Практические задачи по курсу ООП

Правила оформления и сдачи практических задач

Задачи

1.1 Пирамидальная сортировка Task_1_1_1 2

1.2 Консольный блэкджек Task_1_1_2 3

1.3 Операции с уравнениями Task_1_1_3 5

2.1 Граф Task_1_2_1 7

2.2 Хеш-таблица Task_1_2_2 8

3.1 Поиск подстроки Task_1_3_1 9

4.1 Зачетная книжка Task_1_4_1 10

5.1 Генератор markdown Task_1_5_1 11

1.1 Пирамидальная сортировка (1 неделя) `Task_1_1_1`

Реализуйте классический алгоритм пирамидальной сортировки.
Кроме настроенной сборки через gradle, необходимо предоставить shell-скрипт, который без использования системы сборки с помощью комбинации утилит javac, jar и java компилирует исходники, генерирует документацию и запускает приложение.
Протестируйте свой код с помощью юнит-тестов. При сдаче задания преподавателю объясните, как выбранный набор тестов обеспечивает корректность вашего алгоритма.
Покажите, что теоретическая сложность алгоритма подтверждается на практике.

Вход	`heapsort(new int[] {5, 4, 3, 2, 1});`
Выход	`[1, 2, 3, 4, 5]`

1.2 Консольный блэкджек (2 недели) `Task_1_1_2`

Реализуйте консольный вариант карточной игры Блэкджек по следующим правилам:

1. В игре используется одна или несколько колод из 52 карт.
2. В игре участвуют Игрок (пользователь) и Дилер (эмулируется приложением).
3. Каждой карте соответствует значение очков, которые начисляются тому, кто ее открыл. В комбинации карт их значения суммируются.
   Числовые карты дают значение по указанному на них числу (от 2 до 10), карты с картинками (дама, валет, король) – 10 очков, тузы дают по 11 очков, если итоговая сумма карт на руках (учитывая тузы) не превышает 21, и 1, если превышает.
4. Игра состоит из раундов. В начале раунда Дилер раздает по две карты Игроку и себе, оставляя одну свою карту закрытой.
5. Комбинация из двух карт, дающая сумму 21, называется блэкджек и автоматически делает своего обладателя победителем.
6. После раздачи карт, если Игрок не стал обладателем блек-джека, он делает свой ход. Цель Игрока – победить Дилера, набрав за раунд сумму очков выше, чем у Дилера, но не превышающую значение 21. Для этого в течение своего хода он может открывать по одной очередной карте из колоды до тех пор, пока не пожелает остановиться.
7. По завершении хода Игрока Дилер открывает закрытую карту и, пока сумма комбинации на руках Дилера не превысит или не станет равной значению 17, он обязан открывать по очередной карте из колоды. По достижении этого значения Дилер завершает ход.
8. Любой участник игры сразу проигрывает, если сумма очков его комбинации карт превысила 21.
9. Если в течение раунда не было блекджеков и обе комбинации не превысили значение 21, побеждает участник с наибольшей суммой очков.

В решении используйте объектно-ориентированный подход: продумайте объектную модель приложения и отношения между абстракциями.

Выход	Добро пожаловать в Блэкджек!
	Раунд 1
	Дилер раздал карты
	    Ваши карты: [Пиковая дама (10), Тройка Червы (3)] ⇒ 13
	    Карты дилера: [Туз Трефы (11), <закрытая карта>]

```
Ваш ход
-------
Введите “1”, чтобы взять карту, и “0”, чтобы остановиться...
1
Вы открыли карту Семерка Пики (7)
    Ваши карты: [Пиковая дама (10), Тройка Червы (3), Семерка Пики
(7)] ⇒ 20
    Карты дилера: [Туз Трефы (11), <закрытая карта>]

Введите “1”, чтобы взять карту, и “0”, чтобы остановиться...
0

Ход дилера
-------
Дилер открывает закрытую карту Тройка Трефы (3)
    Ваши карты: [Пиковая дама (10), Тройка Червы (3), Семерка Пики
(7)] ⇒ 20
    Карты дилера: [Туз Трефы (11), Тройка Трефы (3)] ⇒ 14

Дилер открывает карту Десятка Пики (10)
    Ваши карты: [Пиковая дама (10), Тройка Червы (3), Семерка Пики
(7)] ⇒ 20
    Карты дилера: [Туз Трефы (1), Тройка Трефы (3), Десятка Пики
(10)] ⇒ 14

Дилер открывает карту Бубновый король (10)
    Ваши карты: [Пиковая дама (10), Тройка Червы (3), Семерка Пики
(7)] ⇒ 20
    Карты дилера: [Туз Трефы (1), Тройка Трефы (3), Десятка Пики
(10), Бубновый король (10)] ⇒ 24

Вы выиграли раунд! Счет 1:0 в вашу пользу.

Раунд 2
...
```

1.3 Операции с уравнениями (2 недели) `Task_1_1_3`

Реализуйте иерархию классов для работы с математическими выражениями. В корне иерархии расположите абстрактный класс `Expression`. Его производные классы `Number`, `Variable`, `Add`, `Sub`, `Mul`, `Div` должны соответствовать различным выражениям: константе, переменной, сумме, разности, произведению и частному двух других выражений.

Вход	`Expression e = new Add(new Number(3), new Mul(new Number(2), new Variable("x"))); // (3+(2*x))`

Для указанной иерархии реализуйте следующую функциональность:

1. Создание выражения по заданной в файле/консоли строке. Считать, что каждое выражение, кроме переменных и констант, всегда пишется в скобках.
2. Печать выражения в файл/консоль по аналогичным правилам.

Вход	`e.print();`
Выход	`(3+(2*x))`

3. Символьное дифференцирование выражений по заданной переменной.

Вход	`Expression de = e.derivative("x");` `de.print();`
Выход	`(0+((0*x)+(2*1)))`

   При этом изначальное выражение не меняется, а создается новое соответствующее результату дифференцирования. Также обратите внимание, что названия переменных могут быть многобуквенными.

4. Вычисление значения выражения при заданном означивании всех переменных.

Вход	`Expression e = new Add(new Number(3), new Mul(new Number(2), new Variable("x"))); // (3+(2*x))`
	`int result = e.eval("x = 10; y = 13");`
	`System.out.println(result);`

Выход	`23`

   Функция eval в качестве аргумента принимает строку, в которой через разделитель “;” указано означивание переменных.

В качестве дополнительного задания реализуйте:

1. Считывание выражения со строки, работающее со строкой без скобок вокруг каждого выражения.
2. Функцию для упрощения выражений по следующим правилам:
    a. Если выражение можно вычислить без какого-либо означивания (т.е. в нем нет переменных), то выражение заменяется на результат вычисления.
    b. Если выражение представляет из себя умножение на 0, то оно заменяется на константу 0.
    c. Если выражение представляет из себя умножение на 1, то оно заменяется на второй множитель.
    d. Если выражение представляет из себя вычитание двух одинаковых подвыражений, то оно заменяется на константу 0.

    Упрощение не меняет исходное выражение, а создает новое (упрощенное) выражение по заданному.

2.1 Граф (2 недели) `Task_1_2_1`

Создайте интерфейс `Graph` со следующими операциями:

1. добавление и удаление вершины;
2. добавление и удаление ребра;
3. получение списка всех “соседей” вершины;
4. чтение из файла фиксированного формата;
5. другие необходимые операции.

В качестве реализаций используйте три классических способа представления (хранения) графа: через матрицу смежности, матрицу инцидентности и список смежности. Не забудьте поддержать операции сравнения на равенство и вывода в строку.

Для объекта `Graph` реализуйте алгоритмы топологической сортировки вершин. При сдаче задания объясните, как добавить в имеющуюся объектную модель новый способ представления графа или новый алгоритм сортировки вершин.

2.2 Хеш-таблица (2 недели) `Task_1_2_2`

Реализуйте параметризованный контейнер `HashTable<K, V>`, который позволит добавлять, искать и удалять объекты `V` по ключу `K` за константное время. Контейнер должен поддерживать полный набор операций:

1. создание пустой хеш-таблицы;
2. добавление пары ключ-значение *(k, v)*;
3. удаление пары ключ-значение *(k, v)*;
4. поиск значения по ключу;
5. обновление значения по ключу;
6. проверка наличия значения по ключу;
7. итерирование по элементам коллекции (по парам *(k, v)*) с обработкой исключительной ситуации `ConcurrentModificationException`;
8. сравнение на равенство с другой хеш-таблицей;
9. вывод в строку.

В своем решении также учтите проблемы коллизий и использования оптимального объема памяти.

Вход	`HashTable<String, Number> hashTable = new HashTable<>();`
	`hashTable.put("one", 1);`
	`hashTable.update("one", 1.0);`
	`System.out.println(hashTable.get("one"));`

Выход	`1.0`

3.1 Поиск подстроки (2 недели) `Task_1_3_1`

На вход подаётся имя файла и строка, которую нужно найти. Строка может содержать буквы любого алфавита в кодировке UTF-8. Реализуйте функцию, определяющую индекс начала каждого вхождения заданной подстроки.
В своем решении учтите:

- размер входного файла может быть во много раз больше объема оперативной памяти вычислительного устройства, а размер искомой подстроки — во много раз меньше;
- большие данные, необходимые только для тестирования, не нужно добавлять в систему контроля версий, их нужно генерировать в процессе тестирования.

Файл	`input.txt:` `абракадабра`
Вход	`find("input.txt", "бра");`
Выход	`[1, 8]`

4.1 Зачетная книжка (1 неделя) `Task_1_4_1`

Реализуйте класс электронной зачетной книжки студента ФИТ, обеспечив следующие функции вычисления:

1. текущего среднего балла за все время обучения;
2. возможности перевода с платной на бюджетную форму обучения (в случае обучения на платной основе);
3. возможности получения «красного» диплома с отличием;
4. возможности получения повышенной стипендии в текущем семестре.

Требование для перевода с платной на бюджетную форму обучения: отсутствие оценок “удовлетворительно” за последние две экзаменационные сессии (за экзамены, в дифференцированных зачетах допустимы оценки “удовлетворительно”).
Требования для диплома с отличием:

- 75% оценок в приложении к диплому (последняя оценка) – “отлично”;
- отсутствие итоговых оценок “удовлетворительно”. Как по дифференцированным зачетам, так и по экзаменам;
- квалификационная работа на “отлично”.

Обратите внимание, что функции должны учитывать текущее состояние зачетной книжки на текущем этапе обучения. Например, возможность получить “красный” диплом можно спрогнозировать до завершения всех курсов учебного плана и защиты квалификационной работы.
При сдаче задания объясните преподавателю свой выбор объектной модели.

Вид контроля	Количество видов контроля								
	Всего	Первый семестр	Второй семестр	Третий семестр	Четвертый семестр	Пятый семестр	Шестой семестр	Седьмой семестр	Восьмой семестр
Задание	15	2	2	3	2	2	2	2	
Контрольная	13	3	3	2	1	2	2		
Коллоквиум	2	1	1						
Экзамен	19	3	3	2	5	3	2	1	
Дифференцированный зачет	35	3	3	6	5	4	6	4	4
Зачет	7	3	2					1	1
Защита отчета по практике	3							1	2
Защита ВКР	1								1

5.1 Генератор markdown (4 недели) `Task_1_5_1`

Реализуйте библиотеку для генерации текста в формате Markdown. Начиная с базового класса `Element`, постройте иерархию markdown элементов, среди которых:

1. простой текст и его форматирования (текст с выделением, текст курсивом, зачеркнутый текст, однострочный код);
2. списки;
3. заголовки;
4. цитаты;
5. ссылки;
6. изображения;
7. задачи;
8. таблицы;
9. блоки кода.

Для каждого элемента реализуйте операции сериализации в строку и сравнения на равенство с другим элементом. В своем решении используйте паттерн проектирования Builder.

Вход	`Table.Builder tableBuilder = new Table.Builder()`
	`.withAlignments(Table.ALIGN_RIGHT, Table.ALIGN_LEFT)`
	`.withRowLimit(8)`
	`.addRow("Index", "Random");`
	`for (int i = 1; i ⩽ 5; i++) {`
	`final var value = (int) (Math.random() * 10);`
	`if (value > 5) {`
	`tableBuilder.addRow(i, new`
	`Text.Bold(String.valueOf(value)));`
	`} else {`
	`tableBuilder.addRow(i, (int) (Math.random() * 10));`
	`}`
	`}`
	`System.out.println(tableBuilder.build());`

Выход	`| Index | Random |`
	`| -----:| ------ |`
	`| 1 | **8** |`
	`| 2 | 2 |`
	`| 3 | 3 |`
	`| 4 | **6** |`
	`| 5 | 3 |`