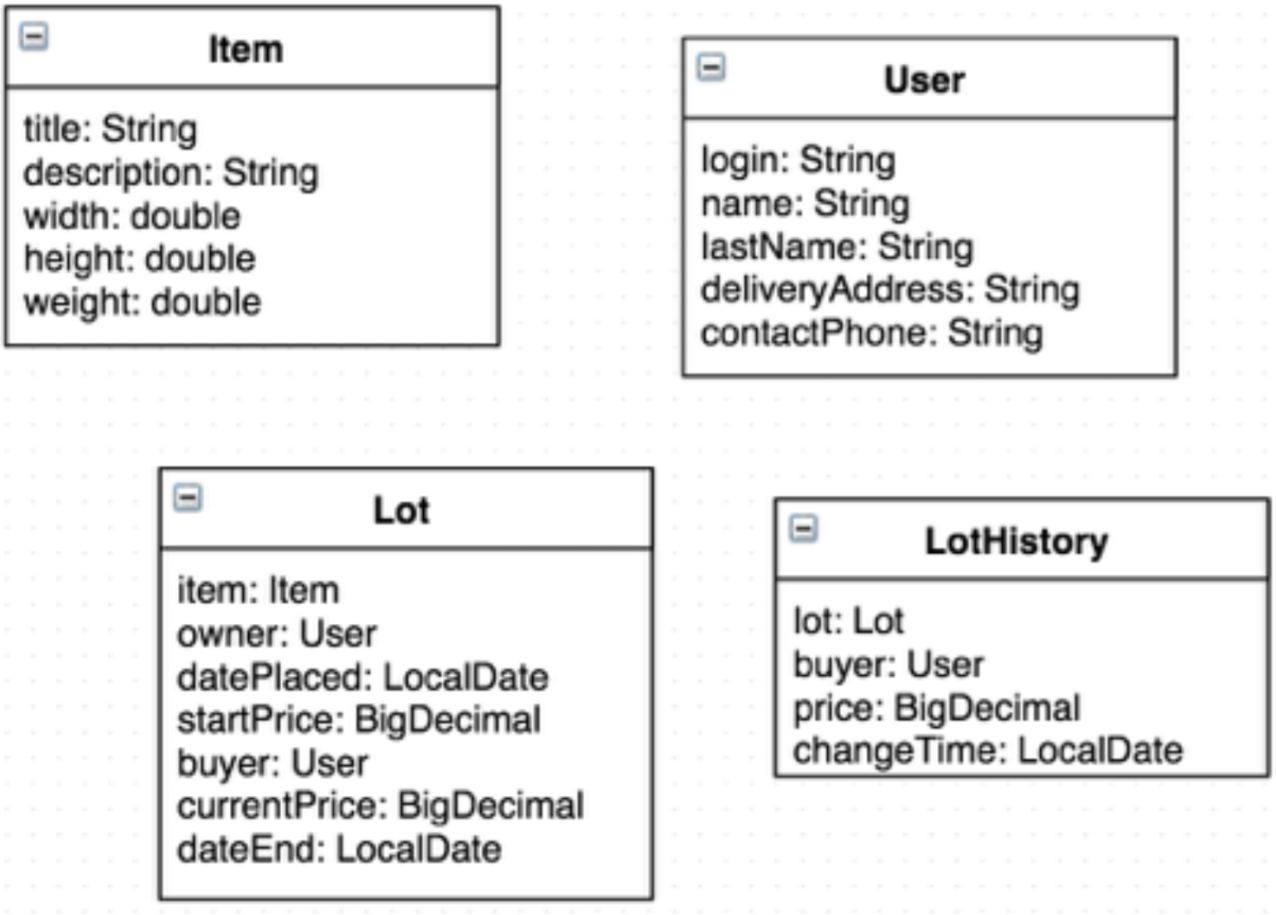
SkillsUp. Java 1. Practice 1

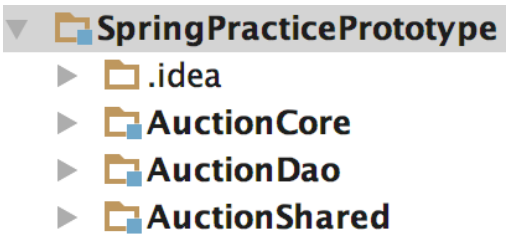
1. Project introduction overview

Main goal: create mocked auction system for entities, described bellow



Modules structure:

- auction-core - contains all business logic, services etc.
- auction-dao - contains mocked DAO services (as we don't use Hibernate or another JDBC stuff)
- auction-shared - contains all common entities as well as utility classes



2. Maven configuration

- Assign to each module name to be like on image bellow

```
[INFO] Reactor Summary:
[INFO]
[INFO] auction-aggregator ................................ SUCCESS [0.340s]
[INFO] auction-practice-core ............................. SUCCESS [1.232s]
[INFO] auction-practice-dao .............................. SUCCESS [0.401s]
[INFO] auction-practice-shared ........................... SUCCESS [1.147s]
```

- In parent module define <dependencyManagement> section to share spring framework dependencies declaration within modules

```
<!--Spring Framework-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${spring.version}</version>
</dependency>
```

- Move spring version declaration to properties section of parent module

```
<properties>
    <spring.version>4.3.0.RELEASE</spring.version>
</properties>
```

- In same way add logging dependencies: log4j : log4j : 1.2.17 and org.slf4j : slf4j-log4j12 : 1.7.13
- By default maven compiler plugin uses Java 1.5, so to not have compile issue in future in build -> pluginManagement -> plugins section of parent module add source and target version as 1.8 for maven-compiler-plugin

```
<plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.3</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
    </configuration>
</plugin>
```

- Add **shared** module like a dependency to both **core** and **DAO** modules.
- Also in **core** and **DAO** define dependencies for spring libraries and logs, declared in parent module. Skip version section as it controlled by dependencyManagement declaration.

3. Implementing DAO

As any DAO class usually operate with common CRUD actions for time saving in DAO module we have GenericDao interface:

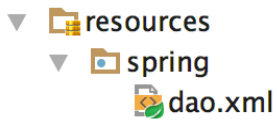
```
public interface GenericDao<E> {

    E save(E entity);
    void update(E entity);
    void delete(E entity);
    E findById(long id);
    List<E> findAll();

}
```

And it's common realisation in GenericDaoImpl abstract class.

- Create interface for each entity that should be extended from GenericDao
- Create implementation for each interface in impl package and extend it from GenericDaoImpl
- Add spring xml context to resources dir -> spring/dao.xml




```
<?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.xsd">

</beans>
```

- Declare UserDao bean in spring context

```
<bean class="skillsup.practice.spring.dao.impl.UserDaoImpl" />
```

- Anywhere in DAO module create class-runner (with main method) and add logic to get UserDao bean from context and call some it's method

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring/dao.xml");

UserDao userDao = context.getBean(UserDao.class);

User savedUser = userDao.save(user);
System.out.println(savedUser);
```

- In context.getBean() change argument to GenericDao.class and rerun main method. Does anything change?
- Add other DAO implementations to context. Rerun main method again and explain what went wrong.
- Change code to get all GenericDao beans at all. Take a look at what ID spring create for each bean if we didn't define it

```
Map<String, GenericDao> allDao = context.getBeansOfType(GenericDao.class);

System.out.println(allDao);
```

- Give ID for each defined bean. Get ItemDao by ID
- For UserDao add initial configuration, so that after starting context bean will have several users in it. Use **initMethod** in bean configuration for it.

4. Implementing business logic

```java
public interface AuctionService {

    /**
     * Create new lot in auction system
     * @param item item to sell
     * @param owner item's owner
     * @return created lot
     */
    Lot createLot(Item item, User owner);

    /**
     * Retrieve all lots with active auction status
     * @return list of active lots in Chronological order by start time
     */
    List<Lot> getAllActiveLots();

    /**
     * vote for lot with default price increase
     * @param lot lot to vote for
     * @param buyer user, who wants to buy an item
     */
    void vote(Lot lot, User buyer);

    /**
     * Vote for lot with custom price rising.
     * @param lot lot to vote for
     * @param buyer user, who wants to buy an item
     * @param newPrice new price shouldn't be less then current!
     *                     Price rising shouldn't be less default delay
     */
    void vote(Lot lot, User buyer, BigDecimal newPrice);

    /**
     * Get lot's history in chronological order
     * @param lot lot, which history should be displayed
     * @return lots history in chronological order
     */
    List<LotHistory> getLotHistory(Lot lot);
}
```

- Interface for AuctionService described above. Add that interface to your project and add class that will implement that interface.
- In **core** module add spring context like for **DAO** in spring/core.xml and define in context bean for auction main service.
- Auction service should have several default parameters, that should be read from external property file: auction duration (how long lot should be available on auction after putting), min bid step (minimal value to change a price of a lot).

  Add fields for that values in AuctionService and prepare setters for it.

In resources put file default.properties and define in it mentioned earlier properties:

```
auction.duration.days=5
min.bid.step=10
```

Configure spring context to obtain properties from file and injecting them into bean:

```
<?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:context="http://www.springframework.org/schema/context"

       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">

    <context:property-placeholder location="default.properties" />

    <bean class="skillsup.practice.spring.core.impl.AuctionServiceImpl">
        <property name="auctionDurationInDays" value="${auction.duration.days}"/>
        <property name="minBidStep" value="${min.bid.step}"/>
    </bean>

</beans>
```

- AuctionService require all DAO we created earlier. To connect them first of all add dependency on DAO module to core/pom.xml. In AuctionService implementation add fields for each DAO and setters for them.
  To have possibility to use already declared beans in spring/dao.xml you should import it to spring/core.xml

```
<import resource="dao.xml"/>
```

  Please, notice that for injecting one bean into another you should use ref instead of value

```
<property name="itemDao" ref="itemDao"/>
```

- Implement AuctionService#getAllActiveLots() method. Write small simulation with creating several Lots and then getting them with this method

5. Adding third party library

- Add additional method, that will consume List<Lot> and instead return it's JSON representation in String. For retrieving JSON use GSON library from Google.

  Example of GSON usage is here:

```
package com.javacreed.examples.gson.part1;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class SimpleExample1 {
    public static void main(String[] args) {
        Gson gson = new GsonBuilder().create();
        gson.toJson("Hello", System.out);
        gson.toJson(123, System.out);
    }
}
```

  So to create Gson object that will make all job for you first of all you should have GsonBuilder bean. On declaring Gson bean in context use GsonBuilder as it's FactoryBean and create() as it's factory method.

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.3.1</version>
</dependency>
```

6. Switching to annotation config.

- Remove bean's declaration from xml files (what declaration couldn't be removed?) and put instead required annotation for each bean.
- To let Spring know where to find beans add following lines to context:

```
<context:annotation-config/>
<context:component-scan base-package="skillsup.practice.spring.core" />
```

Please, notice, that it's better to define only packages of current module. For other modules it's better still to have context imports.

7. Java configuration

- Create spring configuration class in **core** module
- Annotate it with @Configuration, add Component scan annotation to find declared services, add import resource annotation to include beans from **dao** module.
- Try to create bean for Gson object in Java annotation
- In main method replace xml config with following

```
ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfig.class);
```

8. Implement all functionality, declared in AuctionService interface