**МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет Математики и Информатики**

**Департамент Информатики**

JAVA SCRIPT

Индивидуальная работа № 1

Проверил: Нартя Никита

Выполнил: Главчева Марина, IA2303

Кишинев, 2024

# Теоретическая часть

Модуль **fs** в JavaScript (Node.js) представляет собой встроенный модуль для работы с файловой системой. Он предоставляет функции для чтения и записи файлов, создания и удаления каталогов, а также другие операции, связанные с файлами и каталогами.

# В JavaScript класс - это шаблон, который используется для создания объектов. Классы предоставляют способ организации кода и данных в объектно-ориентированном стиле. Введение классов в ECMAScript 2015 (ES6) упрощает написание и понимание кода, особенно для тех, кто привык к объектно-ориентированному программированию (ООП) из других языков, таких как Java или Python.

# Конструктор класса - это специальный метод в JavaScript, который вызывается при создании нового объекта на основе этого класса. Он используется для инициализации нового объекта и установки начальных значений свойств.

# В классах JavaScript конструктор обозначается ключевым словом constructor(). Этот метод выполняется автоматически при создании нового экземпляра класса с помощью оператора new.

# Задача

Создайте консольное приложение для анализа транзакций.

# Цель

1. Создайте класс TransactionAnalyzer для обработки транзакций.
2. Класс должен иметь методы для анализа транзакций, описанные ниже.
3. Конструктор класса будет принимать все транзакции в качестве аргумента.
4. Добавьте методы для добавления новой транзакции и получения списка всех транзакций.
   1. addTransaction()
   2. getAllTransaction()

### Транзакция

1. Каждая транзакция должна быть представлена отдельным объектом, содержащим все необходимые данные.
2. В каждую транзакцию необходимо добавить метод string(), который будет возвращать строковое представление транзакции в формате JSON [extra].

const transaction = {

transaction\_id: "1",

transaction\_date: "2019-01-01",

transaction\_amount: "100.00",

transaction\_type: "debit",

transaction\_description: "Payment for groceries",

merchant\_name: "SuperMart",

card\_type: "Visa",

}

### Анализ транзакций

1. Реализуйте методы для обработки данных о транзакциях.
   1. Метод getUniqueTransactionType().
      1. Возвращает массив всевозможных типов транзакций (например, ['debit', 'credit']).
      2. Используйте Set() для выполнения задания.
   2. Метод calculateTotalAmount().
      1. Рассчитывает общую сумму всех транзакций.
   3. Метод calculateTotalAmountByDate(year, month, day).
      1. Вычисляет общую сумму транзакций за указанный год, месяц и день.
      2. Параметры year, month и day являются необязательными.
      3. В случае отсутствия одного из параметров, метод производит расчет по остальным.
   4. Метод getTransactionByType(type).
      1. Возвращает транзакции указанного типа (debit или credit).
   5. Метод getTransactionsInDateRange(startDate, endDate)
      1. Возвращает транзакции, проведенные в указанном диапазоне дат от startDate до endDate.
   6. Метод getTransactionsByMerchant(merchantName).
      1. Возвращает транзакции, совершенные с указанным торговым местом или компанией.
   7. Метод calculateAverageTransactionAmount().
      1. Возвращает среднее значение транзакций.
   8. Метод getTransactionsByAmountRange(minAmount, maxAmount).
      1. Возвращает транзакции с суммой в заданном диапазоне от minAmount до maxAmount.
   9. Метод calculateTotalDebitAmount() (у которых тип debit).
      1. Вычисляет общую сумму дебетовых транзакций.
   10. Метод findMostTransactionsMonth().
       1. Возвращает месяц, в котором было больше всего транзакций.
   11. Метод findMostDebitTransactionMonth()
       1. Возвращает месяц, в котором было больше дебетовых транзакций.
   12. Метод mostTransactionTypes()
       1. Возвращает каких транзакций больше всего.
       2. Возвращает debit, если дебетовых.
       3. Возвращает credit, если кредитовых.
       4. Возвращает equal, если количество равно.
   13. Метод getTransactionsBeforeDate(date)
       1. Возвращает транзакции, совершенные до указанной даты.
   14. Метод findTransactionById(id).
       1. Возвращает транзакцию по ее уникальному идентификатору.
   15. Метод mapTransactionDescriptions()
       1. Возвращает новый массив, содержащий только описания транзакций.

# Практическая часть

const fs = require('fs'); //модуль fs используется для чтения и записи файлов

class TransactionAnalyzer { //Классы - это шаблоны для создания объектов с определенными свойствами и методами.

    constructor(transactions) { //создании нового объекта класса ,аргумент

        this.transactions = transactions; //что мы сохраняем переданный массив транзакций внутри объекта,

    }

    // Метод для добавления новой транзакции

    addTransaction(transaction) {

        this.transactions.push(transaction); //Метод push() используется для добавления новых элементов в массив.

    }

    // Метод для получения всех транзакций

    getAllTransactions() {

        return this.transactions; //возращает массив всех транзакций

    }

    // Метод  который возвращает массив уникальных типов транзакций

    getUniqueTransactionType() {

        const uniqueTypes = new Set(); // Объект Set позволяет хранить только уникальные значения, отбрасывая дубликаты.

        // Проходим по всем транзакциям и добавляем их типы в Set

        for (const transaction of this.transactions) {

            uniqueTypes.add(transaction.transaction\_type);//Для каждой транзакции он добавляет ее тип

        }

        // Преобразуем Set в массив и возвращаем его

        return Array.from(uniqueTypes); //Эта строка преобразует объект Set uniqueTypes в массив с помощью метода Array.from()

    }

    //Метод предназначен для вычисления общей суммы всех транзакций

    calculateTotalAmount() {

        let totalAmount = 0; // Переменная для хранения общей суммы

        for (const transaction of this.transactions) {

            totalAmount += transaction.transaction\_amount; // Добавляем сумму каждой транзакции к общей сумме

        }

        return totalAmount; // Возвращаем общую сумму

    }

    //Метод предназначен для вычисления общей суммы всех транзакций на определенную дату

    calculateTotalAmountByDate(year, month, day) {

        let totalAmount = 0; // Переменная для хранения общей суммы

        // Проходим по всем транзакциям

        for (const transaction of this.transactions) {

            const transactionDate = new Date(transaction.transaction\_date); //специальный конструктор в JavaScript для создания объектов Date.

            // Проверяем, соответствует ли текущая транзакция указанным параметрам даты

            if (transactionDate.getFullYear() === year &&

                transactionDate.getMonth() + 1 === month &&

                transactionDate.getDate() === day) {

                totalAmount += transaction.transaction\_amount; // Если да, добавляем сумму транзакции к общей сумме

            }

        }

        return totalAmount; // Возвращаем общую сумму

    }

    //Метод предназначен для получения всех транзакций определенного типа

    getTransactionByType(type) {

        // Фильтруем массив транзакций, оставляя только те, у которых тип совпадает с указанным

        return this.transactions.filter(transaction => transaction.transaction\_type === type); //filter(): Это метод массива который создает новый массив, содержащий все элементы исходного массива,

    }

    //Метод предназначен для получения всех транзакций, которые находятся в заданном диапазоне дат.

    getTransactionsInDateRange(startDate, endDate) {

        // Преобразуем даты начала и конца периода в объекты типа Date

        const startDateObj = new Date(startDate);

        const endDateObj = new Date(endDate);

        // Фильтруем массив транзакций, оставляя только те, чья дата входит в указанный диапазон

        return this.transactions.filter(transaction => {

            const transactionDate = new Date(transaction.transaction\_date);

            return transactionDate >= startDateObj && transactionDate <= endDateObj;

        });

    }

    getTransactionsByMerchant(merchantName) {

        // Фильтруем массив транзакций, оставляя только те, у которых название торгового места или компании совпадает с указанным

        return this.transactions.filter(transaction => transaction.merchant\_name === merchantName);

    }

    //предназначен для вычисления средней суммы транзакций.

    calculateAverageTransactionAmount() {

        // Если нет транзакций, возвращаем 0

        if (this.transactions.length === 0) {

            return 0;

        }

        // Суммируем все суммы транзакций

        let totalAmount = 0;

        for (let i = 0; i < this.transactions.length; i++) {

            totalAmount += this.transactions[i].transaction\_amount;

        }

        // Вычисляем среднее значение

        return totalAmount / this.transactions.length;

    }

    //Метод для получения списка транзакций, сумма которых находится в определенном диапазоне.

    getTransactionsByAmountRange(minAmount, maxAmount) {

        // Фильтруем массив транзакций, оставляя только те, с суммой в заданном диапазоне

        return this.transactions.filter(transaction => {

            const amount = transaction.transaction\_amount;

            return amount >= minAmount && amount <= maxAmount;

        });

    }

    //Метод для вычисления общей суммы дебетовых транзакций

    calculateTotalDebitAmount() {

        // Фильтруем массив транзакций, оставляя только дебетовые

        const debitTransactions = this.transactions.filter(transaction => transaction.transaction\_type === 'debit');

        // Суммируем суммы всех дебетовых транзакций

        let total = 0;

        for (let i = 0; i < debitTransactions.length; i++) {

            total += debitTransactions[i].transaction\_amount;

        }

        return total;

    }

    //Метод для поиска месяца с наибольшим количеством транзакций.

    findMostTransactionsMonth() {

        // Создаем объект для хранения количества транзакций по месяцам

        const transactionsByMonth = {};

        // Проходим по всем транзакциям и подсчитываем количество транзакций для каждого месяца

        this.transactions.forEach(transaction => {

            const month = new Date(transaction.transaction\_date).getMonth() + 1;

            transactionsByMonth[month] = (transactionsByMonth[month] || 0) + 1; //увеличивает счетчик для этого месяца в объекте

        });

        // Находим месяц с наибольшим количеством транзакций

        let mostTransactionsMonth;

        let maxTransactions = 0;

        for (const month in transactionsByMonth) {

            if (transactionsByMonth[month] > maxTransactions) {

                mostTransactionsMonth = month;

                maxTransactions = transactionsByMonth[month];

            }

        }

        return mostTransactionsMonth;

    }

     //Метод для поиска месяца с наибольшим количеством дебетовых транзакций.

    findMostDebitTransactionMonth() {

        // Создаем объект для хранения количества дебетовых транзакций по месяцам

        const debitTransactionsByMonth = {};

        // Проходим по всем транзакциям и подсчитываем количество дебетовых транзакций для каждого месяца

        this.transactions.forEach(transaction => {

            if (transaction.transaction\_type === 'debit') {

                const month = new Date(transaction.transaction\_date).getMonth() + 1;

                debitTransactionsByMonth[month] = (debitTransactionsByMonth[month] || 0) + 1;

            }

        });

        // Находим месяц с наибольшим количеством дебетовых транзакций

        let mostDebitTransactionMonth;

        let maxDebitTransactions = 0;

        for (const month in debitTransactionsByMonth) {

            if (debitTransactionsByMonth[month] > maxDebitTransactions) {

                mostDebitTransactionMonth = month;

                maxDebitTransactions = debitTransactionsByMonth[month];

            }

        }

        return mostDebitTransactionMonth;

    }

    mostTransactionTypes() {

        // Счетчики для дебетовых и кредитовых транзакций

        let debitCount = 0;

        let creditCount = 0;

        // Подсчет количества дебетовых и кредитовых транзакций

        this.transactions.forEach(transaction => {//Это функция перебора элементов массива функцию один раз для каждого элемента массива.

            if (transaction.transaction\_type === 'debit') {

                debitCount++;

            } else if (transaction.transaction\_type === 'credit') {

                creditCount++;

            }

        });

        // Возвращаем результат на основе подсчетов

        if (debitCount > creditCount) {

            return 'debit';

        } else if (debitCount < creditCount) {

            return 'credit';

        } else {

            return 'equal';

        }

    }

    getTransactionsBeforeDate(date) {

        return this.transactions.filter(transaction => new Date(transaction.transaction\_date) < new Date(date));

    }

    findTransactionById(id) {

        return this.transactions.find(transaction => transaction.transaction\_id === id);

    }

    mapTransactionDescriptions() {

        return this.transactions.map(transaction => transaction.transaction\_description);

    }

}

if (!fs.existsSync('transactions.json')) {

    console.error('File transactions.json does not exist.');

    process.exit(1); // Выход из программы с кодом ошибки

}

const transactionsData = fs.readFileSync('transactions.json', 'utf8');

const transactions = JSON.parse(transactionsData);

// Создание объекта TransactionAnalyzer

const analyzer = new TransactionAnalyzer(transactions);

console.log(analyzer.getUniqueTransactionType());

console.log(analyzer.calculateTotalAmount());

console.log(analyzer.calculateTotalAmountByDate(2019, 1, 1));

//console.log(analyzer.getTransactionByType('debit'));

console.log(analyzer.getTransactionsInDateRange(new Date(2019, 1, 4), new Date(2019, 1, 5)));

console.log(analyzer.getTransactionsByMerchant("FashionStoreXYZ"));

console.log(analyzer.calculateAverageTransactionAmount());

console.log(analyzer.getTransactionsByAmountRange(100, 200));

console.log(analyzer.calculateTotalDebitAmount());

console.log(analyzer.findMostTransactionsMonth());

console.log(analyzer.findMostDebitTransactionMonth());

console.log(analyzer.getTransactionsBeforeDate(new Date(2019, 0, 5)));

console.log(analyzer.findTransactionById("5"));

console.log(analyzer.mapTransactionDescriptions());

Краткое описание особенностей реализации:

# 1*. Использование модуля `fs`:*

# - Модуль `fs` используется для работы с файловой системой в Node.js.

# - В данном коде он используется для чтения файла `transactions.json`, содержащего данные о транзакциях.

# 2. *Класс `TransactionAnalyzer`:*

# - Класс представляет собой шаблон для анализа транзакций.

# - Методы класса позволяют выполнить различные операции с данными о транзакциях, такие как добавление новой транзакции, получение всех транзакций, вычисление общей суммы транзакций и другие.

# 3. *Использование Set для уникальных значений:*

# - В методе `getUniqueTransactionType()` используется объект Set для хранения уникальных типов транзакций.

# - Set позволяет автоматически удалять дубликаты значений, обеспечивая уникальность элементов.

# 4. *Обработка дат:*

# - Методы `calculateTotalAmountByDate()` и `getTransactionsInDateRange()` используют объекты типа `Date` для работы с датами.

# - Даты в транзакциях преобразуются в объекты `Date`, чтобы можно было сравнивать их с указанными датами.

# 5. *Обработка исключений:*

# - При чтении файла `transactions.json` используется обработка исключений для обнаружения ошибок чтения или парсинга файла.

# - Если возникает ошибка, выводится сообщение об ошибке, и программа завершается с кодом ошибки.

# 6. *Использование методов класса:*

# - Создается экземпляр класса `TransactionAnalyzer` для анализа данных о транзакциях.

# - Вызываются различные методы класса для выполнения операций с данными о транзакциях.

# - Результаты анализа выводятся в консоль для дальнейшего анализа и использования.

# Ответы на контрольные вопросы

1. Какие примитивные типы данных существуют в JavaScript?

String (Строка): Представляет последовательность символов, заключенную в кавычки (одинарные или двойные).

Number (Число): Представляет числовые значения. Может быть целым или числом с плавающей точкой.

Boolean (Логическое значение): Представляет логическое значение true или false.

null: Специальное значение, которое представляет отсутствие значения или "ничего".

undefined: Специальное значение, которое представляет отсутствие значения или "неопределенное".

BigInt: Представляет целые числа произвольной длины. Значения этого типа записываются с помощью суффикса n.

Symbol: Представляет уникальный и неизменяемый идентификатор. Создается с помощью функции Symbol(). Каждый символ уникален.

1. Какие методы массивов вы использовали для обработки и анализа данных в вашем приложении, и как они помогли в выполнении задачи?

push(): Добавляет новый элемент в конец массива. В вашем приложении используется для добавления новой транзакции в массив транзакций.

filter(): Создает новый массив, содержащий элементы, для которых переданная функция возвращает true. В вашем приложении используется для фильтрации транзакций по различным критериям, таким как тип транзакции, дата и т.д.

map(): Создает новый массив, содержащий результат вызова указанной функции для каждого элемента массива. В вашем приложении используется для получения массива описаний транзакций.

reduce(): Применяет функцию к аккумулятору и каждому значению массива (слева направо), чтобы преобразовать его в единое значение. В вашем приложении используется для вычисления общей суммы транзакций и других агрегированных значений.

forEach(): Вызывает функцию один раз для каждого элемента в массиве. В вашем приложении используется для обхода всех транзакций и выполнения операций над ними.

Эти методы массивов позволяют эффективно обрабатывать данные и извлекать необходимую информацию из массива транзакций. Они помогают в выполнении различных задач, таких как нахождение уникальных типов транзакций, вычисление общей суммы транзакций и т.д.

1. В чем состоит роль конструктора класса?

Роль конструктора класса заключается в инициализации новых объектов, созданных на основе этого класса. Конструктор выполняет следующие основные задачи:

Инициализация свойств объекта: Конструктор устанавливает начальные значения свойств объекта. Он принимает аргументы, которые могут быть использованы для инициализации свойств объекта.

Резервирование памяти: При создании нового объекта конструктор резервирует необходимое количество памяти для его хранения в памяти компьютера.

Вызов родительского конструктора (если есть наследование): В случае, если класс наследует от другого класса, конструктор класса может вызывать конструктор родительского класса для инициализации свойств, унаследованных от родителя.

Исполнение дополнительных операций: Конструктор может выполнять любые другие необходимые операции при создании объекта, например, проверку входных данных или выполнение сложной инициализации.

1. Каким образом вы можете создать новый экземпляр класса в JavaScript?

В JavaScript новый экземпляр класса можно создать с помощью ключевого слова new, за которым следует вызов конструктора класса. Это выглядит следующим образом:

const myObject = new MyClass();

Где MyClass - это имя класса, а myObject - переменная, которая будет содержать новый экземпляр этого класса.

При создании экземпляра класса может потребоваться передача аргументов конструктору класса, если конструктор класса определен с параметрами. Например:

const person = new Person("John", 30);

Здесь Person - это класс, а "John" и 30 - это аргументы, передаваемые в конструктор класса Person.

После создания экземпляра класса, можно использовать его свойства и методы для выполнения различных операций.

# Вывод

В ходе работы был разработан класс `TransactionAnalyzer` для обработки транзакций. Этот класс предоставляет удобные методы для анализа данных о транзакциях, таких как вычисление общей суммы транзакций, поиск уникальных типов транзакций, анализ транзакций по датам и другие.

**Ссылка на репозиторий Git:**

# Список использованных источников

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Classes>

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global\_Objects/Set/Set