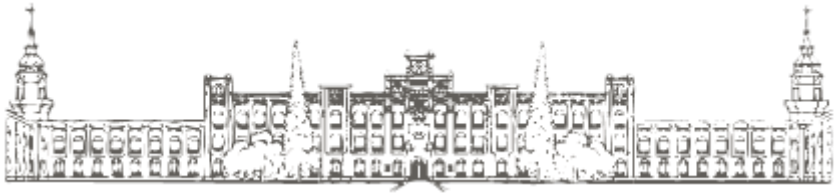

Міністерство освіти і науки України

Національний технічний університет України «КПІ

імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ЗВІТ

лабораторної роботи №2

з дисципліни «Моделювання систем»

Перевірила:

Дифучина О. Ю.

Виконав:

Студент Гр. ІП-01

Пашковський Є. С.

Київ 2023

Завдання

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об'єктно-орієнтованого підходу. **5 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **5 балів.**
3. Створити модель за схемою, представленою на рисунку 2.1. **30 балів.**
4. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **10 балів.**

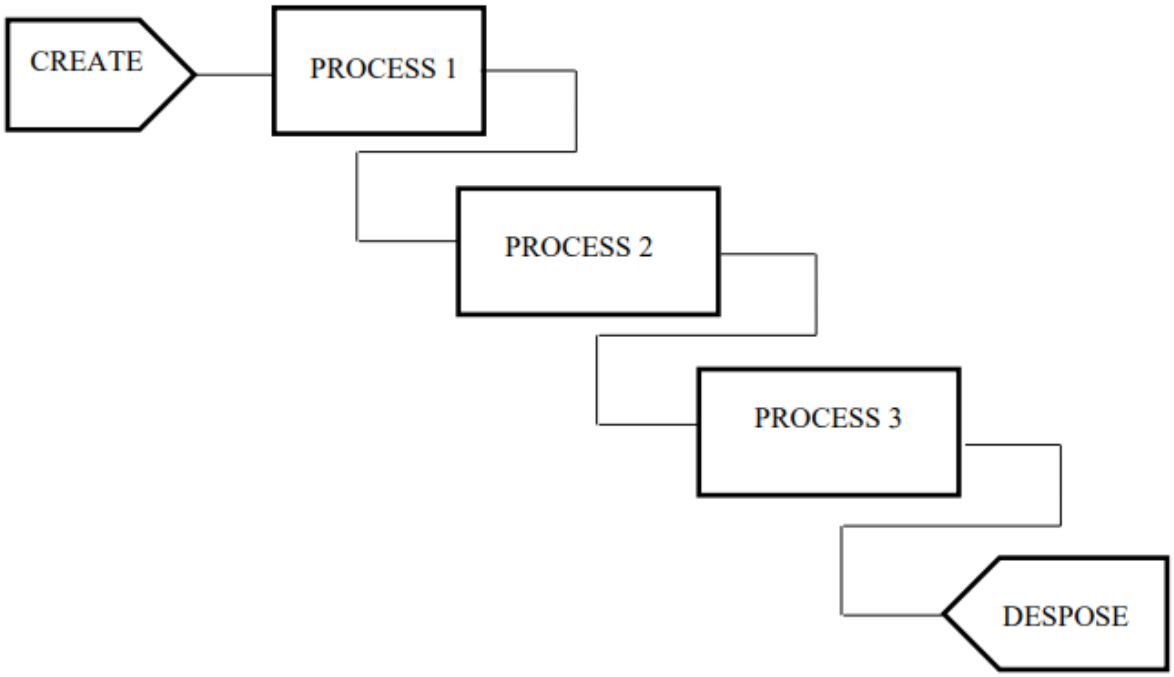

Рисунок 2.1 – Схема моделі.

5. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. **20 балів.**
6. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки. **30 балів.**

Хід роботи

Код для виконання завдання:

```
import Create from './Create';
import Distribution from './Distribution';
import Model from './Model';
import Process from './Process';

const create = new Create(2);
const p1 = new Process(1);
const p2 = new Process(1);
const p3 = new Process(1);

create.setNextElements([{ element: p1, probability: 1 }]);
p1.setNextElements([{ element: p2, probability: 1 }]);
p2.setNextElements([{ element: p3, probability: 1 }]);

p1.setMaxQueueLength(2);
p2.setMaxQueueLength(2);
p3.setMaxQueueLength(3);

create.setDistribution(Distribution.EXPONENTIAL);
p1.setDistribution(Distribution.EXPONENTIAL);
p2.setDistribution(Distribution.EXPONENTIAL);
p3.setDistribution(Distribution.EXPONENTIAL);

p1.setResourcesCount(1);
p2.setResourcesCount(1);
p3.setResourcesCount(1);

const model = new Model([create, p1, p2, p3]);

model.simulate(1000);
```

```
import CustomRandom from './CustomRandom';
import Distribution from './Distribution';

export default abstract class Element {
  protected static nextId = 0;

  public readonly id: number;
  public readonly name: string;
  private nextElements: { element: Element; probability: number }[] = [];
  private nextT: number = 0;
  private currentT: number = 0;
  private distribution: Distribution;
  public readonly delayMean: number;
  public readonly delayVariance: number;
  protected quantity: number = 0;
```

```
protected state: number = 0;

constructor(
  name = '',
  distribution = Distribution.STATIC,
  delayMean = 0,
  delayVariance = 0
) {
  this.id = Element.nextId++;
  this.name = name || 'element' + this.id;
  this.distribution = distribution;
  this.delayMean = delayMean;
  this.delayVariance = delayVariance;
}

public getDelay(): number {
  switch (this.distribution) {
    case Distribution.NORMAL:
      return CustomRandom.generateNormal(
        Math.sqrt(this.delayVariance),
        this.delayMean
      );
    case Distribution.EXPONENTIAL:
      return CustomRandom.generateExponential(1 / this.delayMean);
    case Distribution.UNIFORM:
      return CustomRandom.generateUniform() * 2 * this.delayMean;
    case Distribution.STATIC:
    default:
      return this.delayMean;
  }
}

public inAction() {}

public outAction() {
  this.quantity++;
}

public getQuantity() {
  return this.quantity;
}

public setNextElements(
  elements: { element: Element; probability: number }[]
) {
  if (elements.reduce((acc, el) => acc + el.probability, 0) !== 1)
    throw new Error('Sum of possibilities should equal 1');

  this.nextElements = elements;
}
```

```
public getNextElement() {
  const rand = Math.random();

  let sum = 0;

  for (const { element, probability } of this.nextElements) {
    sum += probability;

    if (rand < sum) return element;
  }

  return undefined;
}

public getNextT() {
  return this.nextT;
}

public setNextT(t: number) {
  this.nextT = t;
}

public getCurrentT() {
  return this.currentT;
}

public setCurrentT(t: number) {
  this.currentT = t;
}

public doStatistics(delta: number) {}

public printResult() {
  console.log(this.name + ' quantity = ' + this.quantity + '\n');
}

public printInfo() {
  console.log(
    this.name +
      ' state= ' +
      this.state +
      ' quantity = ' +
      this.quantity +
      ' tnext= ' +
      this.nextT
  );
}

public setDistribution(distribution: Distribution) {
  this.distribution = distribution;
}
```

```
}
```

```
enum Distribution {
  EXPONENTIAL = 'exponential',
  NORMAL = 'normal',
  UNIFORM = 'uniform',
  STATIC = 'static',
}

export default Distribution;
```

```
import generateRandomOne from '../../lab1/src/server/generateRandomOne';
import generateRandomThree from '../../lab1/src/server/generateRandomThree';
import generateRandomTwo from '../../lab1/src/server/generateRandomTwo';

export default class CustomRandom {
  public static generateNormal(o: number, a: number) {
    return generateRandomTwo(o, a);
  }

  public static generateExponential(lambda: number) {
    return generateRandomOne(lambda);
  }

  public static generateUniform() {
    return generateRandomThree();
  }
}
```

```
import Distribution from './Distribution';
import Element from './Element';

export default class Create extends Element {
  constructor(delay: number) {
    super('create', Distribution.STATIC, delay);
  }

  public outAction() {
    super.outAction();
    super.setNextT(super.getCurrentT() + super.getDelay());
    super.getNextElement()?.inAction();
  }
}
```

```
import Element from './Element';
import Process from './Process';

export default class Model {
  private elements: Element[] = [];
  private currentT: number = 0;
  private nextT: number = 0;
  private event: number = 0;

  constructor(elements: Element[]) {
    this.elements = elements;
  }

  public simulate(t: number) {
    while (this.currentT < t) {
      this.nextT = Infinity;
      for (let i = 0; i < this.elements.length; i++) {
        const element = this.elements[i];
        if (element.getNextT() < this.nextT) {
          this.nextT = element.getNextT();
          this.event = i;
        }
      }
      console.log(
        '\nIts time for event in ' +
          this.elements[this.event].name +
          ', time = ' +
          this.nextT
      );
      for (const element of this.elements) {
        element.doStatistics(this.nextT - this.currentT);
      }
      this.currentT = this.nextT;
      for (const element of this.elements) {
        element.setCurrentT(this.currentT);
      }
      this.elements[this.event].outAction();
      for (const element of this.elements) {
        if (element.getNextT() === this.currentT) {
          element.outAction();
        }
      }
      this.printInfo();
    }
    this.printResult();
  }

  public printInfo() {
    for (const element of this.elements) {
      element.printInfo();
    }
```

```
  }

  public printResult() {
    console.log('\n-------------RESULTS-------------');
    for (const element of this.elements) {
      element.printResult();
      if (element instanceof Process) {
        console.log(
          `mean length of queue = ${
            element.getMeanQueue() / this.currentT
          }\nmean resources busy = ${
            element.getMeanBusyResources() / this.currentT
          }\nfailure probability = ${
            element.getFails() / (element.getQuantity() + element.getFails())
          }\n`
        );
      }
    }
  }
}
```

```
import Distribution from './Distribution';
import Element from './Element';

export default class Process extends Element {
  private queue = 0;
  private maxQueueLength = Number.MAX_VALUE;
  private fails = 0;
  private meanQueue = 0;
  private meanBusyResources = 0;
  private resourcesCount = 1;

  constructor(delay: number) {
    super('process' + Element.nextId, Distribution.STATIC, delay);
  }

  public inAction() {
    super.inAction();

    if (this.state !== this.resourcesCount) {
      this.state++;
      this.setNextT(this.getCurrentT() + this.getDelay());
      return;
    }

    if (this.queue < this.maxQueueLength) {
      this.queue++;
      return;
    }
```

```
    this.fails++;
  }

  public outAction() {
    const nextElement = super.getNextElement();

    this.quantity += this.state;

    for (let i = 0; i < this.state; i++) {
      nextElement?.inAction();
    }

    this.setNextT(Infinity);
    this.state = 0;

    if (this.queue > 0) {
      while (this.state < this.resourcesCount && this.queue > 0) {
        this.queue--;
        this.state++;
      }

      this.setNextT(this.getCurrentT() + this.getDelay());
    }
  }

  public doStatistics(delta: number) {
    this.meanQueue += this.queue * delta;
    this.meanBusyResources += this.state * delta;
  }

  public getQueue() {
    return this.queue;
  }

  public getMeanQueue() {
    return this.meanQueue;
  }

  public getMeanBusyResources() {
    return this.meanBusyResources;
  }

  public setMaxQueueLength(length: number) {
    this.maxQueueLength = length;
  }

  public setResourcesCount(resourcesCount: number) {
    this.resourcesCount = resourcesCount;
  }
```

```
  public getFails() {
    return this.fails;
  }

  public printInfo() {
    super.printInfo();
    console.log('fails = ' + this.fails);
  }
}
```

Верифікація моделі:

	Вхідні змінні										Вихідні змінні									
Прогін	Середній інтервал надходження вимог	Кількість пристроїв СМО1	Обмеження черги СМО1	Середня тривалість обслуговування СМО1	Кількість пристроїв СМО2	Обмеження черги СМО2	Середня тривалість обслуговування СМО2	Кількість пристроїв СМО3	Обмеження черги СМО3	Середня тривалість обслуговування СМО3	Кількість вимог, що надійшли у мережу	Кількість не обслугованих у СМО1	Кількість не обслугованих у СМО2	Кількість не обслугованих у СМО3	Ймовірність відмови	Середня довжина черги СМО1	Середня кількість зайнятих пристроїв СМО1	Середня довжина черги СМО2	Середня кількість зайнятих пристроїв СМО2	Середня довжина черги СМО3	Середня кількість зайнятих пристроїв СМО3
1	0,2	5	10	1,2	7	8	2	2	1	1	5058	1494	909	1821	0,8351126928	5,44	4,54	2,86	5,08	0,29	0,8
2	**0,5**	5	10	1,2	7	8	2	2	**3**	**0,5**	2007	14	92	600	0,3517688092	0,71	2,54	1,02	3,44	0,56	0,7
3	**0,2**	5	10	1,2	7	8	2	**4**	3	0,5	5060	1357	1119	422	0,5727272727	5,28	4,48	3,18	5,23	0,43	1,07
4	1	2	5	1	3	8	2	2	2	3	982	21	21	427	0,4775967413	0,44	0,95	1,35	1,89	0,91	1,54
5	1	2	**0**	1	3	8	2	2	2	**2**	1032	289	10	180	0,4641472868	0	0,77	0,95	1,68	0,5	1,1
6	1	2	0	1	3	**3**	**0,5**	2	2	2	954	241	0	112	0,3700209644	0	0,72	0,002	0,36	0,43	1,21
7	1	2	0	**3**	**2**	3	0,5	2	2	2	1014	582	2	37	0,6124260355	0	1,31	0,01	0,22	0,22	0,85
8	1	**1**	**2**	3	2	8	2	2	2	2	987	664	1	9	0,6828774063	1,58	0,97	0,003	0,15	0,09	0,57

Результати роботи коду:

```
-------------RESULTS-------------
create quantity = 509

process1 quantity = 466

mean length of queue = 0.2998565348762671
mean resources busy = 0.4680995692041779
failure probability = 0.08447937131630648

process2 quantity = 441

mean length of queue = 0.24760466610471876
mean resources busy = 0.4585796951509255
failure probability = 0.0536480686695279

process3 quantity = 434

mean length of queue = 0.29379062060844235
mean resources busy = 0.44698181398748194
failure probability = 0.013636363636363636

PS F:\Projects\system modelling labs>
```

Висновки: під час виконання цього завдання було побудовано дискретно-імітаційну модель та проведено її дослідження та верифікацію.