Домашнее задание к лекции 3.2 «Иерархия прототипов и наследование»

Перед началом работы

1.Активируйте строгий режим соответствия.

2.Добавьте в редактор следующий блок кода:

```
class Calendar {
  constructor(now = new Date()) {
    this.now = now;
  }

  setDate(now) {
    this.now = now;
  }

  get today() {
    return this.now.toLocaleString('ru-Ru');
  }
}

class PaymentTerminal {
  constructor(title, calendar) {
    this.title = title;
    this.calendar = calendar;
  }

  get status() {
    return this.isActive ? 'работает' : 'недоступен';
  }

  get isActive() {
    return this.checkActive();
  }

  checkActive() {
    return false;
  }
}
```

```
class RegistrationError extends Error {
  constructor(field = null) {
    super(`Ошибка в поле ${field}`);
    this.field = field;
  }
}

class NotValidEmailRegistrationError extends RegistrationError {
  constructor(field, email) {
    super(field);
    this.email = email;
  }
}

class NotUniqueRegistrationError extends RegistrationError {
  constructor(field, value) {
    super(field);
    this.value = value;
  }
}

class NotSameRegistrationError extends RegistrationError {}

function isValidEmail(email) {
  return /^\w+(\.\w+)*@\w+(\.\w+)+$/i.test(email);
}

function isUniqueLogin(login) {
  return !['admin', 'boss'].includes(login);
}

function checkPassword(original, copy) {
  return original === copy;
}

function registerNewUser(data) {
  if (!isValidEmail(data.email)) {
    throw new NotValidEmailRegistrationError('Адрес электронной почты',
data.email);
  }
```

```
  if (!isUniqueLogin(data.login)) {
    throw new NotUniqueRegistrationError('Логин', data.login);
  }
  if (!checkPassword(data.password, data.passwordCopy)) {
    throw new NotSameRegistrationError('Пароль');
  }
}
```

Задача № 1. Работа с датой

Для решения различных логистических задач нам нужно усовершенствовать объект Date, добавив туда ряд полезных функций.
Создайте класс SpaceDate, который, кроме базового функционала Date, будет иметь удобный функционал копирования, получения следующего дня и другие функции.
Описание конструктора и экземпляра SpaceDate

Конструктор

Работает точно так же, как и оригинальный конструктор Date.

Экземпляр

Помимо всех методов экземпляра Date, должен иметь следующие методы:
copy — возвращает объект SpaceDate с той же датой и временем, что и исходный;
getNextDate — возвращает объект SpaceDate с тем же временем, но со следующим календарным днём;
getPrevDate — возвращает объект SpaceDate с тем же временем, но с предыдущим календарным днём;
getDayBeginning — возвращает объект SpaceDate с той же датой, но время установлено в 00:00:00.000;
getDayEnd — возвращает объект SpaceDate с той же датой, но время установлено в 23:59:59.999.

Пример использования

```
let dateOriginal = new SpaceDate(2017, 1, 22);
let dateCopy = dateOriginal.copy();
dateCopy.setYear(2022);
console.log(`Оригинальная дата: ${dateOriginal.toLocaleDateString('ru-Ru')}`);
console.log(`Измененная копия: ${dateCopy.toLocaleDateString('ru-Ru')}`);

let orderDate = new SpaceDate(2017, 2, 10);
let deliveryDate = orderDate.getNextDate();
console.log(`Дата заказа: ${orderDate.toLocaleDateString('ru-Ru')}`);
console.log(`Дата доставки: ${deliveryDate.toLocaleDateString('ru-Ru')}`);
```

```
let supplyDate = new SpaceDate(2017, 3, 3);
let requestDate = supplyDate.getPrevDate();
console.log(`Дата поставки: ${supplyDate.toLocaleDateString('ru-Ru')}`);
console.log(`Дата заявки поставщику: ${requestDate.toLocaleDateString('ru-Ru')}`);

let someDate = new SpaceDate(2017, 2, 10, 12, 44);
let from = someDate.getDayBeginning();
let to = someDate.getDayEnd();
console.log(`В любое время с ${from.toLocaleString('ru-Ru')} по ${to.toLocaleString('ru-Ru')}`);
```

Если все реализовано верно, вы получите такой вывод:

```
Оригинальная дата: 22.02.2017
Измененная копия: 22.02.2022
Дата заказа: 10.03.2017
Дата доставки: 11.03.2017
Дата поставки: 03.04.2017
Дата заявки поставщику: 02.04.2017
В любое время с 10.03.2017, 0:00:00 по 10.03.2017, 23:59:59
```

Процесс реализации

1. Создайте класс `SpaceDate`, унаследовав его от `Date`.
2. Прописывать конструктор не нужно, пусть используется конструктор `Date`. Он нам полностью подходит.
3. Добавьте метод copy. Учтите, что создаваться должен экземпляр `SpaceDate`.
4. Добавьте методы `getNextDate` и `getPrevDate`. Оригинальная дата изменяться при их вызове не должна. По возможности подумайте, как избавиться от дублирования кода в них.
5. Добавьте методы `getDayBeginning` и `getDayEnd`. Оригинальная дата изменяться при их вызове не должна.

Проверьте работу вашего кода на примере использования. Также протестируйте его, используя свои примеры.

Задача № 2. Терминалы оплаты

Мы решили показывать на сайте информацию о состоянии терминалов оплаты, и нам нужно реализовать различные графики работы этих терминалов. Часть терминалов работает только в будние дни, часть — круглосуточно, и часть — круглосуточно, за исключением праздников.

Наш ведущий разработчик предложил использовать механизм наследования для того, чтобы функционал можно было легко расширять, и при этом проверки в коде оставались простыми и понятными.

Он уже создал базовый класс `PaymentTerminal`, который берет текущую дату из календаря (так удобнее для тестирования). Базовый терминал всегда недоступен. Вам нужно расширить его возможности, создав классы:
`AllDayPaymentTerminal` — реализовав терминал, который доступен 24/7, или круглосуточно.
`AllDayExceptHolidaysPaymentTerminal` — реализовав терминал, который доступен 24/7, кроме определенных дней в году, которые заданы в третьем аргументе.
`WorkspacePaymentTerminal` — терминал доступен только в будние дни (с понедельника по пятницу) с 8 утра до 18 вечера (в 8:00 терминал доступен, в 18:00 уже недоступен).

Описание конструкторов и экземпляров

Конструкторы

Конструкторы `AllDayPaymentTerminal` и `WorkspacePaymentTerminal` работают точно так же, как `PaymentTerminal`, принимают два аргумента:

1. `title` — название пункта, строка;
2. `calendar` — календарь, из которого нужно получать текущую дату — она в свойстве `now`, объект `Calendar`.

Конструктор `AllDayExceptHolidaysPaymentTerminal` принимает дополнительный третий аргумент:

1. `title` — название пункта, строка;
2. `calendar` — календарь, из которого нужно получать текущую дату — она в свойстве `now`, объект `Calendar`.
3. `holidays` — список праздничных дней в году, массив. Каждый день представлен объектом со свойствами `date`— число, и `month` — номер месяца, начиная с нуля.

Экземпляры

Должны переопределить в соответствии с их логикой работы метод:

`checkActive` — проверяет, работает ли терминал в настоящий момент. Настоящий момент берется из календаря, свойство `this.calendar.now`, там объект `Date`.

Пример использования

```
const holidays = [
  { date: 11, month: 3 - 1 },
  { date: 23, month: 2 - 1 }
];

const calendar = new Calendar();
const terminals = [
  new WorkspacePaymentTerminal('Терминал в офисе Убербанка', calendar),
  new AllDayPaymentTerminal('Терминал в аэропорту', calendar),
  new AllDayExceptHolidaysPaymentTerminal('Терминал в торговом центре',
    calendar, holidays)
];
```

```
function showTerminals(date) {
  if (date !== undefined) {
    calendar.setDate(date);
  }
  console.log(calendar.today);
  terminals
    .filter(terminal => terminal instanceof PaymentTerminal)
    .forEach(terminal => console.log(`${terminal.title} ${terminal.status}`));
}

showTerminals(new Date(2017, 2 - 1, 23));
showTerminals(new Date(2017, 3 - 1, 11));
showTerminals(new Date(2017, 3 - 1, 14, 18, 1));
showTerminals(new Date(2017, 3 - 1, 14, 8, 3));
```

Если все реализовано верно, вы получите такой вывод:

```
23.02.2017, 0:00:00
Терминал в офисе Убербанка недоступен
Терминал в аэропорту работает
Терминал в торговом центре недоступен
11.03.2017, 0:00:00
Терминал в офисе Убербанка недоступен
Терминал в аэропорту работает
Терминал в торговом центре недоступен
14.03.2017, 18:01:00
Терминал в офисе Убербанка недоступен
Терминал в аэропорту работает
Терминал в торговом центре работает
14.03.2017, 8:03:00
Терминал в офисе Убербанка работает
Терминал в аэропорту работает
Терминал в торговом центре работает
```

Процесс реализации

1. Реализуйте класс `AllDayPaymentTerminal`.
2. Определите метод `checkActive` так, чтобы терминал был доступен круглосуточно в любой день.
3. Реализуйте класс `AllDayExceptHolidaysPaymentTerminal`.
4. Определите метод `checkActive` так, чтобы терминал был недоступен в праздничные дни из аргумента `holidays`конструктора. Постарайтесь сделать код метода максимально понятным и простым.

5. Реализуйте класс `WorkspacePaymentTerminal`.
6. Определите метод `checkActive` так, чтобы терминал был недоступен в субботу и воскресенье, а также в любой день до 8:00 и после 18:00. Постарайтесь сделать код метода максимально понятным и простым.

Проверьте работу вашего кода на примере использования. Также протестируйте его, используя свои примеры.

Задача № 3. Ошибки в форме регистрации

Реализовать функцию `handleRegistration`, которая будет обрабатывать форму, заполненную пользователем, используя функцию `registerNewUser`, и выводить сообщение Пользователь успешно зарегистрирован, если функция не выбросила никаких исключений. Либо сообщения об ошибках, соответствующие брошенному исключению:

- «`test`» не является адресом электронной почты — если брошено исключение `NotValidEmailRegistrationError`, где `test` — введенный адрес электронной почты;
- Пользователь с логином «`boss`» уже зарегистрирован — если брошено исключение `NotUniqueRegistrationError`, где `boss` — введенный логин;
- Введенные пароли не совпадают — если брошено исключение `NotSameRegistrationError`.

Описание функции

Принимает один аргумент:

1. `data` — поля заполненной формы, объекта.

Пример использования

```
const notValidEmailUser = { email: 'test' };
handleRegistration(notValidEmailUser);

const notUniqueLoginUser = { email: 'test@test.co', login: 'boss' };
handleRegistration(notUniqueLoginUser);

const differentPwUser = { email: 'test@test.co', login: 'ivan',
  password: '123', passwordCopy: '456' };
handleRegistration(differentPwUser);

const normalUser = { email: 'test@test.co', login: 'ivan', password: '123',
passwordCopy: '123' };
handleRegistration(normalUser);
```

Если все реализовано верно, вы получите такой вывод:

```
«test» не является адресом электронной почты
Пользователь с логином «boss» уже зарегистрирован
Введенные пароли не совпадают
```

```
Пользователь успешно зарегистрирован
```

Процесс реализации

1. Создайте функцию `handleRegistration`.
2. Вызовите функцию `registerNewUser`, передав туда данные формы.
3. Если `registerNewUser` не бросила исключений, выведите сообщение об успешной регистрации.
4. Перехватите выброшенное исключение.
5. Выведите сообщение об ошибке.

Проверьте работу вашего кода на примере использования. Также протестируйте его, используя свои примеры.