залежить від змісту пам'яті. Більш того, якщо допущено помилку в обчисленні адреси, то може бути знайдена зовсім інша інформація. Асоціативна пам'ять, чи пам'ять, що адресується за змістом, буде доступною за вказівкою заданого змісту. Вміст пам'яті може бути викликано навіть за частковим фрагментом або спотвореним змістом. Асоціативна пам'ять є потрібною при створенні мультимедійних інформаційних баз даних.

7. **Керування.** Розглянемо динамічну систему, задану сукупністю {u(t), y(t)}, де u(t) є вхідним керуючим впливом, а y(t) - виходом системи в момент часу t. В системах керування з еталонною моделлю метою керування є розрахунок такого вхідного впливу u(t), при якому система діє за бажаною траєкторією, яка задана еталонною моделлю. Прикладом є оптимальне керування двигуном.

Незважаючи на переваги нейронних мереж в певних областях над традиційними обчисленнями, існуючі нейромережі не є досконалими рішеннями. Вони навчаються і можуть робити "помилки" [3].

Список літератури

1. *НЕЙРОННІ МЕРЕЖІ: ЇХ ЗАСТОСУВАННЯ, РОБОТА. URL: https://www.poznavayka.org/uk/nauka-i-tehnika-2/neyronni-merezhi-yih-zastosuvannya-robota/ (дата звернення 23.10.2021)*
2. *Інтелектуальні системи автоматизації : монографія / Аврунін О. Г., Владов С. І., Петченко М. В., Семенець В. В., Татарінов В. В., Тельнова Г. В., Філатов В. О., Шмельов Ю. М., Шушляпіна Н. О. – Кременчук : Видавництво «НОВАБУК», 2021.*
3. *Оссовський С. Нейронні мережі для обробки інформації/Станіслав Осовський. Пров. з польського І.Д. Рудинського. - М.: Фінанси та статистика, 2002.*

УДК 004.04

Швець Х. І., студентка
3 курсу спеціальності 122
Горяшин А. С., асистент
кафедри інформаційних технологій

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ КЛАСОВИЙ ПІДХІД В МОВІ ПРОГРАМУВАННЯ JAVASCRIPT

Донецький національний університет імені Василя Стуса, м. Вінниця

Об'єктно-орієнтоване програмування (ООП) — це шаблон проектування ПЗ, що дозволяє вирішувати завдання з погляду об'єктів та його взаємодій. ООП зазвичай реалізується за допомогою класів чи прототипів. Більшість об'єктно-орієнтованих мов (Java, C++, Ruby, Python та ін.) використовують наслідування на основі класів. JavaScript реалізує ОВП через прототипне успадкування. У цій тезі ми розглянемо обидва ці підходи до JS,

обговоримо їх переваги та недоліки, а також запропонуємо альтернативу для розробки більш модульних та масштабованих додатків.

Що таке об'єкт?

Принцип ООП полягає в тому, щоб складати систему з об'єктів, які вирішують прості таски, які складають складну програму. Об'єкт складається з приватних змінних станів та функцій (методів), які працюють із цими станами. У об'єктів є себе (self, this) і поведінка, успадковане від креслення, тобто. класу (класове наслідування) або інших об'єктів (прототипне наслідування).

Успадкування – спосіб сказати, що ці об'єкти схожі на інші за винятком деяких деталей. Успадкування дозволяє нам прискорити розробку за рахунок повторного використання коду.

У класовому ООП класи є кресленнями об'єктів. Об'єкти (або екземпляри) створюються з урахуванням класів. Існує конструктор, який використовується для створення екземпляра класу із заданими властивостями.

Наприклад:

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }
  getFullName() {
    return this.firstName + ' ' + this.lastName
  }
}
```

Малюнок 1 - Приклад класу в мові програмування JS

Тут за допомогою ключового слова classз ES6 ми створюємо клас Personз властивостями **firstName та lastName**, які зберігаються у **this**. Значення властивостей задаються у конструкторі, а доступом до них здійснюється у методі **getFullName**().

Ми створюємо екземпляр класу Personз ім'ям personза допомогою ключового слова new:

```
let person = new Person('Dan', 'Abramov')
person.getFullName() //> "Dan Abramov"

person.firstName //> "Dan"
person.lastName //> "Abramov"
```

Малюнок 2 - Приклад створення екземпляру на основі класу

Загальні принципи ООП:

- *Інкапсуляція*

Інкапсуляція захищає від впливу ззовні внутрішні змінні кожного об'єкта. В ідеалі програма повинна складатися з «острівів об'єктів»: кожен із них зі своїми станами, який передає повідомлення туди й назад. Звучить як хороша ідея в тому випадку, якщо ви створюєте ідеально розподілену систему, але на практиці розробка такої програми складна і вганяє певні рамки.

- *Поліморфізм*

Поліморфізм дозволяє описувати поведінку незалежно від типу даних. В ООП це означає створення класу або прототипу, який може бути адаптований до об'єктів, що працюють з іншими типами даних. Об'єкти, які використовують поліморфний клас/прототип, повинні визначити специфічну для типу даних поведінку, щоб усе запрацювало. Подивимося на приклад.

Припустимо, що ми хочемо створити загальний (поліморфний) об'єкт, який приймає якісь дані та прапор стану як параметри. Якщо стан говорить, що дані валідні (тобто status === true), на даних можна застосувати функцію, результат якої буде повернутий разом із прапором стану. В іншому випадку ми не застосуємо функцію і просто повернемо дані та прапор.

Почнемо зі створення поліморфного об'єкта-прототипу Maybe:

```
function Maybe({data, status}) {
  this.data = data
  this.status = status
}
```

Малюнок 3 *- Створення поліморфного об'єкту*

Maybe- Це обгортка для даних. Щоб обернути їх, ми додали поле status, яке свідчить про валідність даних.

Ми можемо додати до прототипу функцію apply(), яка приймає функцію і застосовує її на даних, якщо статус говорить, що вони валідні:

```
Maybe.prototype.apply = function (f) {
  if(this.status) {
    return new Maybe({data: f(this.data), status: this.status})
  }
  return new Maybe({data: this.data, status: this.status})
}
```

Малюнок 4 *- Додвання до прототипу Maybe функції aplly*

Можемо зробити висновок що, розробникам часто доводиться шукати компроміс між повторним використанням коду та його масштабованістю. Ймовірно, використання класового ООП має сенс для корпоративного програмного забезпечення, оскільки воно не сильно змінюється. Поведінка в ООП чітко прописана в абстрактних класах, але її можна певною мірою налаштувати під час

створення екземплярів класу. Це сприяє кращому повторному використанню коду, що заощаджує розробникам багато часу. Тим не менш, якщо ви очікуєте, що в майбутньому багато разів доведеться доповнювати код і навіть переглядати проект, тоді ОВП у результаті заважатиме продуктивності розробника і код стане нетестованим і сильно пов'язаним із середовищем.

Список літературних джерел :

1. «Басюк Т.М. Основи інформаційних технологій [Текст]: навч. посібн. / Т.М. Басюк, Н.О. Думанський, О.В. Пасічник [нове видання]. – Львів : «Новий Світ – 2000», 2020. – 390, с. ISBN 978-966-418-121-8»

2. «Dr. Alan Kay on the Meaning of "Object-Oriented Programming"". 2003. Retrieved 11 February 2010.»

3. «You Don't Know JS: this & Object Prototypes.» By Kyle Simps