time1));
}
}
```

ProceedingJoinPoint Class

proceed()

Proceed with the next advice or target method invocation

Signature getSignature()

Returns the signature at the join

Object[] getArgs()

Returns the arguments at this join point.

Object getTarget()

Returns the reference to a target object

Exercise – Defining Time Aspect

- Develop Spring aspect allowing defining time of running test in the Person Application
- Create file for class FileRequester containing 1000 random requests for adding persons, 1000 random requests for getting persons by name, 1000 random requests for getting persons by Id, 1000 random requests for getting persons by year
- Create object of the class PersonsTest as a Spring component-bean
- Measure time of the running test for different classes implementing PersonsInfo interface

JDBC Overview

- Loading MySql JDBC driver to JVM
  - Class Reflection

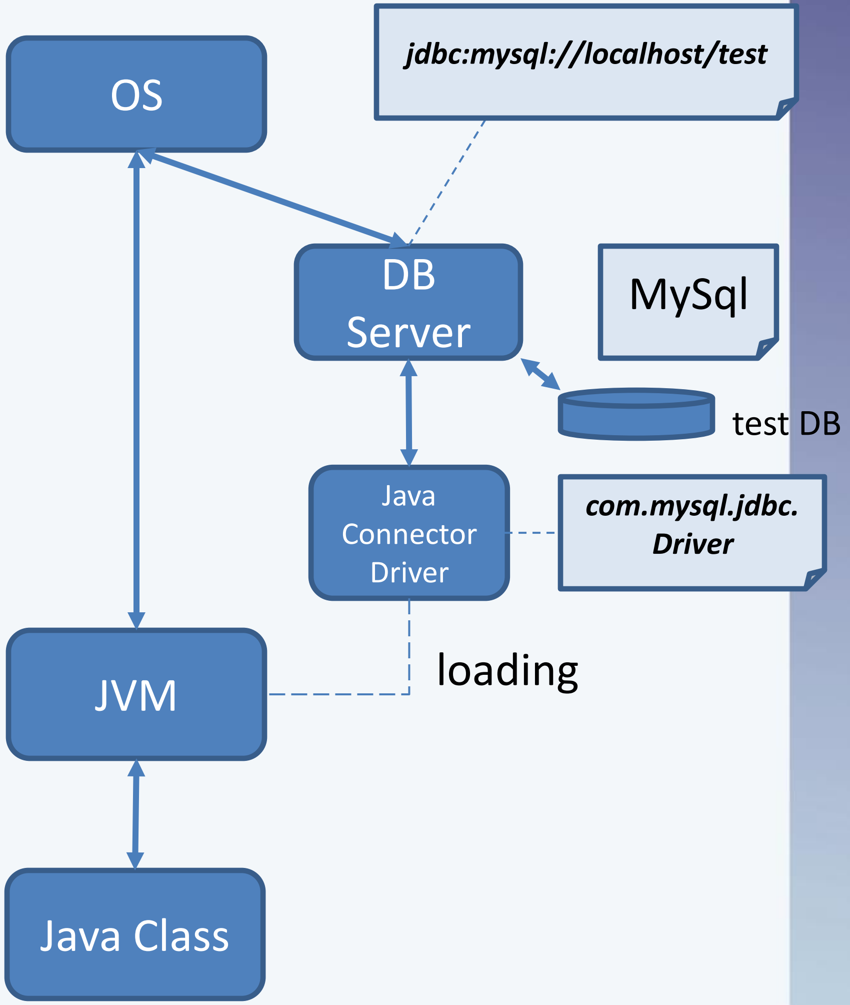
Class.forName ("com.mysql.jdbc.Driver");

- URL string for local MySql server

String url = "jdbc:mysql://localhost/test";

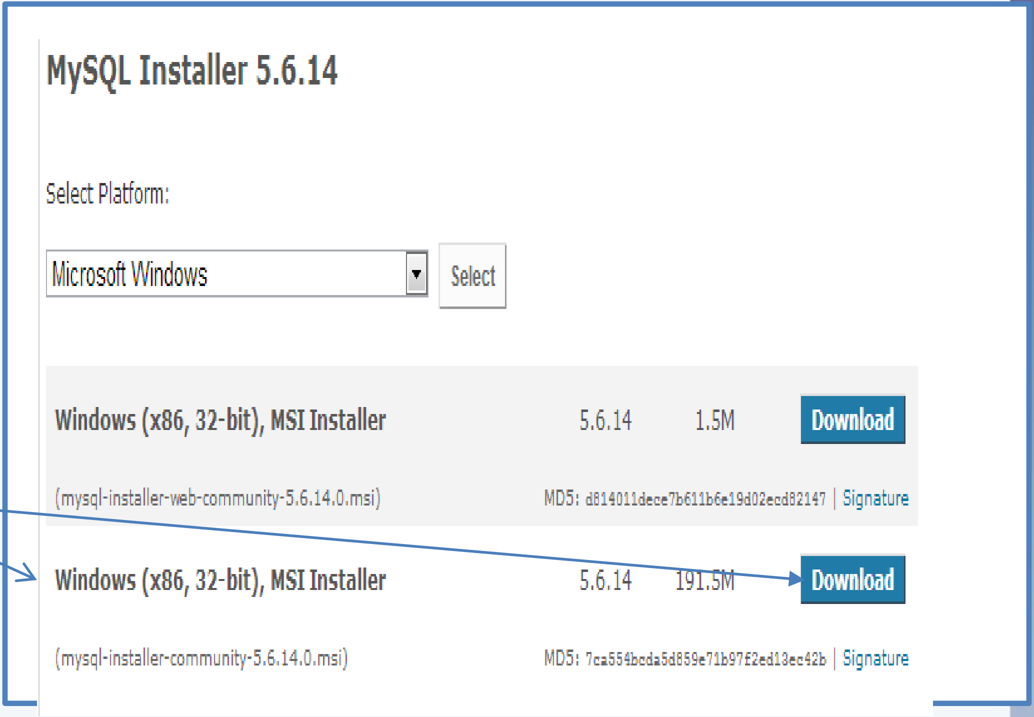
- Getting Connection

conn = DriverManager.getConnection (url, userName, password);



MySQL Installation

- Access site http://dev.mysql.com/downloads/windows/installer/#downloads
- Scroll down until the picture
  - Download 191.5M installer
  - Run installer and follow the instructions (Don't forget the root password you should enter during installation)

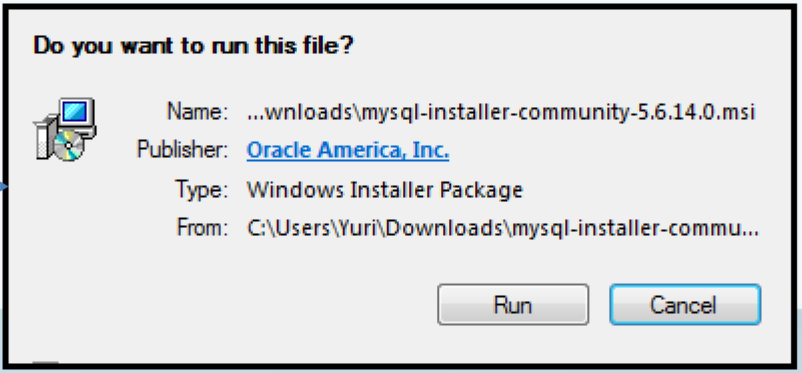


MySQL Installer 5.6.14

Select Platform:

Microsoft Windows | Select

Windows (x86, 32-bit), MSI Installer	5.6.14	1.5M	Download
(mysql-installer-web-community-5.6.14.0.msi)		MD5: d814011dece7b611b6e19d02ecd82147 | Signature	
Windows (x86, 32-bit), MSI Installer	5.6.14	191.5M	Download
(mysql-installer-community-5.6.14.0.msi)		MD5: 7ca554bcda5d859e71b97f2ed13ec42b | Signature	



Do you want to run this file?

Name: ...wnloads\mysql-installer-community-5.6.14.0.msi
Publisher: Oracle America, Inc.
Type: Windows Installer Package
From: C:\Users\Yuri\Downloads\mysql-installer-commu...

JDBC MySQL Adding to Java Project

- Adding MySQL connector jar to the build path of the project

C:\Program Files (x86)\MySQL\MySQL Connector J\mysql-connector-java-5.1.26-bin

  - Adding External JAR
    - The path to the external JAR (can’t be transferred out the local machine)

Database Connection

Singleton class

```
public class DatabaseConnection {
		private static final String DRIVER_NAME = "com.mysql.jdbc.Driver";
		private static final String URL_DB_TEST = "jdbc:mysql://localhost/test";
		static DatabaseConnection dbc=null;

		Statement sqlSt;
		private DatabaseConnection(String userName,String password)
throws ClassNotFoundException, SQLException{

		Class.forName(DRIVER_NAME);
		Connection cnn=DriverManager.getConnection(URL_DB_TEST,userName,password);
		sqlSt=cnn.createStatement();
  }
static public Statement getDatabaseConnection
(String userName,String password) throws ClassNotFoundException, SQLException{
		if(dbc==null){
		dbc=new DatabaseConnection(userName, password);
		}
		return dbc.sqlSt; //returns object of the class Statement intended to run any
					SQL statement
  }
}
```

Class Statement has two main methods:

executeUpdate(String sql) and ***executeQuery(String sql)***

Table

- Table is an main aggregated information unit into Relative Databases
- Create Table

```
 String sql="CREATE TABLE IF NOT EXISTS Person (id INTEGER," +
"name VARCHAR(254), birthYear INTEGER, PRIMARY KEY(id)," +
"INDEX(name), INDEX(birthYear))";
statement.executeUpdate(sql);
```

Column

Row

123457	Petya	1960
123458	Vasya	1953
123459	Moshe	1948
123460	Petya	1985

Insert Row

SQL Statement syntax

INSERT INTO <table name> (<column name>,…) VALUES (<value>,…),[(<value>,…)…]

- – Number of the values inside brackets shall be equal the number of columns (<column name>,…)
- – String value should be in ordinary quotes ‘<value>’

- Example

```
String sql="INSERT INTO person (id,name,birthYear) "
        + "VALUES ("+id+",'"+name+"',"+birthYear+')';
```

Select Query

- SELECT very lite syntax

SELECT *|<column name>, … FROM <table name> [WHERE <column name> <condition> <value> [OR|AND <column name> <condition> <value>]] …

- Example

String sql="SELECT * FROM Person WHERE birthYear="+ birthYear

Result Set

```
ResultSet rs = statement.executeQuery(sql)
while (rs.next()) {
            String name=rs.getString("name");
            int id=rs.getInt("id");
            int age=rs.getInt("age");
            }
```

Delete row

- Syntax

DELETE FROM <table name> [WHERE <condition statement>]

  - If WHERE clause is not specified, then all the records will be deleted from the given MySQL table.
  - Any condition may be specified using WHERE clause.

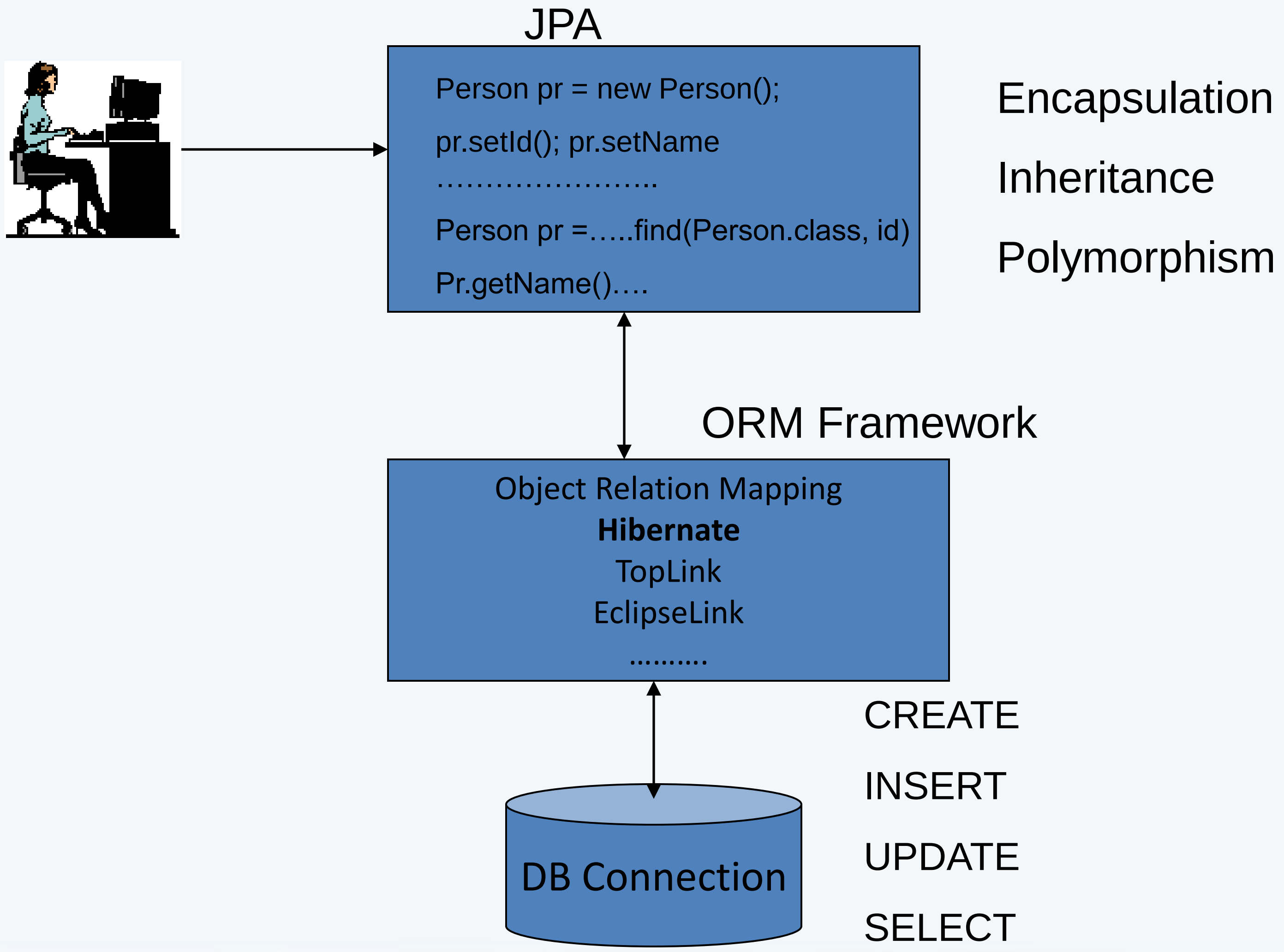
- Example

```
String sql="DELETE FROM person WHERE id="+id;
```

Exercise - PersonInfoSQL

- Develop class PersonInfoSQL based on JDBC requests, implementing interface PersonInfoModel
- Run test and measure time for PersonInfoSQL object

JPA – Java Persistence API



Entity, Entity Manager and Persistence Unit

- Entity
  - POJO (Plain Old Java Object) regular Java class with annotation @Entity
  - Annotations for defining specific fields as primary key and relations with other entities
- Entity Manager
  - JPA component for transform operations between Entity and Database tables (Object Relation Management)
- Persistence Unit
  - Unit containing all required data for connection between Entity Manager and Java application

Entity Example

```
@Entity
public class Person {
        @Id
        private long id;
        private String name;
        public long getId() {
                return id;
        }
        public void setId(long id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }


}
```

@Entity – says that the POJO is entity

@Id – says that long id is a primary key

Just two annotations @Entity and @Id should be added for Entity Manager functionality

Transactions – ACID

- Atomicity
  - A transaction is a group of operations a a single unit. One operation fails – whole transaction fails
- Consistency
  - Data stays consistent in terms of schema constraints once transactions ends
- Isolation
  - Data are not viewed by other processes while transaction is not ended successfully (DBMS isolation level)
- Durability
  - Once Transaction ended successfully, the committed data should be available

Spring Managed JPA

- Spring is responsible for
  - Opening and closing of the entity manager
  - Transactions boundaries
  - Database connection management under Persistence Unit
    - Creation – Removal
    - Injection to a bean

Persistence XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd ">
………………………………………………
</persistence>
```

Basic persistence parameters

```
<persistence-unit name=" springHibernate ">
```

Relation between persistence unit and Spring application

```
<provider>org.hibernate.ejb.HibernatePersistence</provider>
```

Provider implementing ORM platform

```
<property name="hibernate.connection.url"
value="jdbc:mysql://localhost/test"/>
<property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
<property name="hibernate.connection.username" value="root"/>
<property name="hibernate.connection.password" value="12345.com"/>
```

Properties defining Database connection

```
<property name="hibernate.hbm2ddl.auto" value="create"/>
```

Table policy: “create” – always create table each time the bean with persistence unit getting started; “create-drop” – always create table and drop after PC removal; “update” – using existing table

Persistence XML Content Example

<Project folder>\src\META-INF\persistence.xml

```
<persistence-unit name="springHibernate">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<properties>
<property name="hibernate.hbm2ddl.auto" value="update"/>
<property name="hibernate.connection.url"
value="jdbc:mysql://localhost/test"/>
<property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
<property name="hibernate.connection.username" value="root"/>
<property name="hibernate.connection.password" value="12345.com"/>
</properties>
 </persistence-unit>
```

Saving Entity Object

- The same operator new for creating POJO presenting Entity

```
Person pr = new Person();
```

- The regular set operations

```
pr.setId(id);
pr.setName(name);
```

- Saving to Persistence Context through Entity Manager

```
em.persist(pr);
```

Finding Entity Object

- Find Entity from the DB and put it inside PC

em.find(Person.class, id), where id is the primary key of the entity

- Method find returns reference to POJO representing Entity inside PC

Person pr = em.find(Person.class, id);

- Having the reference you may do everything Note: any setter will update Entity inside PC.

Java Persistence Query Language

- Querying objects
  - Very similar to SQL but in terms of objects and object fields
- Simple examples
  - retrieving from DB object/s of the class/entity “Person” with “age” field value 80

```
 Query q = em.createQuery("Select p FROM Person p WHERE p.age =
?1");
 q=q.setParameter(1, 80);
 List<Person>res= q.getResultList();
```

  - retrieving from DB all objects of the class/entity Person

```
   Query qr = em.createQuery ("select p from Person p");
   List<Employee2>employees = qr.getResultList();
```

Spring XML Schema for JPA

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:tx="http://www.springframework.org/schema/tx"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-4.0.xs
  http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd ">
```

Spring XML File for JPA & Hibernate

```
<bean id="emf"
  class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="springHibernate" />
</bean>
<bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostP
rocessor"
/>
<bean id="person" class="tel_ran.spring.db.PersonDB"/>
<tx:annotation-driven transaction-manager="transactionManager"/>
<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="emf"/>
</bean>
```

Transaction Creation

- Transaction should be created for any Database update
- Before function updating Database there should be the following annotation

```
@Transactional(readOnly = false,
propagation = Propagation.REQUIRES_NEW)
```

Transaction Attributes

Transaction Attribute	Client Transaction TC	Method Transaction TM
REQUIRED	None TC	TM TC
REQUIRES_NEW	None TC	TM TM
MANDATORY	None TC	Error TC
SUPPORTS	None TC	None TC
NOT_SUPPORTED	None TC	None None
NEVER	None TC	None Error

Transactional Example

- Transaction is mandatory in the case of Database update

```
@Transactional(readOnly = false,
propagation = Propagation.REQUIRES_NEW)
@Override
public boolean addPerson(Person prs) {
boolean res=false;
if(em.find(Person.class, prs.getId())==null){
em.persist(prs);
res=true;
}
return res;
}
```

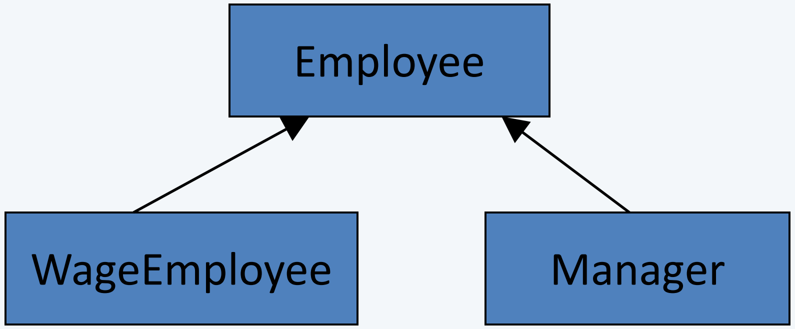
Method addPerson will create new transaction

Exercise – PersonInfoHibernate

- Update class Person for Entity class
- Develop class PersonInfoHibernate based on Hibernate Persistence Unit
- Run test and measure time for PersonInfoHibernate object

Inheritance & Polymorphism

public abstract double computePay();



```
@Entity
public abstract class Employee {
@Id
private long employeeId;
private String name;
private double basicSalary;
..................
```

```
@Entity
public class Manager extends
Employee{
private int coeff;
public double computePay() {
return getBasicSalary()*coeff;
}
........................
```

```
@Entity
public class WageEmployee
extends Employee {
private double hours;
private double wage;
public double computePay()
{
return getBasicSalary()+
hours*wage;
}
........................
```

What Hibernate Allows !!!

```
public double comuteBudget(){
        Query q=em.createQuery("SELECT e FROM Employee e ");
        List<Employee> employees=q.getResultList();
        double budget=0
       for(Employee empl: employees)
                  budget+=empl.computePay();
}
```

Working with only class Employee with no knowing about specific employee types (WageEmployee, Manger, SalesPerson, etc.)

Little bit more about SELECT

<field>:=<object name>|<aggregate function>|<object field>

- Format 1

SELECT <field> FROM <from clause>

***List<Object>** list=query.getResultList();*

- Format 2

SELECT <field>,<field> FROM <from clause>

***List<Object[]>** list = query.getResultList();*

TEL-RAN

Entity Fields

- Embedded/Embeddable
  - Reference to a user Class object as persistent field, where user class is not entity
- java.util.Date and java.util.Calendar types should be annotated with the @Temporal annotation
- Reference to another Entity should be annotated with one from the following annotations
  - @OneToOne
  - @OneToMany
  - @ManyToOne
- Unmapped field should be annotated with @Transient annotation

Embeddable Example

```
@Embeddable // marking the class as Embeddable
public class Size implements Serializable{
	private int width;
	private int height;
	public Size() { // required no-args constructor
			}
	public Size( int width, int height) {
			this.height = height;
			this.width = width;
	}
	public int getWidth() {return width;}
	public void setWidth(int width) {this.width = width;}
	public int getHeight() {return height;}
	public void setHeight(int height) {this.height =
							height;}
}
```

Embedded Example

```
@Entity
public class Item {
      private String name;
@Id
      private int id;
       @Embedded
      private Size size;
      public Item() {}
      public Item(String name) {this.name = name;}
      public int getId() {return id;}
      public void setId(int id) {this.id = id;}
      public Size getSize() {return size;}
      public void setSize(Size size) {this.size = size;}
      public String getName() {return name;}
      public void setName(String name) {this.name = name;}
}
```

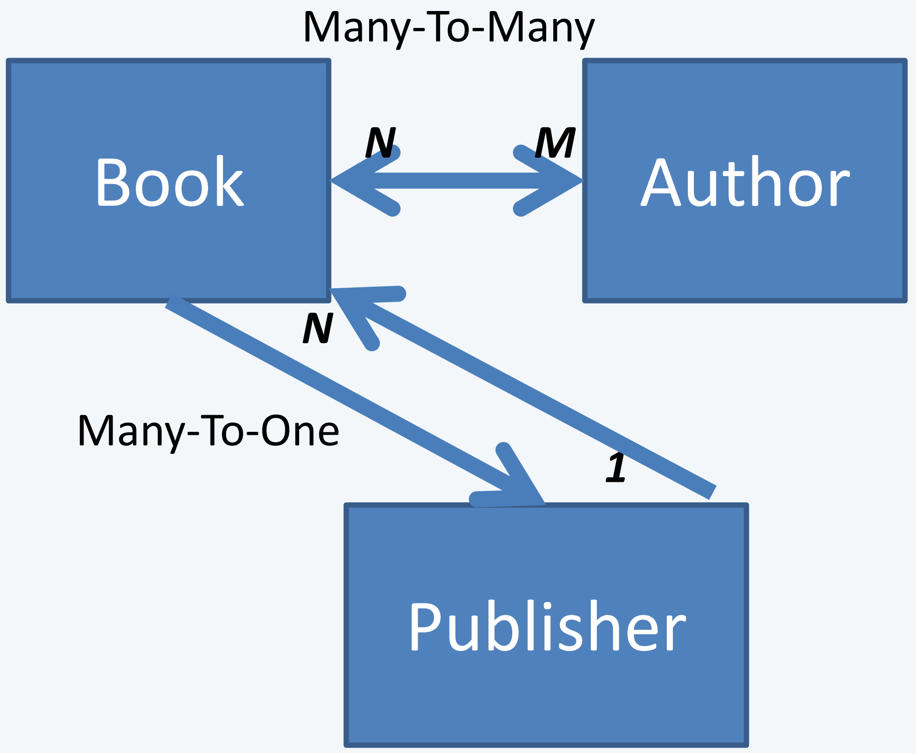
One-To-One Example

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String name;
    @OneToOne

    private Address address;

    public int getId() {return id;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    public Address getAddress() {return address;}
    public void setAddress(Address address) {this.address =
    address;}
}
```

Relations between Entities



Hibernate performs mapping automatically according to the relation annotation attribute ***mappedBy.*** Value of this attribute defines what is mapped by what (Which entity should be created before)

Authors and Publishers should be mapped by Book

One-To-Many/Many-To-One Example

```
@Entity
public class Publisher {
    @Id
    String publName;
    @OneToMany
    (mappedBy="publisher")
    List<Book> books;
    //Regular getters and
    setters
    }
```

```
@Entity
public class Book {
    @Id
    String title;
    @ManyToOne
    Publisher publisher;
    @ManyToMany
    List<Author> authors;
    //Regular getters and
    setters
    }
```

Many-To-Many Example

```
@Entity
public class Author {
    @Id
    String name;
    @ManyToMany
    (mappedBy="authors")
    List<Book> books;
    public String getName() {
        return name;
    }
```

```
@Entity
public class Book {
    @Id
    String title;
    @ManyToOne
    Publisher publisher;
    @ManyToMany
    List<Author> authors;
    //Regular getters and
    setters
    }
```

Adding Book

```
 public void createBook(String title, String[] authorNames, String publName)
{
List<Author> authors = new ArrayList<Author>();
for(String name: authorNames){
       Author author = new Author();
        author.setName(name);
       if(em.find(Author.class, name)==null)
              em.persist(author);
       authors.add(author);
}
Publisher publisher=new Publisher();
publisher.setPublName(publName);
if(em.find(Publisher.class, publName)==null)
    em.persist(publisher);
Book book=new Book();
book.setTitle(title);
book.setPublisher(publisher);
book.setAuthors(authors);
Book bookOld=em.find(Book.class, title);
if(bookOld != null)
     em.remove(bookOld);
em.persist(book);
              }
```

Possible Simple Queries

- Getting all books written by author ‘author1’

"SELECT b FROM Book b JOIN b.authors a WHERE a.name=‘author1'“

- Getting all authors who wrote book ‘book1’

"SELECT a FROM Author a JOIN a.books b WHERE b.title=‘book1'“

- Getting all books published by publisher “publisher28”

"SELECT b FROM Book b JOIN b.publisher p WHERE p.publName='publisher28'“

- Getting publisher that published book ‘book1’

"SELECT p FROM Publisher p JOIN p.books b WHERE b.title='book1'“

Possible Complex Queries

- Get titles of all books written by two authors

"SELECT b.title FROM Book b JOIN b.authors a GROUP BY b.title HAVING COUNT(b.title)=2“

- Getting all books written by author10 and author11 together

"SELECT b FROM Book b JOIN b.authors a WHERE a.name = 'author10' AND EXISTS (SELECT b1 FROM Book b1 JOIN b1.authors a1 WHERE a1.name = 'author11‘ AND b.title=b1.title)";

Possible Complex Queries

- Getting authors who wrote books “book181” and “book192”

"SELECT a FROM Author a JOIN a.books b WHERE b.title='book181‘ AND EXISTS (SELECT a1 FROM Author a1 JOIN a1.books b1 WHERE b1.title='book192‘ AND a.name=a1.name)“

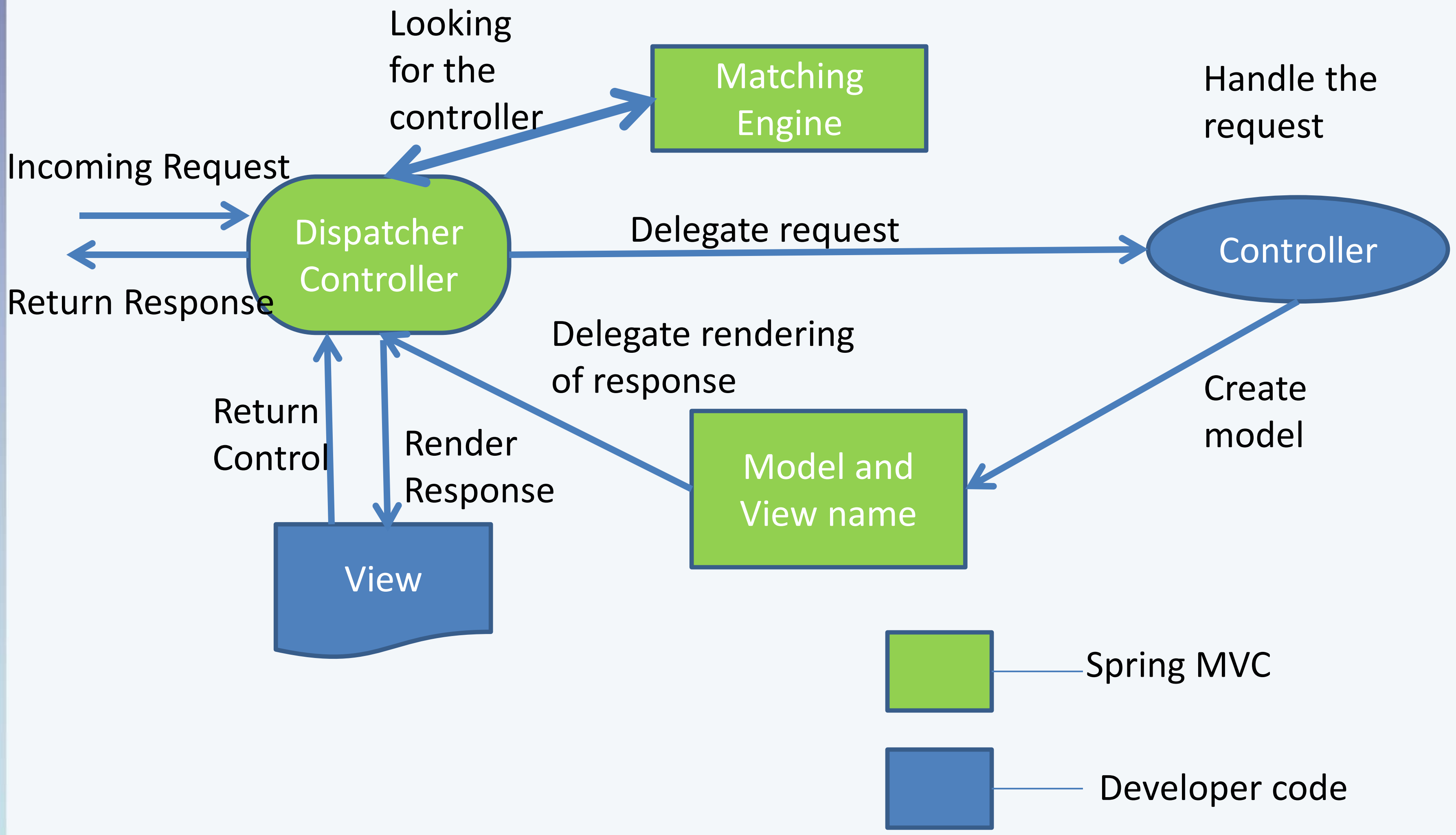
- Getting author names who wrote more than two books

"SELECT a.name FROM Author a JOIN a.books b GROUP BY a.name HAVING COUNT(a.name)>2"

WEB Server Tomcat Installation

- Download apache-tomcat-7.0.53-windows-x86.zip from dropbox
- Extract in some folder of the local PC
- Enter C:\<apache root directory>\apache-tomcat-7.0.53\bin
  - Send windows batch file startup to desktop for creating shortcut (This shortcut nay be used for the Tomcat starting)

Spring MVC


Looking for the controller
Matching Engine
Handle the request
Incoming Request
Dispatcher Controller
Delegate request
Controller
Return Response
Delegate rendering of response
Create model
Return Control
Render Response
Model and View name
View
Spring MVC
Developer code

WEB.xml File Schema

All required information for Spring MVC application controller

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">


</web-app>
```

Example of WEB.xml file Servlet Element

hello – prefix of the Spring initial Application Context XML file. It means that the Spring application context XML file shall have name ***hello-servlet.xml***

org.springframework.web.servlet.DispatcherServlet – Spring class that is the first front-end Spring controller

```
<servlet>
  <servlet-name>hello</servlet-name>
  <servlet-class>
org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>
```

Listener and Context Elements

For adding additional Spring XML files to the application context

```
 <listener>
   <listener-class>
org.springframework.web.context.ContextLoaderListener
   </listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/beans1.xml</param-value>
</context-param>
```

Servlet-mapping Element

Matching of URL request and Spring Application Context initial XML file
hello* –** name of the XML file ***hello-servlet.xml

/* -** means that ***hello-servlet.xml file will be the initial Spring MVC application file for any application WEB request

```
<servlet-mapping>
 <servlet-name>hello</servlet-name>
 <url-pattern>/</url-pattern>
 </servlet-mapping>
```

Initial XML File for Spring MVC Application- XML Schema

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
   http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">


</beans>
```

Initial XML File for Spring MVC Application- Content Example

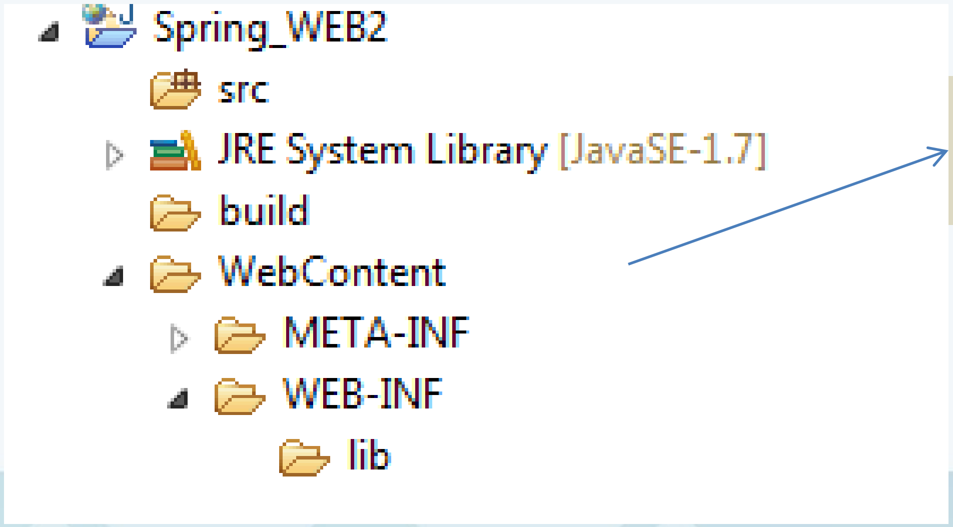
tel_ran.spring.web-Application package where there is initial controller (instead method ***main***

"prefix" – place where there are view application files
"suffix" – extension of the view application files

```
<context:component-scan base-package="tel_ran.spring.web" />
  <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/" />
      <property name="suffix" value=".jsp" />
   </bean>
```

Dynamic WEB Project

- Create dynamic WEB project (Eclipse should be either for JEE development core or with JEE plug-in)
  - ***new->project->WEB->Dynamic WEB Project***
  - fill project name and ***finish***



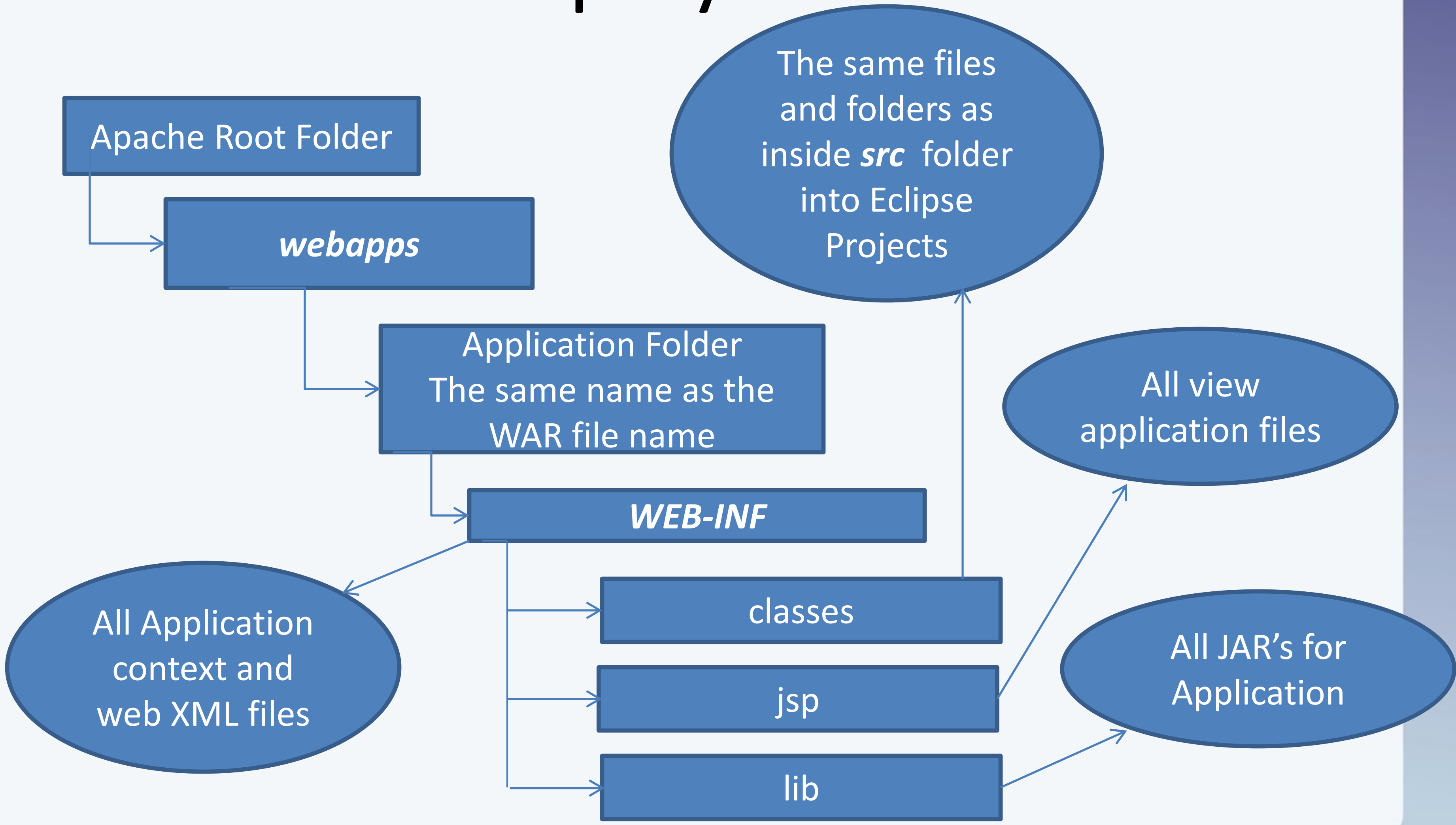
```
Spring_WEB2
    src
    JRE System Library [JavaSE-1.7]
    build
    WebContent
        META-INF
        WEB-INF
            lib
```

The WEB Project structure

Spring MVC Application Deployment on Tomcat

- All classes under folder ***src*** of the Dynamic WEB project
- All additional project JAR's under WebContent/WEB-INF/lib
- Deployment of an WAR file on an Web Server
  - WAR (Web Archive File) file containing full information for the deployment on an Web Server
  - Export->WEB->WAR file->inside <Tomcat root>***webapps*** folder
  - Launch Tomcat
    - After launching Tomcat in the folder ***webapps*** there should be new folder with the same name as the name of WAR file

Application Structure after Deployment


Apache Root Folder
webapps
Application Folder The same name as the WAR file name
WEB-INF
classes
jsp
lib
The same files and folders as inside ***src*** folder into Eclipse Projects
All view application files
All JAR's for Application
All Application context and web XML files

TEL-RAN

Spring Controller

- Allows launching of an application inside WEB server

  <servlet-class>
  org.springframework.web.servlet.DispatcherServlet
  </servlet-class>

  – Forwards to the application controller (like method ***main***)

  <context:component-scan base-package = "tel_ran.spring.web" /> **-** the package where there is application controller

  – Application Controller

    - Class with annotation @Controller

Controller Example

```
@Controller
@RequestMapping({"/“})
public class myController {
        @Autowired
        private ApplicationContext ctx;
        @RequestMapping({"/“})
        public String mainMethod()throws Exception{
        Sportsman sportsman=(Sportsman)ctx.getBean(“sportsman”);
        sportsman.action();
        return null;
        }
```

Model – View – Controller Implementation

- Spring Controller (DispatcherController) forwards control to an application controller and involves the proper controller method according to the matching patterns
  - Each application controller should have @RequestMapping annotation with mapping rules
    - Each public method of the controller should have @RequestMapping annotation with mapping rules
- Spring Model is similar to a map where a key is an attribute name and value presents any Java object
  - The attribute name is the one that may exist in the Spring view as “${<attribute name>}”
- Spring View is an JSP (Java Server Package) file

Application Controller Functionality

- A method getting control from URL request may have any name
  - The Spring Controller invokes the proper method. The invoked method usually returns name of a view. Below are several examples

```
@RequestMapping ({“/”})  //method returning initial home view
String homeMethod()          //URL – root application folder
{                                       //(http://localhost:8080/books)
        return “home”;           //books – root application directory
}
@RequestMapping({“/add”}) //method returning input form view
String addMethod()              // (http://localhost:8080/books/add)
{
        return “add_view”;
}
@RequestMapping({“/query”}) //method returning input form view
String queryMethod()              // (http://localhost:8080/books/add)
{
        return “query_view”;
}
```

Application Controller Functionality – Example (add)

- A method getting control from URL request may have reference to an object of the class ***HttpServletRequest***
  - The Spring Controller passes reference to the object of ***HttpServletRequest***

```
@RequestMapping ({“/add_performing”})  //method returning view with result
String adding(HttpServletRequest  request, Model model)
{                                    //(http://localhost:8080/books/add_performing)
        IBookDB db=ctx.getBean(“database”); //getting reference for database
        String title=request.getParameter(“title”);
        String authors=request.getParameter(“authors”);
        String publisher=request.getParameter(“publisher”);
        boolean result=db.createBook(title, authors.split(“;”), publisher);
        if(result) model.addAttribute(“result”,”book ”+title+” was added”);
        else model.addAttribute(“result”,”book ”+title+” was not added”);
        return “result_view”;
}
```

Useful Methods of class HttpServletRequest

String getParameter(String parameterName); - returns the value of the parameter

parameterName – name pointed in the input element of an JSP file

Enumeration<String> getParameterNames(); - returns enumeration of the all parameter names

Enumeration is the class having two main methods:

boolean hasMoreElements();

String nextElement();

```
Wile(names.hasMoreElements()) {
    String name=names.nextElement();
          ……………………………….
}
```

View Example for Input home_view

```
<title>Home Book</title>
</head>
<body>
<p> please choose action </p>
<form action="add">
  <input type="submit" name="add book"/> <br>
</form>
<form action="query">
  <input type="submit" name="query book"/> <br>
</form>
</body>
```

View Example for Input add_view

```
<title>Input Book</title>
</head>
<body>
<form action="add_performing">
title <input type="text" name="title"/> <br>
authors <input type="text" name="authors"/><br>
publisher <input type="text" name="publisher"/> <br>
<input type="submit" value="add book"/>
</form>
</body>
```

Application Controller Functionality – Example (query)

- A method getting control from URL request may have references to the String parameters with the same names as input field names
  - The Spring Controller passes reference to the proper string values

```
@RequestMapping ({“/query_performing”})
String adding(String jpaStr, Model model)
{                    //(http://localhost:8080/books/query_performing)
        ICliTest test=ctx.getBean(“test”); //getting reference for database
        String [] result=test.execute(jpaStr);
        StringBuffer buf=new StringBuffer();
        for(String str: result) {
           buf.append(str);
            buf.append(“<br>”);
        }
        return “result_view”;
}
```

View Example for Output result_view

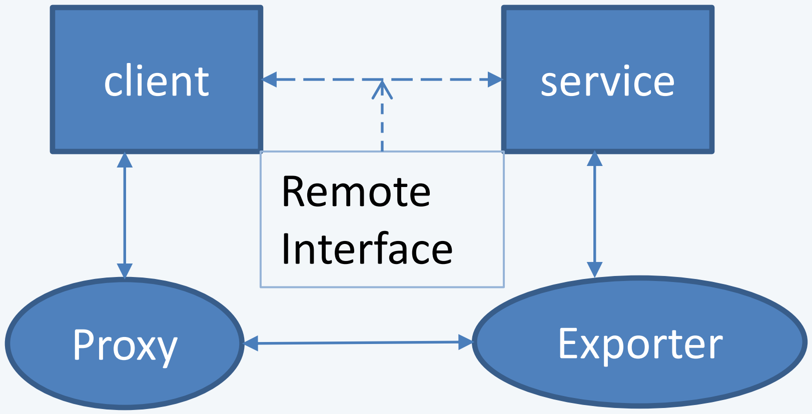
```
<title>Result Book</title>
</head>
<body>
<script type="text/javascript">
        document.write("${result}");
</script>
```

Services and SOA

- Component that can be used remotely through a remote interface either synchronously or asynchronously (e.g. Web service, messaging system, sockets, RPC etc)
- Step up from “distributed objects”
- Function that has a clearly defined service contract to their consumers or clients, self contained and does not depend on the context or state of other services
- SOA – Service Oriented Architecture
  - OOD for services

Spring and Services

- Remote Method Invocation - RMI
- Web Services
  - Caucho Hessian
    - Binary format
  - Caucho Burlap
    - XML formAT
  - HTTP Invoker
  - JAX-WS / JAX-RPC



Proxy communicates with service on behalf of the client

Exporter communicates with client on behalf of the service

Java Messaging Service

- JMS provides a vendor-independent API for messaging systems
- JMS provides a reliable, asynchronous and loosely coupled way to communicate between applications
  - Reliable: even if the recipient is down, the message will arrive when it is up again
  - Asynchronous: the sender doesn’t wait for the message to be received
  - Loosely-coupled: the sender isn’t aware of the implementation of the receiver. There are standard message types that can be used
- Two messaging types
  - PTP point-to-point, queue from which only one receiver may get message
  - Publish-and-Subscribe, publisher publishes message on the “topic” and each subscriber will get a message.

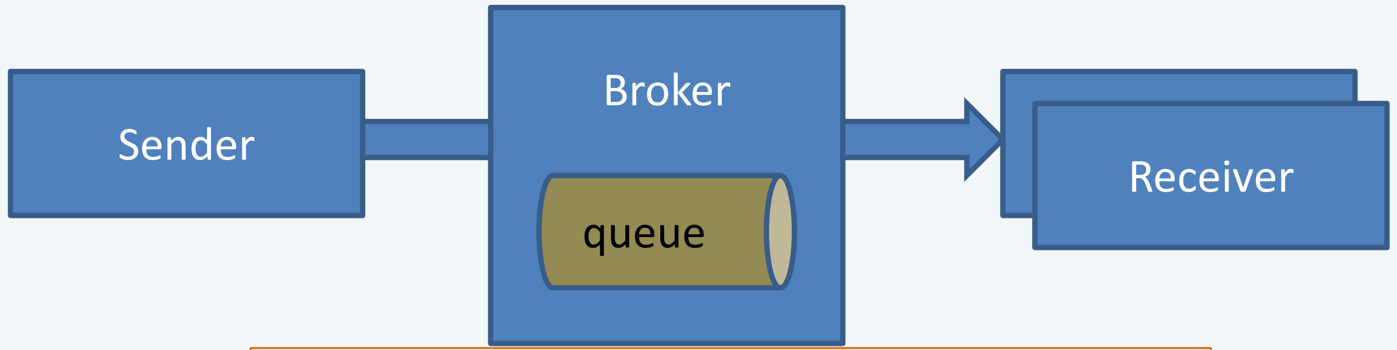
Message Types

- TextMessage – holds a String.
- MapMessage – holds a map of key-value pairs in

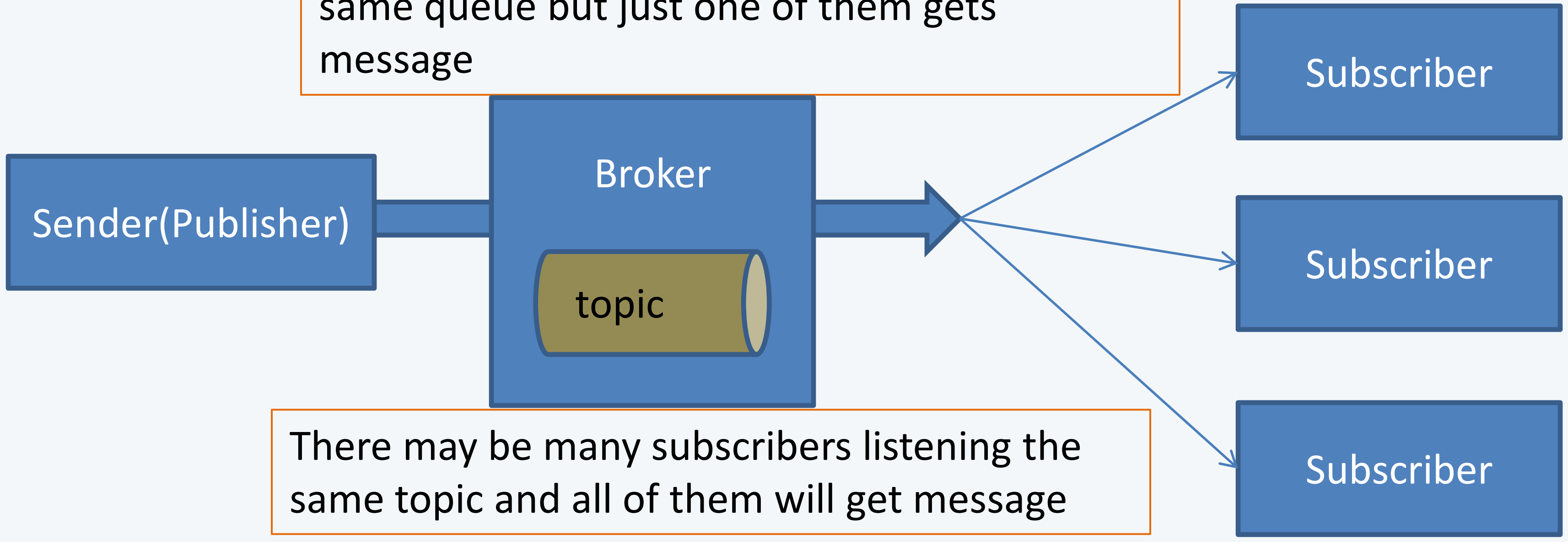
which the key is a String and the value a primitive.

- StreamMessage – a stream of Java primitives written sequentially.
- BytesMessage – a stream of un-interpreted bytes. Used mainly to support a legacy formats.
- ObjectMessage – Holds a Serializable object.

Architecture JMS



There may be many receivers listening the same queue but just one of them gets message



There may be many subscribers listening the same topic and all of them will get message

ActiveMQ

- Open well known Messaging Broker
  - Supports (C++,Java,C#, and many others)
- Installation
  - Download from …lesson8/apache-activemq-5.10.0-bin.zip
  - Unzip
  - Launch from command string …/activemg start
- Management
  - http://localhost:8161/admin/ (admin/admin)

XML file for Sender

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jms="http://www.springframework.org/schema/jms"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xsi:schemaLocation="http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core-5.10.0.xsd
    http://www.springframework.org/schema/jms
    http://www.springframework.org/schema/jms/spring-jms-3.0.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:61616"/>
    <bean id="jmsTemplate"
   class="org.springframework.jms.core.JmsTemplate">
  <property name="connectionFactory" ref="connectionFactory" />
  <property name="defaultDestinationName" value="myqueue"></property>
</bean>
</beans>
```

XML Receiver

```
<amq:connectionFactory id="connectionFactory"
brokerURL="tcp://localhost:61616"/>
        <bean id="queue"
class="org.apache.activemq.command.ActiveMQQueue">
    <constructor-arg value="myqueue"/>
</bean>
<jms:listener-container connection-factory="connectionFactory">
    <jms:listener destination="myqueue"
        ref="pointHeandler" method="processPoint"  />

</jms:listener-container>
 <bean id="pointHeandler" class="PointHeandler"/>
```

Sender using JmsTemplate

void send(MessageCreator creator);

MessageCreator – interface that a programmer should implement writing method

Message createMessage(Session session);

This method gets reference to object session and programmer should call method create according to the message type
createTextMessage(String text);
createObjectMessage(Object obj);

…………………………………………………

Receiver – Message Driven POJO (MDP)

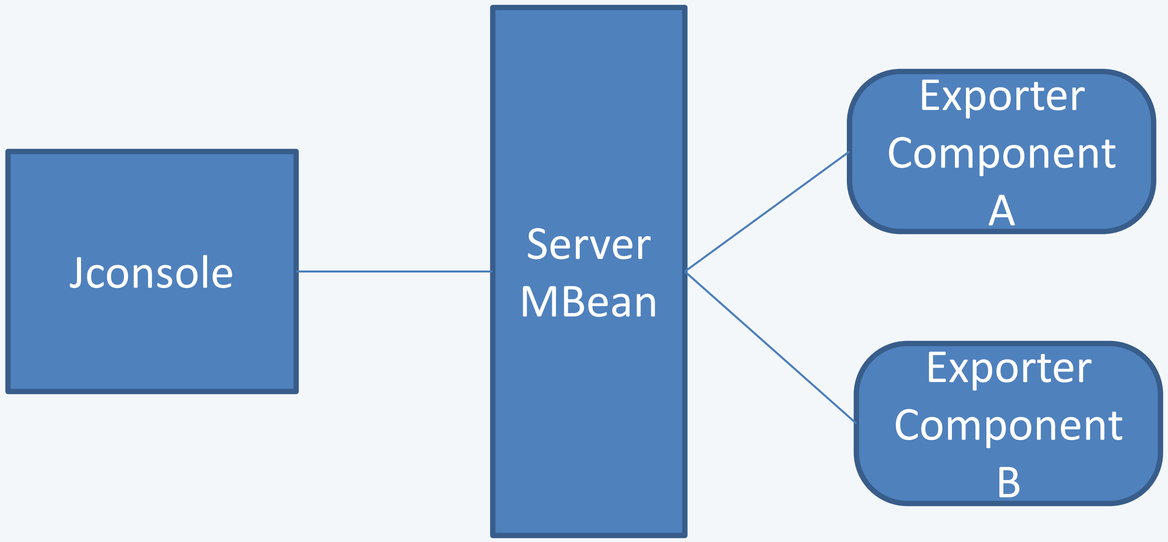
- MDP – is a regular POJO defined as Spring bean in the configuration file
- Spring listener listens to queue/topic and in the case there is new message forwards control to the pointed in the configuration method

```
public class PointHeandler {
public void processPoint(Point point){
System.out.println("x="+point.getX()
);
System.out.println("y="+point.getY()
);
}
```

JMX – Java Management Extensions

- Tools for control, monitoring and maintaining of the Java applications
- MBean – Java component intended for a remote control using JMX
- Spring allows exporting the java components as MBeans

Control from Jconsole

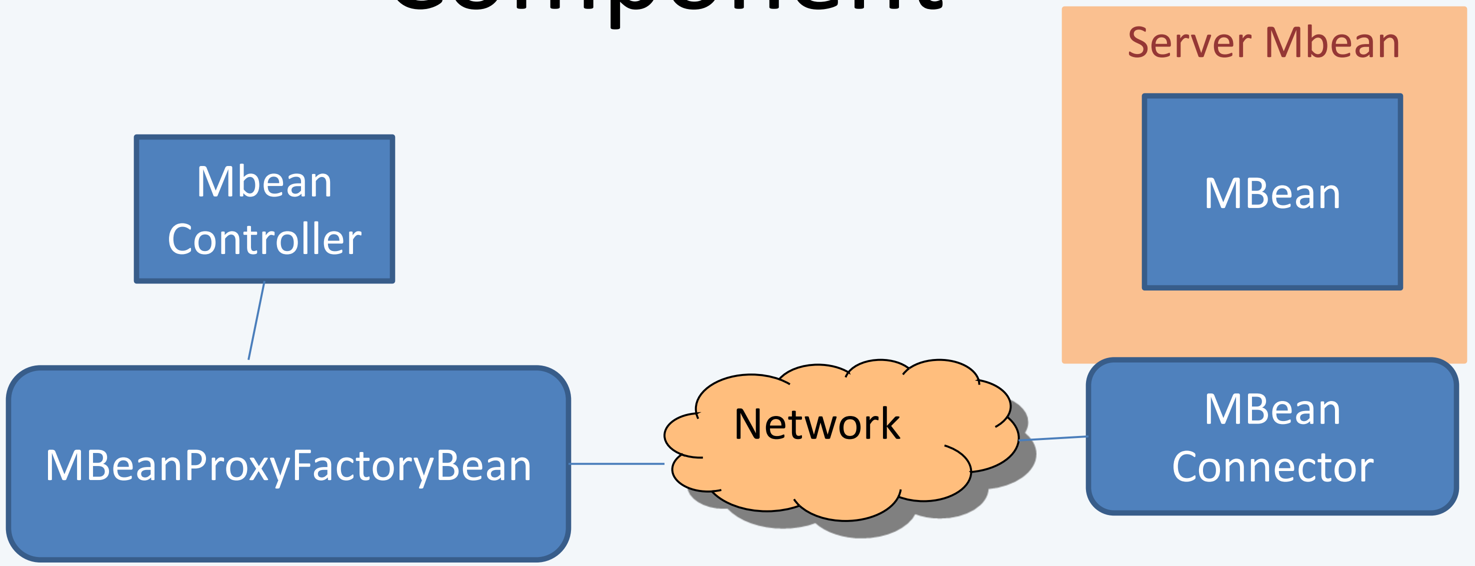


- Jconsole is tool of Java
  - <jdk folder>/bin/jconsole

Configuration file for MBean Server and Spring Component

```
<context:mbean-server />
  <bean id="mbeanExporter"
    class="org.springframework.jmx.export.MBeanExporter">
  <property name="beans">
    <map>
      <entry key="sportsman:name=Sport"
            value-ref="sport"/>
    </map>
  </property>
  <property name="server" ref="mbeanServer" />
</bean>
```

Remote Control from Spring Component



Proxy Factory Bean communicates with MBean component and invokes any methods declared in the client interface

Configuration on MBean Server

```
<bean class="org.springframework.jmx.support.ConnectorServerFactoryBean"
>
 <property name="serviceUrl"
value="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/sportsman"></p
roperty>
  </bean>
  <bean class="org.springframework.remoting.rmi.RmiRegistryFactoryBean">
  <property name="port" value="1099"></property>
  </bean>
```

Requires launching of the RMI Registry:
`<jdk tool folder>/bin/rmiregistry`

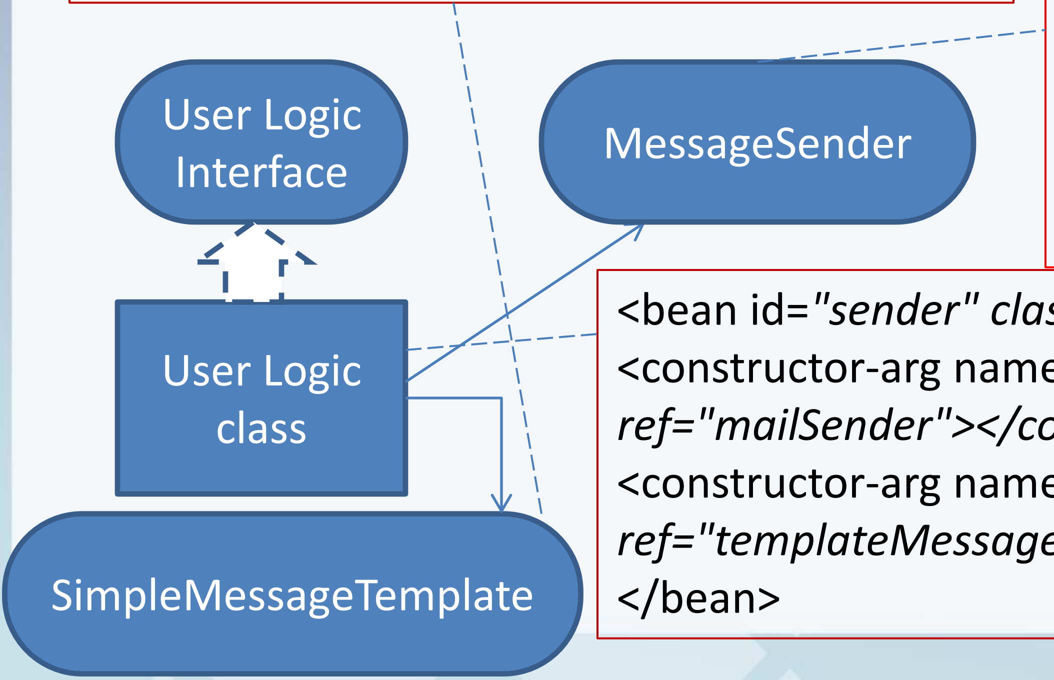
Configuration of MBean Controller

```
<bean id="mBeanServerClient"
    class=
        "org.springframework.jmx.support.MBeanServerConnectionFactoryBean">
        <property name="serviceUrl"
value="service:jmx:rmi://localhost/jndi/rmi://localhost:1099/sportsman"></property>
        </bean>
<bean id="remoteHomeControllerMBean"
    class="org.springframework.jmx.access.MBeanProxyFactoryBean">
        <property name="objectName" value="sportsman:name=Sport"></property>
        <property name="server" ref="mBeanServerClient"></property>
        <property name="proxyInterface"
value="sport.operations.SportExportOperations"></property>
 </bean>
```

Mail Sending Service

```
<bean id="templateMessage"
class="org.springframework.mail.SimpleMail
Message">
   <property name="from"
value="yuragranovsky@gmail.com"/>
   <property name="subject"
value="Invitation"/>
</bean>
```

```
<bean id="mailSender"
class="org.springframework.mail.javamail.Java
MailSenderImpl">
   <property name="host"
value="smtp.gmail.com"/>
   <property name="port" value="25"/>
   <property name="username"
value="yuragranovsky"/>
   <property name="password"
value="${password}"/>
   <property name="javaMailProperties">
    <props>
       <prop
key="mail.smtp.starttls.enable">true</prop>
       <prop key="mail.debug">false</prop>
    </props>
   </property>
</bean>
```



```
<bean id="sender" class= "tel_ran.mail.InvitationSender">
<constructor-arg name="sender"
ref="mailSender"></constructor-arg>
<constructor-arg name="template"
ref="templateMessage"></constructor-arg>
</bean>
```

Parameterized Spring Configuration

```
<bean id="applicationProperties" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer" lazy-init="default">
   <property name="location" value="classpath:application.properties"/>
</bean>
```

There is file application.properties into directory …/src inside the project
For each parameter there should be line as follows:
<parameter>=<value>

Inside Spring configuration file:

……………………………………………………………………………………………….

```
<property name="password" value="${password}"/>
```

TEL-RAN