Лабораторная работа № 3

Тема: Перегрузка методов

Цель работы. Изучить принципы использования перегрузки методов.

При перегрузке методов создается несколько методов с одинаковыми именами, но разными сигнатурами. Сигнатура метода состоит из типа возвращаемого методом результата, имени метода и списка аргументов, с учетом их типов. Так как при перегрузке методов используется общее имя, то разные варианты метода могут отличаться типом возвращаемого результата и (или) списком аргументов. Технически речь идет о разных методах, но поскольку все они имеют одинаковые названия, обычно говорят об одном методе. Относительно функциональности различных вариантов перегруженного метода, то при отсутствии формальных ограничений общепринятым является реализация одного общего алгоритма. Например, несколько одинаково именуемых методов предназначены для отрисовки визуальных оконных элементов, а различие в списке аргументов этих методов позволяет подобрать наиболее подходящий из них — возможность задавать геометрические размеры элемента, цвет, положение, делать привязку к родительскому объекту. Поэтому считается плохой ситуация, когда одна версия метода решает одну задачу, а другая — принципиально другую, хотя с технической точки зрения это возможно.

Конструктор — это метод, который используется при создании объекта. Имя конструктора совпадает с именем класса, он не возвращает результат, поэтому идентификатор типа для конструктора не указывается, а также конструктор может иметь аргументы и его можно перегружать. Если конструктор класса явно не описан, применяется конструктор по умолчанию. Конструктор по умолчанию не имеет аргументов и создается компилятором автоматически на основе имеющихся в классе полей. Если в классе имеется описание хотя бы одного конструктора (с аргументами или без них) конструктор по умолчанию не создается и становится недоступным. Если для создания объектов понадобятся конструкторы как с аргументами, так и без них, то конструктор без аргументов придется описывать явно. Перегрузка конструкторов означает, что в классе может быть одновременно несколько конструкторов, которые различаются количеством аргументов в них. С практической точки зрения, это позволяет при создании объектов использовать наиболее подходящий конструктор.

Объект может быть аргументом и результатом метода. При описании передачи объекта методу в качестве аргумента в списке его аргументов указывается имя соответствующего класса. Также метод может иметь объектный тип, тогда в качестве результата он возвращает объект. В действительности, и при передаче и при возвращении эти операции выполняются не с самими объектами, а с соответствующими ссылками на них.

В Java существует два способа передачи аргументов методам: по значению и по ссылке. При передаче аргумента по значению в метод передается копия этого аргумента и все операции в методе выполняются не с самим аргументом, а с его копией. Если аргумент передается по ссылке, все операции в теле метода выполняются непосредственно с аргументом. На практике разница между способами передачи аргументов проявляется в том случае, если в методе предпринимается попытка их изменить. В Java переменные базовых (простых) типов передаются по значению, а объекты — по ссылке, что объясняется способом создания объектов и реальную связь объекта и объектной переменной. При объявлении объектной переменной (переменная типа «класс») объект не создается. Для этой переменной выделяется место в памяти, и в эту область памяти может быть записана ссылка на объект. Для создания объекта используется оператор new с вызовом соответствующего конструктора. В этом случае в

памяти выделяется место под объект и если конструктором предусмотрено, то заполняются поля этого объекта и выполняются определенные действия. В результате действия оператора new объектной переменной присваивается ссылка на созданный объект.

Когда объект передается методу в качестве аргумента, то указывается имя объекта, то есть, по-сути, ссылка на этот объект. Сама ссылка передается по значению, а значит внутри метода объектная переменная ссылается на тот же объект, что и оригинальный аргумент с именем объекта. Поэтому какие-либо операции в методе над объектной переменной затрагивают сам объект, на который она ссылается. Изменения, применяемые к объекту внутри метода, будут изменять сам оригинал объекта. В этом смысле объектная переменная в Java напоминает указатель на объект в C++, но без использования реальных адресов.

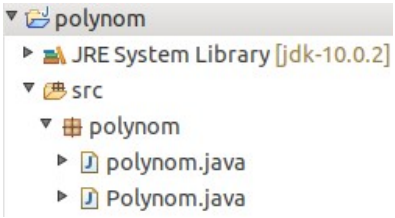
Задания к работе

Задание 1

В научных и инженерных задачах широко используются выражения полиномиального типа:

$$P(x)=\sum_{k=0}^{n} a_k x^k$$

Ниже представлена программа, в которой для работы с выражениями полиномиального типа создается класс Polynom. В классе описаны методы сложения, вычитания, умножения полиномов, умножения и деления полинома на число и вычисления производной для полинома.




```
package polynom;

public class polynom {

		public static void main(String[] args){
			// Коэффициенты для полинома:
		double[] coefs=new double[]{3,-2,-1,0,1};
			// Создание полинома:
		Polynom P=new Polynom(coefs);
		System.out.println("\tКоэффициенты исходного полинома:");
			// Коэффициенты полинома:
		P.show();
		System.out.println("\tЗначение полинома в точке:");
			// Значение полинома для единичного аргумента:
		P.show(1);
		System.out.println("\tВторая производная:");
			// Вычисление второй производной для полинома:
```

```
                Polynom Q=P.diff(2);
                        // Результат вычисления производной (коэффициенты):
                Q.show();
                System.out.println("\tСумма полиномов:");
                        // Сумма полиномов (результат):
                Q.plus(P).show();
                System.out.println("\tПроизведение полиномов:");
                        // Произведение полиномов (результат):
                Q.prod(P).show();
                }
        }


package polynom;

public class Polynom {
                // Полином степени n-1:
        private int n;
                // Коэффициенты полинома:
        private double[] a;
                // Определение коэффициентов полинома на основе массива:
        void set(double[] a){
                this.n=a.length;
                this.a=new double[n];
                int i;
                for(i=0;i<n;i++)
                        this.a[i]=a[i];
        }
        // Определение коэффициентов полинома
        // (аргументы - размер массива и число для заполнения):
        void set(int n, double z){
                this.n=n;
                this.a=new double[n];
                int i;
                for(i=0;i<n;i++)
                        this.a[i]=z;
        }

        // Определение коэффициентов полинома
        // (аргумент - размер массива, массив заполняется нулями):
        void set(int n){
                set(n, 0);}
                // Вычисление значения полинома в точке:
                double value(double x){
                double z=0,q=1;
                for(int i=0;i<n;i++){
                        z+=a[i]*q;
                        q*=x;}
                return z;
        }
        // Отображение коэффициентов полинома:
        void show(){int i;
                System.out.print("Степень x:\t");
                for(i=0;i<n-1;i++){ System.out.print(" "+i+"\t");}
                System.out.println(" "+(n-1));
                System.out.print("Коэффициент:\t");
                for(i=0;i<n-1;i++){ System.out.print(a[i]+"\t");}
                System.out.println(a[n-1]);
        }
                // Отображение значения полинома в точке:
        void show(double x){
                System.out.println("Значение аргумента x="+x);
```

```
        System.out.println("Значение полинома P(x)="+value(x));
}
        // Производная от полинома:
Polynom diff(){
        Polynom t=new Polynom(n-1);
        for(int i=0;i<n-1;i++)
                t.a[i]=a[i+1]*(i+1);
        return t;
}
        // Производная от полинома порядка k:
Polynom diff(int k){
        if(k>=n) return new Polynom(1);
        if(k>0) return diff().diff(k-1);
        else return new Polynom(a);
}
        // Сумма полиномов:
Polynom plus(Polynom Q){
        Polynom t;
        int i;
        if(n>=Q.n){
                t=new Polynom(a);
                for(i=0;i<Q.n;i++)
                        t.a[i]+=Q.a[i];
        }
        else{
                t=new Polynom(Q.a);
                for(i=0;i<n;i++)
                        t.a[i]+=a[i];
        }
        return t;
}
        // Разность полиномов:
Polynom minus(Polynom Q){
        return plus(Q.prod(-1));
}
Polynom div(double z){
        return prod(1/z);
}
        // Произведение полинома на число:
Polynom prod(double z){
        Polynom t=new Polynom(a);
        for(int i=0; i<n; i++)
                a[i]*=z;
        return t;
}
        // Произведение полинома на полином:
Polynom prod(Polynom Q){
        int N=n+Q.n-1;
        Polynom t=new Polynom(N);
        for(int i=0;i<n;i++){
                for(int j=0;j<Q.n;j++){
                        t.a[i+j]+=a[i]*Q.a[j];
                }
        }
        return t;
}
        // Конструкторы класса:
Polynom(double[] a){
        set(a);
}
Polynom(int n,double z){
        set(n,z);
```

```
}
Polynom(int n){
        set(n);}
}
```

В классе Polynom закрытое целочисленное поле n определяет степень полинома (степень полинома равняется n-1 , и такой полином определяется набором из n коэффициентов), а закрытое поле-массив a (ссылка на массив типа double) предназначено для записи коэффициентов полинома. Для заполнения поля a предназначен перегружаемый метод set() . В методе предусмотрена возможность передавать для заполнения коэффициентов полинома ссылки на массив, заполнять массив одинаковыми числами, передав аргументом методу размер массива и число для заполнения, а также описан частный случай, когда аргументом методу передается только размер массива (при этом массив заполняется нулями). Реализация последней версии метода set() базируется на вызове варианта этого же метода с первым аргументом — размером массива, и нулевым вторым аргументом.

Метод value() имеет аргумент типа double и в качестве результата возвращает значение полинома в точке, то есть для переменной, переданной аргументом метода. Перегружаемый метод show() имеет две версии: без аргументов и с одним аргументом типа double . В первом случае на экран вместе с дополнительной информацией выводятся значения коэффициентов полинома. Если методу show() передается аргумент (значение типа double), то выводится сообщение о значении полинома в этой точке.

В классе Polynom предусмотрена возможность вычислять производные произвольного порядка. Результатом вычисления производной от полинома также является полином. Задача по вычислению производной от полинома сводится к расчету на основе коэффициентов исходного полинома коэффициентов полинома-производной. Для степенной функции $y=x^n$ производная $\frac{dy}{dx}$ определяется как $\frac{dy}{dx}=nx^{n-1}$, а так как производная суммы функций есть сумма их производных, то производной для полинома $P(x)=\sum_{k=0}^{n} a_k x^k$ будет функция-полином:

$$P'(x)=\sum_{k=1}^{n} k\,a_k x^{k-1}=\sum_{k=0}^{n-1}(k+1)\,a_{k+1}x^k\equiv\sum_{k=0}^{n-1} b_k x^k$$

Коэффициенты полинома-производной b_k связаны с коэффициентами a_k исходного полинома соотношением b_k = (k + 1) a_{k+1} для k = 0,1, ..., n − 1. Производная является полиномом степени, на единицу меньшей, чем исходный полином, так как производная константы равна 0, а первое слагаемое в исходном полиноме - константа. В методе diff() это учитывается, а в качестве результата этот метод возвращает объект класса Polynom. Это объект является первой производной для полинома, реализованного через объект, из которого вызывается метод. В теле метода командой *Polynom t=new Polynom(n-1)* создается объект *t*, который является полиномом степени, на единицу меньшей, чем исходный полином. При создании объекта использован конструктор с одним аргументом - размер массива, поэтому коэффициенты поля *a* этого объекта заполняются нулями. В цикле с индексной переменной *i* командой *t.a*[*i*]=*a*[*i+1*]*(*i+1*) выполняется последовательное заполнение коэффициентов полинома-производной. После этого объект *t* возвращается как результат метода. Метод diff() является перегруженным. Целочисленный аргумент, переданный этому методу, позволяет в качестве результата получить производную соответствующего порядка. Если значение аргумента превышает степень полинома, то возвращается полином нулевой степени, т.е. с единственным

нулевым коэффициентом. При положительном (больше нуля) значении порядка производной в качестве результата возвращается объект diff().diff(k-1). Иначе возвращается объект, который является копией исходного полинома, так как производная нулевого порядка по определению совпадает с исходной функцией. Так как производная порядка k является производной порядка $k-1$ от первой производной, поэтому в методе diff() реализован ее рекурсивный вызов с уменьшением порядка.

Методом plus() в качестве результата вычисляется объект класса Polynom, соответствующий сумме двух полиномов — один реализован через объект вызова, а объект для второго полинома передается методу в качестве аргумента. Общий принцип вычисления суммы полиномов состоит в том, что нужно сложить коэффициенты, соответствующие одинаковым показателям степени переменной полинома. На начальном этапе проверяется, какой из объектов имеет больший размер поля-массива. На основе этого объекта создается его локальная копия. Затем перебираются элементы второго объекта с меньшим полем-массивом и эти элементы прибавляются к соответствующим элементам локального объекта. После этого локальный объект возвращается в качестве результата метода.

Разность полиномов вычисляется методом minus() . С помощью метода prod() полином, соответствующий объекту-аргументу метода minus() , умножается на –1, после чего с помощью метода plus() полученный в результате полином прибавляется к исходному, реализованному через объект вызова метода minus(). Процедура по вычислению объекта-результата разности двух полиномов реализована в виде plus(Q.prod(-1)), где Q — объект класса Polynom, переданный как аргумент методу minus().

Метод prod() позволяет умножать полином на число и на другой полином. Число передается как аргумент типа double методу prod(). На это число умножается каждый коэффициент полинома. Если же аргументом методу prod() передать объект класса Polynom, то вычисляется произведение полиномов (один реализован через объект вызова, второй — через объект-аргумент метода). Результатом также является объект класса Polynom . При вычислении его параметров принято во внимание, что результатом умножения полиномов степени n и m будет полином степени n + m. Локальный объект создается как Polynom *t=new Polynom*(*N*), где *N=n+Q.n*-1. Начальные значения элементов поля-массива локального объекта t равны нулю. Далее запускается двойной цикл, индексная переменная в одном цикле перебирает элементы первого поля-массива первого объекта, а вторая — второго. Изменение значений поля-массива локального объекта выполняется командой *t.a*[*i+j*]+=*a*[*i*]**Q.a*[*j*] .

Если

$$P(x)=\sum_{i=0}^{n} a_i x^i \qquad \text{и} \qquad Q(x)=\sum_{j=0}^{m} b_j x^j$$

то

$$R(x)\equiv P(x)Q(x)=\sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j = \sum_{\substack{0\leqslant i\leqslant n,\\ 0\leqslant j\leqslant m}} a_i b^j x^{i+j}$$

Другими словами, чтобы рассчитать коэффициент для полинома-результата с индексом k, необходимо найти сумму $\sum_{i+j=k} a_i b_j$ произведений коэффициентов исходных полиномов, сумма индексов которых равняется k.

Конструкторы класса Polynom реализованы на основе различных версий метода set().

В методе main() класса PolynomDemo проверяется работа некоторых методов, описанных в классе Polynom.

1. Реализуйте приведенную программу и проверьте результат ее работы.
2. Проверьте работу всех методов, реализованных в классе Polynom.
3. Внесите изменения в main() так, чтобы стало возможным задавать параметры полиномов в диалоговом режиме с пользователем.
4. Результаты работы представьте в отчет.

Задание 2

Разработайте класс LinkedList (связный список) со следующими характеристиками:

- должно быть несколько конструкторов, которые позволяют создавать объект (связанный список) без элементов (пустой), на основе одного элемента, на основе задаваемого массива элементов и на основе ранее созданного объекта-списка;
- методы должны позволять добавлять новый элемент: по-умолчанию, в конец списка; с указанием местоположения нового элемента; задаваемым списком элементов в конец списка и по указанному расположению; задаваемым объектом-списком, по-умолчанию в конец списка или по указанному местоположению;
- методы должны позволять удалить элемент (последовательность элементов, список элементов): по-умолчанию, последний в списке; с указанием местоположения удаляемого элемента; с указанием удаляемого элемента; по задаваемому списку элементов; по задаваемому объекту-списку;
- методы должны позволять проверять целостность связанного списка, т. е. возможность «пройти» по списку от метки начала списка до метки конца списка; находить в списке минимальный и максимальный элемент; строить новый объект-список из элементов существующего объекта-списка по определенному критерию (например, по возрастанию или убыванию; только отрицательные или только положительные; больше некоторого значения и т. п.).

Программный код и результаты работы методов (с возможными пояснениями) представьте в отчете.