Билет №13

Задание 1

Познакомьтесь с предложенной реализацией классов. Определите тип отношения между классами (предоставьте описание в виде UML), ответ обоснуйте.

Объявление класса A

```
#include "ClassB.h"
class ClassB;
class ClassA
{
	int m_n; //фактическое число частей
	ClassB* m_pB[100]; //массив указателей на части
public:
	ClassA();
	~ClassA();
	const int getN() const; //вернуть число частей
	const ClassB** getB(int&) const; //вернуть части
	bool add(const int); //добавить часть
	bool del(const int); //удалить часть
	int find(const int) const; //вернуть индекс части

	const int getX(const int) const;//вернуть значение объекта
};
```

Реализация класса A

```


#include "ClassA.h"
ClassA::ClassA() : m_n(0) {}
ClassA :: ~ClassA()
{
	for (int j(0); j < m_n; j++)
		delete m_pB[j];
}
//вернуть количество частей
const int ClassA::getN() const { return m_n; }
//вернуть части const
const ClassB** ClassA::getB(int& n) const
{
	n = m_n;
	return const_cast <const ClassB**> (m_pB);
}
//добавить часть
//метод получает все необходимые параметры для создания части
bool ClassA::add(const int x)
{
	if (find(x) >= 0) return false;
	m_pB[m_n] = new ClassB;
	m_pB[m_n]->setX(x);
	m_n++; return true;
}
//удалить часть
bool ClassA::del(int key)
{
	int JDel = find(key);
	//поиск индекса части
	if (JDel < 0)
		return false;
	//часть не найдена
```

```
	delete m_pB[JDel];
	//разрушение части
	while (JDel < m_n - 1)
	{
		//сжатие массива
		m_pB[JDel] = m_pB[JDel + 1];
		JDel++;
	}
	m_n--;
	return true;
}
//вернуть индекс части
int ClassA::find(const int key) const
{
	for (int j(0); j < m_n; j++)
		if (m_pB[j]->verify(key))
			return j;
	return -1;
}
//вернуть значение объекта с индексом
// вызывается метод класса частей m_pB[j]->getX();
const int ClassA::getX(const int j) const
{
	return m_pB[j]->getX();
}
```

Объявление класса B

```
class ClassB
{
	int m_x; //член-данное части

public:
	ClassB();
	~ClassB();
	//методы установки и получения значения m_x
	void setX(const int);
	const int getX() const;
	bool verify(const int) const; //проверить объект
	ClassB& operator = (const ClassB&); //перегрузка оператора =
	//перегрузка операторов == и !=
	bool operator == (const ClassB&) const;
	bool operator != (const ClassB&) const;
};
```

Реализация класса B

```
#include "ClassB.h"
ClassB::ClassB() : m_x(0) {}
ClassB :: ~ClassB() {}
//установить m_x
void ClassB::setX(const int x) { m_x = x; }
//получить значение т_х
const int ClassB::getX() const { return m_x; }
bool ClassB::verify(const int x) const
{
	return m_x == x;
}
```

```
//перегрузка оператора =
ClassB& ClassB :: operator = (const ClassB& r)
{
	if (this != &r)
		m_x = r.m_x;
	return *this;
}
//перегрузка оператора ==
bool ClassB :: operator == (const ClassB& r) const
{
	return m_x == r.m_x;
}
//перегрузка оператор !=
bool ClassB :: operator != (const ClassB& r) const
{
	return m_x != r.m_x;
}
```

Задание 2

Базовые принципы проектирования (SOLID). Принцип открытости/закрытости. На представленном примере определить, существует ли нарушение принципа открытости/закрытости. Если да, **то предложите решение**. Ответ обоснуйте.

Легенда

Банк предлагает различные типы сберегательных счетов (***накопительные***, ***регулярные сбережения*** и т. Д.), которые удовлетворяют потребности многих клиентов. У банка разные наборы правил, и каждый ***тип сберегательного счета*** имеет свой набор правил для расчета процентов. Для расчета ***процентов*** по счету разработан следующий класс с методом (**MyCalculateInterest**) расчета процентов.

```
enum AccountType { Regular=1, Salary};
class MySavingAccount
{
	int myInterest;// Процентная ставка по счету
	int myBalance; // Текущий баланс счета
	int myAmount;  // Сумма
	//Описание нужных конструкторов, методы и свойства класса
public:

	int MyCalculateInterest(AccountType accountType)
	{
		if(accountType == Regular)
		{
	// Рассчитываем проценты для регулярного сберегательного счета на основе правил и
	// регулирование банка
			myInterest = myBalance * 0.4;
			if(myBalance < 1000) myInterest -= myBalance * 0.2;
			if(myBalance < 50000) myInterest += myAmount * 0.4;
		}
		else if (accountType == Salary)
		{
		// Расчет процентов на сберегательный счет в соответствии с правилами и
		//положениями банка
			myInterest = myBalance * 0.5;
		}
	}
```

```
}
```

Задание 3

Паттерн стратегия. Его назначение, архитектура. Рассмотрите предложенную легенду, примените к ней паттерн стратегия. Решение представить в виде кода с подробными объяснениями.

Легенда

Рассмотрим ***индикатор выполнения*** – это окно, которое приложение может использовать для индикации хода длительности операции (например, процесса установки). Обычно это прямоугольное окно, которое постепенно заполняется слева направо цветом выделения по мере выполнения операции. У него есть ***диапазон*** и ***текущая позиция***. Диапазон представляет собой всю продолжительность операции, а текущая позиция представляет прогресс, достигнутый приложением в завершении операции. Диапазон и текущая позиция используются для определения ***процента*** индикатора выполнения, который нужно заполнить ***цветом выделения***. Существуют различные ***направления заполнения***, такие как справа налево, сверху вниз и снизу вверх, также с заданным направлением заливки можно использовать различные ***типы заливок***, такие как непрерывная заливка, прерывистая заливка или заливка на основе узора.

Задание.

Для пользовательского приложения реализовать возможность настройки индикатора выполнения с конкретным классом-заполнителем.

Требования к ответам

Задание представляете в файле типа word. Каждое задание снабжаете подробными комментариями (обоснованиями предложенного решения). UML диаграмму отдельной картинкой (можно от руки нарисовать), также с объяснениями + ссылка на код в онлайн компиляторе. https://www.onlinegdb.com/

Ответ, скопированный откуда-либо в качестве ответа на задание, не рассматривается, такая работа не зачитывается.