МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра технологий программирования

Кот Максим Артурович
Ларионова Анастасия Александровна
Щербаченя Михаил Евгеньевич

Технологии программирования

Отчет по лабораторной работе №6

«Основы JavaScript. Синтаксис языка JavaScript. Функции, массивы, строки, сортировки. ES6 классы. Реализация функционала в консоли.»

студентов 3 курса 8 группы

Преподаватель
Терех Владимир Сергеевич

Минск, 2024

Описание предметной области

АИС «ЖД вокзал покупка билетов на поезд» — это система, которая предназначена для автоматизации процесса выбора билетов на пригородные, междугородние и международные рейсы поездов, упрощения процесса их бронирования, оплаты и возврата. Приложение автоматизирует процесс покупки билетов как в онлайн-режиме, так и через кассу. Кроме того, система предоставляет пользователям возможность зарегистрироваться и просмотреть доступные рейсы. Приложение включает механизм взаимодействия с банковской системой для обработки оплаты и возврата средств.

Мы познакомились с основами JavaScript, познакомились с синтаксисом классов JavaScript, после чего перенесли данные приложения из HTML в JavaScript (согласно индивидуальному заданию). Научились запускать и делать Debug JavaScript-кода в консоли браузера. Реализовали основные методы для работы приложения в классе коллекции (модели MVC).

Пункт 1. Перенос данных в JavaScript

Мы создали массив для хранения информации о билетах, где каждый объект соответствует классу Ticket. Поля, которые используются в коде:

- start_place — начальная станция маршрута.
- end_place — конечная станция маршрута.
- start_date — дата и время отправления.
- finish_date — дата и время прибытия.
- cost — стоимость билета.
- type — тип вагона (например, "Купе", "Плацкарт").
- purchased — статус покупки (логическое значение).

Пример массива:

```
class Ticket {
  constructor(start_place, end_place, start_date, finish_date, cost, type, purchased) {
    if (!Ticket.validateProperties([start_place, end_place, start_date, finish_date, cost, type, purchased])) {
      throw new Error("Fields should be defined.");
    }
```

```
    this.start_place = start_place;
    this.end_place = end_place;
    this.start_date = start_date;
    this.finish_date = finish_date;
    this.cost = cost;
    this.type = type;
    this.purchased = purchased;
  }

  static validateProperties(requiredFields) {
    for (const field of requiredFields) {
      if (field === null || field === undefined) {
        return false;
      }
    }
    return true;
  }
}

const ticketsArray = [
  new Ticket("Москва", "Владивосток", new Date(2025, 5, 10), new Date(2025, 5, 12), 1200, "Купе", true),
  new Ticket("Санкт-Петербург", "Ростов-на-Дону", new Date(2025, 6, 15), new Date(2025, 6, 16), 800, "Сидячий", false),
  // ... добавлено 20 объектов
];
```

Пункт 2. Создание класса с методами для работы с массивом

Создан класс TicketInfManager, содержащий массив билетов в приватном поле #objInfArray и методы для работы с ним:

Метод getObjs

Позволяет получать отсортированный массив объектов с учетом фильтрации и пагинации.

```
getObjs(skip = 0, top = 0, filterConfig={}) {
	let filteredObjs = this.#objInfArray;
	for (let key in filterConfig) {
	filteredObjs = filteredObjs.filter(item => item.Obj[key] === filterConfig[key]);
	}
	if (skip < 0 || skip > filteredObjs.length) {
		throw new Error("Skip out of range!");
	}
	if (top < 0 || top > filteredObjs.length - skip) {
		throw new Error("Top out of range!");
	}
	if (top == 0) {
		return filteredObjs.slice(skip, filteredObjs.length - skip);
	}
	return filteredObjs.slice(skip, skip + top);
}
```

Пример вызова:

```
console.log(manager.getObjs(0, 5, { start_place: "Москва" }));
```

Метод getObj

Позволяет получить объект по его id.

```
getObj(id) {
	if (!id instanceof Number || id > this.#size || id < 1) {
		throw new Error("Incorrect ID.");
	}
	const found = this.#objInfArray.find(item => item.id === id);
	if (found) {
		return found;
	} else {
		console.log("Object not found");
		return false;
	}
}
```

Пример вызова:

```
console.log(manager.getObj(3));
```

Метод addObj

Добавляет новый объект в массив, предварительно проверяя его валидность.

```
addObj(Obj) {
  if (!(Obj instanceof Ticket)) {
    throw new Error("Only class Ticket objects can be added.");
  }
… //другие проверки
  const id = this.#generateId();
  this.#objInfArray.push({ id, Obj });
  return id;
}
```

Пример добавления:

```
const newTicket = new Ticket("Казань", "Волгоград", new Date(2025, 7, 20), new Date(2025, 7, 22), 1000, "Плацкарт", true);

manager.addObj(newTicket);
```

Метод editObj

Редактирует объект по его id. Допускается изменение всех полей, кроме id, author и createdAt.

```
editObj(id, Obj) {

… //проверки

  const index = this.#objInfArray.findIndex(item => item.id === id);

  const fieldsToUpdate = ["start_place", "end_place", "start_date", "finish_date", "cost", "type", "purchased"];

  fieldsToUpdate.forEach(field => {

    if (Obj[field] !== undefined) {

      this.#objInfArray[index].Obj[field] = Obj[field];

    }

  });

  return true;

}
```

Пример изменения:

```
manager.editObj(3, { cost: 950, purchased: false });
```

Метод removeObj

Удаляет объект по его id.

```
removeObj(id) {
                let old_Obj = this.getObj(id);
                if (!old_Obj) {
                        return false;
                }
```

```
        this.#objInfArray = this.#objInfArray.filter(item => item.id !== id);
        return true;
}
```

Пример удаления:

```
manager.removeObj(5);
```

Пункт 3. Приватные поля и методы

В классе TicketInfManager приватные поля и методы реализованы в соответствии с современной практикой, используя синтаксис #, который был добавлен в стандарт JavaScript ES2020.

Приватные поля

1. #objInfArray — массив, используемый для хранения информации об объектах.
    - Это внутреннее поле, недоступное извне класса.
    - Защищает данные от прямого изменения пользователями.
2. #size — числовой счётчик, используемый для генерации уникальных идентификаторов объектов.
    - Начальное значение — 0.
    - Инкрементируется при вызове метода #generateId.

Приватный метод

1. #generateId() — вспомогательный метод, который увеличивает значение счётчика #size и возвращает новый уникальный идентификатор.
    - Этот метод скрыт от внешнего доступа, что предотвращает случайное или намеренное нарушение логики идентификации объектов.

```
class TicketInfManager {
  #objInfArray = []; // Приватное поле для хранения данных
  #size = 0;         // Приватное поле-счётчик

  constructor() {}
```

```
  #generateId() {    // Приватный метод для генерации уникальных ID
    this.#size = this.#size + 1;
    return this.#size;
  }
}
```

Таким образом, приватные поля и методы обеспечивают инкапсуляцию данных и реализацию вспомогательной логики, недоступной извне, что соответствует требованиям задания.

Пункт 4. Дополнительные методы

Метод addAll

Добавляет несколько билетов одновременно, при этом проверяет валидность каждого объекта.

```
addAll(objs) {
  const invalidObjs = objs.filter(obj => !(obj instanceof Ticket));
  if (invalidObjs.length > 0) {
    throw new Error("Some objects are invalid.");
  }
  objs.forEach(obj => this.addObj(obj));
}
```

Метод clear

Очищает коллекцию билетов.

```
clear() {
  this.#objInfArray = [];
}
```

Пункт 5. Тестирование

Для проверки корректной работы методов класса TicketInfManager было проведено тестирование основных операций. Приведённый ниже код демонстрирует, как тестировались функциональные возможности системы:

1. Добавление нового объекта

Сначала сгенерировалось 20 случайных объектов типа Ticket, которые добавлялись в массив через метод addObj:

```
for (let i = 0; i < 20; i++) {
   const start_place = startPlaces[Math.floor(Math.random() * startPlaces.length)];
   const end_place = endPlaces[Math.floor(Math.random() * endPlaces.length)];
   const start_date = new Date(2025, Math.floor(Math.random() * 11), Math.floor(Math.random() * 28), Math.floor(Math.random() * 24), Math.floor(Math.random() * 60));
   const finish_date = new Date(start_date);
   finish_date.setHours(finish_date.getHours() + Math.floor(Math.random() * 5) + 1);
   const cost = Math.floor(Math.random() * (1500 - 500) + 500);
   const type = ticketTypes[Math.floor(Math.random() * ticketTypes.length)];
   const purchased = Math.random() > 0.5;
   const ticket = new Ticket(start_place, end_place, start_date, finish_date, cost, type, purchased);
   manager.addObj(ticket);
}
```

После добавления объектов проводилась проверка их корректного сохранения в массиве.

2. Изменение информации в объекте

Метод editObj протестировали, заменяя данные конкретного объекта новым билетом:

```
console.log(manager.editObj(3, new Ticket("Заслоново", "Никосия", new Date(2025, 7, 2, 16, 29), new Date(2025, 7, 6, 12, 0), 800, "CB", true)));
console.log(manager.getObj(3));
```

Сначала объект с индексом 3 заменялся новым, затем проверялось, что изменения успешно применились.

3. Удаление объекта по id

Для проверки метода removeObj удаляли объект с определённым id и затем пытались его получить:

console.log(manager.removeObj(13));

console.log(manager.getObj(13));

Метод корректно удалял указанный объект, а последующий вызов возвращал undefined или аналогичное значение.

4. Получение отсортированного массива с фильтрацией и пагинацией

Тестирование метода getObjs включало проверку пагинации, сортировки и фильтрации:

console.log(manager.getObjs(2, 4)); // Получение объектов с индексами 2, 3, 4.

console.log(manager.getObjs(0, 0, { start_place: "Москва" })); // Фильтрация по полю start_place.

Метод корректно возвращал массив, соответствующий заданным условиям.

Вывод

Тестирование показало, что реализованные методы (addObj, editObj, removeObj, getObjs) функционируют корректно. CRUD-система успешно обрабатывает создание, изменение, удаление и получение объектов, включая фильтрацию, сортировку и пагинацию.

Пункт 6. Заключение

Таким образом рамках данной лабораторной работы был создан класс TicketInfManager, который выполняет роль управляющего объекта для хранения и обработки данных о билетах. Основные этапы разработки включали:

1. **Создание объекта-массива для хранения данных о билетах.**
    - Для хранения информации о билетах реализовано приватное поле #objInfArray, которое представляет собой массив объектов. Это позволяет централизованно управлять всеми данными, связанными с билетами, и гарантировать их защиту от внешних воздействий.

2. **Реализация валидации данных.**
    - Валидация данных выполняется с использованием приватных методов и внутренней логики класса. Такие методы позволяют проверить корректность введённых данных перед добавлением или изменением информации о билетах. Это обеспечивает консистентность данных и предотвращает ошибки на уровне пользователя.
3. **Создание методов для управления данными (CRUD-система):**
    - **Создание новых объектов:** реализован метод добавления новых билетов в массив с автоматической генерацией уникальных идентификаторов. Это обеспечивает структурированное добавление информации.
    - **Чтение (поиск):** разработан метод поиска объектов по определённым критериям, что позволяет быстро находить нужные данные в массиве.
    - **Обновление (замена):** реализован метод изменения информации в объекте, что позволяет редактировать уже существующие данные.
    - **Удаление объектов:** создан метод для удаления записей о билетах по идентификатору или другому критерию. Это упрощает управление данными и позволяет эффективно освобождать ресурсы.

В результате получилась полноценная система для работы с информацией о билетах, включающая функции создания, чтения, обновления и удаления объектов. Применение приватных полей и методов обеспечивает безопасность и инкапсуляцию, а использование валидаторов и уникальных идентификаторов гарантирует целостность и корректность данных.

Данная реализация может быть легко расширена за счёт добавления дополнительных методов или логики для работы с другими аспектами данных о билетах.

Все требования лабораторной работы выполнены.