Министерство образования Республики Беларусь

Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6.
"Наследование и виртуальные функции"

Выполнил:
Ст. 2 курса гр. АС-53
Бранчук Д. В.
Проверила:
Давидюк Ю. И.

Брест, 2020

1. Цель. Получить практические навыки создания иерархии классов и использования статических компонентов класса.

2. Постановка задачи (Вариант 3)

Написать программу, в которой создается иерархия классов. Включить полиморфные объекты в связанный список, используя статические компоненты класса. Показать использование виртуальных функций.

рабочий, кадры, инженер, администрация;

Классы: Cadre, Administration, Worker, Engineer
Конструкторы:

- Пустой
- С параметрами
- Копирования

Деструктор.
Виртуальные фукнции:

- Добавления в список
- Вывода информации

3. Иерархия классов в виде графа:

- Кадры
  - Администрация
  - Рабочие
    - Инженер

4. Определение пользовательских классов с комментариями.

```
//Базовый класс
class Cadre {
protected:
	std::string	FirstName;
	std::string	SecondName;
	std::string	LastName;
	float		DateOfBirth;
public:
	Cadre();
	Cadre(
		std::string,
		std::string,
		std::string,
		float
	);
	Cadre(const Cadre&);
	~Cadre();
	virtual void Show() = 0;
	virtual void Add() = 0;

};
```

```
Cadre::Cadre() {
	FirstName = "UndefinedFirstName";
	SecondName = "UndefinedSecondName";
	LastName = "UndefinedLastName";
	DateOfBirth = 1;
}
Cadre::Cadre(std::string _FirstName, std::string _SecondName, std::string _LastName, float
_DateOfBirth) {
	FirstName = _FirstName;
	SecondName = _SecondName;
	LastName = _LastName;
	DateOfBirth = _DateOfBirth;
}
Cadre::Cadre(const Cadre& rCadre) { }
Cadre::~Cadre() { }
//Наследственный класс Администрация
class Administration : public Cadre {
protected:
	std::string Post;
public:
	Administration();
	Administration(
		std::string,
		std::string,
		std::string,
		float,
		std::string
	);
	Administration(const Administration&);
	~Administration();
	void Show();
	void Add();
};

Administration::Administration() {
	Post = "UndefinedPost";
}
Administration::Administration(
	std::string   _FirstName,
	std::string   _SecondName,
	std::string   _LastName,
	float         _DateOfBirth,
	std::string   _Post
) : Cadre(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth
) {
	Post = _Post;
}
Administration::Administration(const Administration& rAdministration) { }
Administration::~Administration() { }

void Administration::Show() {
	std::cout << "Administration:\nFirstName: " << Cadre::FirstName << "\nSecondName: " <<
Cadre::SecondName << "\nLastName: " << Cadre::LastName << "\nDate of birth: " <<
Cadre::DateOfBirth << "\nPost: " << this->Post << std::endl << std::endl;
}
```

```
void Administration::Add() {
	list* pTemp = new list;
	pTemp->pData = new Administration(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Post
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}
// Наследственный класс Работник
class Administration : public Cadre {
protected:
	std::string Post;
public:
	Administration();
	Administration(
		std::string,
		std::string,
		std::string,
		float,
		std::string
	);
	Administration(const Administration&);
	~Administration();
	void Show();
	void Add();
};

Administration::Administration() {
	Post = "UndefinedPost";
}
Administration::Administration(
	std::string   _FirstName,
	std::string   _SecondName,
	std::string   _LastName,
	float         _DateOfBirth,
	std::string   _Post
) : Cadre(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth
) {
	Post = _Post;
}
Administration::Administration(const Administration& rAdministration) { }
Administration::~Administration() { }

void Administration::Show() {
```

```
	std::cout << "Administration:\nFirstName: " << Cadre::FirstName << "\nSecondName: " <<
Cadre::SecondName << "\nLastName: " << Cadre::LastName << "\nDate of birth: " <<
Cadre::DateOfBirth << "\nPost: " << this->Post << std::endl << std::endl;
}

void Administration::Add() {
	list* pTemp = new list;
	pTemp->pData = new Administration(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Post
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}
// Класс Рабочий
class Worker : public Cadre {
protected:
	std::string Experience;
public:
	Worker();
	Worker(
		std::string,
		std::string,
		std::string,
		float,
		std::string
	);
	Worker(const Worker&);
	~Worker();
	void Show();
	void Add();
};

Worker::Worker() { Experience = "UndefinedExperience"; }
Worker::Worker(
	std::string  _FirstName,
	std::string  _SecondName,
	std::string  _LastName,
	float        _DateOfBirth,
	std::string  _Experience
) : Cadre(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth
) {
	Experience = _Experience;
}
Worker::Worker(const Worker& rWorker) { }
Worker::~Worker() { }
```

```
void Worker::Show() {
	std::cout << "Worker:\nFirstName: " << Cadre::FirstName << "\nSecondName: " <<
Cadre::SecondName << "\nLastName: " << Cadre::LastName << "\nDateOfBirth: " <<
Cadre::DateOfBirth << "\nExperience: " << this->Experience << std::endl << std::endl;
}

void Worker::Add() {
	list* pTemp = new list;
	pTemp->pData = new Worker(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Experience
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}
FirstName = "UndefinedFirstName";
SecondName = "UndefinedSecondName";
LastName = "UndefinedLastName";
DateOfBirth = 1;
}
Cadre::Cadre(std::string _FirstName, std::string _SecondName, std::string _LastName, float
_DateOfBirth) {
	FirstName = _FirstName;
	SecondName = _SecondName;
	LastName = _LastName;
	DateOfBirth = _DateOfBirth;
}
Cadre::Cadre(const Cadre& rCadre) { }
Cadre::~Cadre() { }
// Класс Инженер
class Engineer : public Worker {
protected:
	std::string Education;
public:
	Engineer();
	Engineer(
		std::string,
		std::string,
		std::string,
		float,
		std::string,
		std::string
	);
	Engineer(const Engineer&);
	~Engineer();
	void Show();
	void Add();
};
```

```
Engineer::Engineer() { Education = "UndefinedEducation"; }
Engineer::Engineer(
      std::string     _FirstName,
      std::string     _SecondName,
      std::string     _LastName,
      float      _DateOfBirth,
      std::string     _Experience,
      std::string     _Education
) : Worker(
      _FirstName,
      _SecondName,
      _LastName,
      _DateOfBirth,
      _Experience
) {
      Education = _Education;
}
Engineer::Engineer(const Engineer& rEngineer) { }
Engineer::~Engineer() { }

void Engineer::Show() {
      std::cout << "Worker:\nFirstName: " << Cadre::FirstName <<
"\nSecondName: " << Cadre::SecondName << "\nLastName: " << Cadre::LastName
<< "\nDateOfBirth: " << Cadre::DateOfBirth << "\nExperience: " <<
Worker::Experience << "\nEducation: " << this->Education << std::endl <<
std::endl;
}

void Engineer::Add() {
      list* pTemp = new list;
      pTemp->pData = new Engineer(
            FirstName,
            SecondName,
            LastName,
            DateOfBirth,
            Experience,
            Education
      );
      pTemp->pNext = nullptr;
      if (CadreList::asList.GetHead() != nullptr) {
            CadreList::asList.GetTail()->pNext = pTemp;
            CadreList::asList.SetTail(pTemp);
      }
      else {
            CadreList::asList.SetHead(pTemp);
            CadreList::asList.SetTail(pTemp);
      }
}
```

5. Реализация конструкторов с параметрами и деструктора.

- Для класса Cadre:

```
Cadre::Cadre() {
      FirstName = "UndefinedFirstName";
```

```
	SecondName = "UndefinedSecondName";
	LastName = "UndefinedLastName";
	DateOfBirth = 1;
}
Cadre::Cadre(std::string _FirstName, std::string _SecondName, std::string _LastName, float
_DateOfBirth) {
	FirstName = _FirstName;
	SecondName = _SecondName;
	LastName = _LastName;
	DateOfBirth = _DateOfBirth;
}
Cadre::Cadre(const Cadre& rCadre) { }
Cadre::~Cadre() { }
```

Для класса Administration:

```
	 Administration::Administration() {
	Post = "UndefinedPost";
}
Administration::Administration(
	std::string	_FirstName,
	std::string	_SecondName,
	std::string	_LastName,
	float		_DateOfBirth,
	std::string	_Post
) : Cadre(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth
) {
	Post = _Post;
}
Administration::Administration(const Administration& rAdministration) { }
Administration::~Administration() { }
```

- Для класса Worker:

```
  Worker::Worker() { Experience = "UndefinedExperience"; }
Worker::Worker(
	std::string	_FirstName,
	std::string	_SecondName,
	std::string	_LastName,
	float		_DateOfBirth,
	std::string	_Experience
) : Cadre(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth
) {
	Experience = _Experience;
}
Worker::Worker(const Worker& rWorker) { }
```

- Для класса Engineer:

```
 Engineer::Engineer() { Education = "UndefinedEducation"; }
Engineer::Engineer(
	std::string	_FirstName,
	std::string	_SecondName,
	std::string	_LastName,
```

```
	float		_DateOfBirth,
	std::string	_Experience,
	std::string	_Education
) : Worker(
	_FirstName,
	_SecondName,
	_LastName,
	_DateOfBirth,
	_Experience
) {
	Education = _Education;
}
Engineer::Engineer(const Engineer& rEngineer) { }
Engineer::~Engineer() { }
```

6. Реализация методов для добавления объектов в список.

```
void Engineer::Add() {
	list* pTemp = new list;
	pTemp->pData = new Engineer(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Experience,
		Education
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}

void Worker::Add() {
	list* pTemp = new list;
	pTemp->pData = new Worker(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Experience
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}

void Administration::Add() {
```

```
	list* pTemp = new list;
	pTemp->pData = new Administration(
		FirstName,
		SecondName,
		LastName,
		DateOfBirth,
		Post
	);
	pTemp->pNext = nullptr;
	if (CadreList::asList.GetHead() != nullptr) {
		CadreList::asList.GetTail()->pNext = pTemp;
		CadreList::asList.SetTail(pTemp);
	}
	else {
		CadreList::asList.SetHead(pTemp);
		CadreList::asList.SetTail(pTemp);
	}
}
```

7. Реализация метода для просмотра списка.

```
void Engineer::Show() {
	std::cout << "Worker:\nFirstName: " << Cadre::FirstName <<
"\nSecondName: " << Cadre::SecondName << "\nLastName: " << Cadre::LastName
<< "\nDateOfBirth: " << Cadre::DateOfBirth << "\nExperience: " <<
Worker::Experience << "\nEducation: " << this->Education << std::endl <<
std::endl;
}

void Worker::Show() {
	std::cout << "Worker:\nFirstName: " << Cadre::FirstName << "\nSecondName: " <<
Cadre::SecondName << "\nLastName: " << Cadre::LastName << "\nDateOfBirth: " <<
Cadre::DateOfBirth << "\nExperience: " << this->Experience << std::endl << std::endl;
}

void Administration::Show() {
	std::cout << "Administration:\nFirstName: " << Cadre::FirstName << "\nSecondName: " <<
Cadre::SecondName << "\nLastName: " << Cadre::LastName << "\nDate of birth: " <<
Cadre::DateOfBirth << "\nPost: " << this->Post << std::endl << std::endl;
}
```

8. Листинг демонстрационной программы.

```
void classes() {
	std::cout << "List:" << std::endl << std::endl;
	Administration cadre0("Dimka", "Viktorovich", "Branchuk", 1999, "Director");
	cadre0.Add();
	Worker cadre1("Lizka", "Sergeevna", "Ktototavna", 2001, "19 year");
	cadre1.Add();
	Worker cadre11("Dimas", "Andreeevich", "Koren", 1993, "25 year");
	cadre11.Add();
	Worker cadre12("Denis", "Alekseevich", "Boss", 1995, "22 year");
	cadre12.Add();
	Engineer cadre2("Mishka", "Nikitich", "Samss", 1992, "33 year", "Higher technical
education");
	cadre2.Add();
	Engineer cadre21("Dashka", "Vladimirovna", "Lololo", 1991, "6 year", "Higher technical
education");
	cadre21.Add();
```

```
	Administration cadre3;
	cadre3.Add();
	Worker cadre31;
	cadre31.Add();
	Engineer cadre32;
	cadre32.Add();
	CadreList::View();
}

int main() {
	classes();
	return 0;
}
```

Вывод программы:

```
Administration:
FirstName: Dimka
SecondName : Viktorovich
LastName : Branchuk
Date of birth : 1999
Post : Director

Worker :
FirstName: Lizka
SecondName : Sergeevna
LastName : Ktototavna
DateOfBirth : 2001
Experience : 19 year

Worker :
FirstName: Dimas
SecondName : Andreeevich
LastName : Koren
DateOfBirth : 1993
Experience : 15 year

Worker :
FirstName: Denis
SecondName : Alekseevich
LastName : Boss
DateOfBirth : 1995
Experience : 22 year

Worker :
FirstName: Mishka
SecondName : Nikitich
LastName : Samss
DateOfBirth : 1992
Experience : 33 year
Education : Higher technical education

Worker :
FirstName: Dashka
SecondName : Vladimirovna
LastName : Lololo
DateOfBirth : 1991
Experience : 6 year
Education : Higher technical education

Administration :
```

```
FirstName: UndefinedFirstName
SecondName : UndefinedSecondName
LastName : UndefinedLastName
Date of birth : 1
Post : UndefinedPost

Worker :
FirstName: UndefinedFirstName
SecondName : UndefinedSecondName
LastName : UndefinedLastName
DateOfBirth : 1
Experience : UndefinedExperience

Worker :
FirstName: UndefinedFirstName
SecondName : UndefinedSecondName
LastName : UndefinedLastName
DateOfBirth : 1
Experience : UndefinedExperience
Education : UndefinedEducation
```

9. Объяснение необходимости виртуальных функций. Следует показать, какие результаты будут в случае виртуальных и не виртуальных функций.

При наследовании бывает необходимо, чтобы поведение некоторых методов базового класса и классов-наследников различалось, именно для этого и требуется наличие виртуальных функций `virtual void Show() = 0;virtual void Add() = 0;`

В данном коде, в случае отсутствия виртуальной функции нельзя будет переопределить поведение.

10. Вывод:

Получил практические навыки реализации классов на C++.