Создать консольное приложение, удовлетворяющее следующим требованиям:  
• Использовать возможности ООП: классы, наследование, полиморфизм, ин-  
капсуляция.  
• Каждый класс должен иметь отражающее смысл название и информатив-  
ный состав.  
• Наследование должно применяться только тогда, когда это имеет смысл.  
• При кодировании должны быть использованы соглашения об оформлении  
кода java code convention.  
• Классы должны быть грамотно разложены по пакетам  
• Консольное меню должно быть минимальным.

Приложение должно быть в многослойной архитектуре (beans-классы сущностей, services-бизнес-логика, dao или repository-доступ к данным (сначала - эмуляция базы данных коллекцией, потом файлами), controller- диспетчер, распределяющий запросы пользователя)

Написать иерархию классов по одному выбранному проекту в коде.

1.социальная сеть

2.аукцион

3.блог

4.интернет-магазин

5.библиотека

6.таск менеджер

7.поликлиника

8.новостной портал

9.LMS(learning management system)

10.Университет

11.Бронирование билетов

12.Туристические туры

13. Аналог Uber

14.Больница

Или любой другой реалистичный проект

**Entity-классы не следует наполнять методами**, выполняющими функциональные действия  
(методами бизнес-логики, такими как вычисление, поиск и т.д.).

По поводу иерархии классов в проекте.  
Рисуем схему как можно более полную, со всеми необходимыми полями и методами, с количеством сущностей, покрывающих предметную область (чтобы научиться абстракции, декомпозиции и пр.).  
Реализуем в коде около 10-15 классов-бинов(энтитей) (в зависимости от задачи) необходимых для вычисления, описанного в условии. В классе 3-5 полей и методов, необходимых для вычисления, описанного в условии.

Задача учебная, вы должны на ней просто прощупать руками принципы ООП и разные типы связей между объектами (а после и разные возможности java). Нет необходимости писать много похожего кода для этого.

Если необходимо, расширяем условие задачи, когда это требуется для реализации описанных требований к проекту. Лучше меньше да лучше. Сначала реализуем минимальный работающий функционал (MVP), потом, если есть время – наращиваем функционал.

При проектировании классов проекта не забываем пользоваться рекомендациями из книги Блинова (стр. 80-87, 123, 48, 218 и др.)

Переопределить методы equals(), hashCode(), toString().

Добавить в классы проекта необходимые конструкторы  
Добавить в классы проекта геттеры и сеттеры.

Использовать в классах проекта правильные модификаторы доступа.

Реализовать в проекте счѐтчик созданных объектов

Реализовать в проекте интерфейсы и абстрактные классы

Реализовать в проекте модуля custom exceptions и их корректную обработку

Посмотреть в исходном коде Java как реализованы коллекции (интерфейсы, абстрактные классы, конкретные классы). Перенять элементы архитектуры коллекций для своих проектов. Реализовать List, Set, Map в проекте  
Реализовать Comparable и Comparator в проекте  
Использовать в проекте классы даты и времени

Реализовать запись/чтение символьной и байтовой информации в файл в проекте  
Реализовать сериализацию в проекте (иерархия, static, transient)

Реализовать перечисления в проекте  
Реализовать параметризацию (generics) в проекте

Реализовать regexp в проекте

Реализовать функцию поиска по сущностям.

﻿Делать валидацию всех данных  
Добиться “защиты от дурака”, создать непадательное состояние проекта вне зависимости от любых вводимых данных (корректных или не корректных)

Настроить IDE на GoogleStyle

Автоформатирование – необходимо во всех классах

Написать javadoc для классов из проекта  
Сгенерировать javadoc в IDE для этих классов

Перед отправлением кода на проверку, устранять замечания утилит статического анализа кода (Code/Inspect code, Code/Analyze code в Idea, SonarLint, FindBugs, PMD и др.)

Написать юнит-тесты с использованием Junit и Mockito

Проект должен быть размещен на github (gitlab)

В проекте необходимо использовать log4j для логирования, maven для билда

Реализовать логику логина в приложения под разными ролями (admin, user), с доступом к разному функционалу в зависимости от роли

Программировать в интерфейсах

Избегать использовать статические методы (за исключением утилитных)

Изучить Appendix 1

Appendix 1:1. После if всегда следует положительный сценарий, после else -отрицательный2. Если только if, то возможен и отрицательный сценарий3. В if, for, while обязательно использовать { }4. Если генерируется exception, не ловить его сразу же5. В finally не генерировать исключения и не использовать return6. Не генерировавать стандартные исключения. Разрешено только вметодах private7. fileWriter.close(); - в блок finally8. Регулярные выражения в константы9. В именах пакетов не использовать большие буквы10. Не класть сторонние файлы в папки рядом или вместе с классами11. Размещать файлы только в папках в корне проекта12. Использовать для файлов только относительные пути. Папка srcне должна присутствовать в пути к файлу13. Если константа неизменяемая, то имя должно быть в верхнемрегистре, если изменяемая, то как правило именуется как обычноеполе класса14. Элементы перечисления именуются как неизменяемые константы15. Не увлекаться статическими методами16. Не объявлять get-теры и set-теры абстрактными17. Не давать классам имена, совпадающие с именами стандартныхклассов и интерфейсов Java !18. Не разделять объявление переременной и присвоение ейзначения в методах, то есть не писать:Integer count;count = 0;а надо Integer count = 0;19. Расстояние (в строчках кода) между использование переменной иее объявление должно быть минимально!20. В одной строчке – одна точка, то есть надо использоватьлокальные переменные, не надо:sample.getSomething().getData()надо:Something something = sample.getSomething();Data data = something.getData();21. Не писать if (isValid == true), а писать if (isValid)

22. Не писать:

if (someValue == EXPECTED\_VALUE) {

return true;} else {return false;}писать:  
return someValue == EXPECTED\_VALUE;23. Использовать assertEquals вместо assertTrue(some == other)

24. Использовать assertTrue(isValid) вместо assertEquals(true,

isValid)

25. Лучше тестовые объекты размещать в тестах в виде констант, ане создавать их в самом методе26. Не использовать существующий FactoryMethod в тестах длясоздания объектов, объекты в тестах делать через new27. Тест должен иметь структуру: given, when, then, где given –прекондишны (инициализация данных), when – вызов тестируемогометода (всегда одна строчка), then – посткондишн (assert-метод)