**Лабораторная работа № 2**

**Тема:** Классы и объекты в Java

**Цель работы.** Изучить принципы объявления классов и создания объектов на их основе.

Фундаментальной единицей программирования в Java является *класс*. В состав классов входят методы — фрагменты выполняемого кода, в которых происходят все вычисления. Классы также определяют структуру объектов и обеспечивают механизмы для их создания на основе определения класса. В программах можно использовать только предопределенные типы данных — целые, вещественные и т.д., но на практике в программах на Java создаются объекты и работа выполняется над ними.

В объектно-ориентированном программировании между тем, *что* делается, и тем, *как* это делается, существуют четкие различия. “Что” описывается в виде набора методов и данных, а также связанной с ними семантики. Такое сочетание (методы, данные и семантика) часто называется *контрактом* между разработчиком класса и программистом, использующим этот класс, так как оно определяет, что происходит при вызове конкретных методов объекта. Обычно считается, что объявленные в классе методы составляют все содержание его контракта.

На вопрос “как” отвечает класс, на основе которого создавался данный объект. Класс определяет реализацию методов, поддерживаемых объектом. Каждый объект представляет собой *экземпляр* класса. При вызове метода объекта выполняемый код ищется в классе. Объект может использовать и другие объекты. В простых классах все методы реализуются непосредственно.

Java, как и любой другой объектно-ориентированный язык программирования, располагает средствами построения классов и объектов. Каждый объект в Java имеет *тип*; им является тот класс, к которому принадлежит данный объект. В каждом классе есть члены двух видов: поля и методы.

Полями называются переменные, содержащие данные класса и его объектов. В них хранятся результаты вычислений, выполняемых методами данного класса.

Методы содержат исполняемый код класса. Методы состоят из операторов; эти операторы, а также способ вызова методов в конечном счете определяют процесс выполнения программы.

Так может выглядеть объявление простого класса, представляющего точку на плоскости:

```
class Point {
       public double x, y;
}
```

Представленный класс Point содержит два поля с координатами x и y точки, и в нем нет ни одного метода. Подобное объявление класса определяет, как будут выглядеть объекты, созданные на его основе, а также задает поведение объектов с помощью ряда инструкций. Члены класса могут обладать различными правами доступа. Объявление полей x и y класса Point с ключевым словом public означает, что любой метод программы, получивший доступ к объекту Point, сможет прочитать или изменить эти поля. Разрешается ограничить доступ к данным и предоставлять его лишь методам самого класса или связанных с ним классов.

Объекты создаются посредством выражений, в которых используется ключевое слово new. Созданные на основе определения класса объекты часто называют *экземплярами* данного класса. В языке Java создаваемые объекты размещаются в области

системной памяти, которая называется *кучей (heap)*. Доступ к любому объекту осуществляется с помощью *ссылки на объект* — вместо самого объекта в переменных содержится лишь ссылка на него. Когда ссылка не относится ни к какому объекту, она равна null.

Обычно между самим объектом и ссылкой на него не делается особых различий — можно сказать “передать методу объект”, на самом деле имея в виду “передать методу ссылку на объект”.

Пусть разрабатывается графическое приложение, в котором приходится следить за множеством точек (класс Point). Каждая точка представляется отдельным объектом Point.

Вот как может выглядеть создание и инициализация объектов Point:

```
Point lowerLeft = new Point();
Point upperRight = new Point();
Point middlePoint = new Point();

lowerLeft.x = 0.0;
lowerLeft.y = 0.0;

upperRight.x = 1280.0;
upperRight.y = 1024.0;

middlePoint.x = 640.0;
middlePoint.y = 512.0;
```

Каждый объект класса Point обладает собственной копией полей x и y. Например, изменение поля x объекта lowerLeft никак не влияет на значение x объекта upperRight. Поля объектов иногда называют *переменными экземпляра (instance variables)*, поскольку в каждом объекте (экземпляре) класса содержится отдельная копия этих полей.

***Статические поля.*** Чаще всего бывает нужно, чтобы значение поля одного объекта отличалось от значений одноименных полей во всех остальных объектах того же класса. Тем не менее иногда возникает необходимость совместного использования поля всеми объектами класса. Такие совместные поля также называются *переменными класса* — то есть переменными, относящимися ко всему классу, в отличие от переменных, относящихся к его отдельным объектам.

Чтобы использовать поле для хранения информации, относящейся ко всему классу, следует объявить его с ключевым словом static, поэтому такие поля иногда называют *статическими*. Например, объект Point, представляющий начало координат, может встречаться достаточно часто, поэтому имеет смысл выделить ему отдельное статическое поле в классе Point:

```
public static Point origin = new Point();
```

Если это объявление встретится внутри объявления класса Point, то появится ровно один экземпляр данных с именем Point.origin, который всегда будет ссылаться на объект (0,0). Поле static будет присутствовать всегда, независимо от того, сколько существует объектов Point (даже если не было создано ни одного объекта). Значения x и y равны нулю, потому что числовые поля, которым не было присвоено начального значения, по умолчанию инициализируются нулями.

Под термином “поле” обычно имеется в виду поле, уникальное для каждого объекта, хотя в отдельных случаях для большей ясности может использоваться термин “нестатическое поле”.

Неиспользуемые объекты Java автоматически уничтожаются *сборщиком мусора*. Сборщик мусора работает в фоновом режиме и следит за ссылками на объекты. Когда ссылок на объект больше не остается, появляется возможность убрать его из кучи, где он

временно хранился, хотя само удаление может быть отложено до более подходящего момента.

Объекты определенного выше класса Point могут быть изменены в любом фрагменте программы, в котором имеется ссылка на объект Point, поскольку поля этого класса объявлены с ключевым словом public. Класс Point представляет собой простейший пример класса. На самом деле иногда можно обойтись и простыми классами — например, при выполнении чисто внутренних задач пакета или когда хватает простейших типов данных.

Настоящие преимущества объектно-ориентированного программирования проявляются в возможности спрятать реализацию класса. В языке Java операции над классом осуществляются с помощью методов класса — инструкций, которые выполняются над данными объекта, чтобы получить нужный результат. В методах часто используются такие детали реализации класса, которые должны быть скрыты от всех остальных объектов. Данные скрываются в методах и становятся недоступными для всех прочих объектов — в этом заключается основной смысл инкапсуляции данных.

Каждый метод имеет ноль или более параметров. Метод может возвращать значение или объявляться с ключевым словом void, которое означает, что метод ничего не возвращает. Операторы метода содержатся в блоке между фигурными скобками { и }, которые следуют за именем метода и объявлением его *сигнатуры*. Сигнатурой называется имя метода, сопровождаемое числом и типом его параметров. Можно усовершенствовать класс Point и добавить в него простой метод clear, который выглядит так:

```
public void clear() {
 x = 0;
 y = 0;
}
```

Метод clear не имеет параметров, поскольку в скобках ( и ) после его имени ничего нет; кроме того, этот метод объявляется с ключевым словом void, поскольку он не возвращает никакого значения. Внутри метода разрешается прямое именование полей и методов класса — можно просто написать x и y, без ссылки на конкретный объект.

Объекты обычно не работают непосредственно с данными других объектов, хотя, класс *может* сделать свои поля общедоступными. И все же в хорошо спроектированном классе данные обычно скрываются, чтобы они могли изменяться только методами этого класса. Чтобы *вызвать* метод, необходимо указать имя объекта и имя метода и разделить их точкой (.). Параметры передаются методу в виде заключенного в скобки списка значений, разделяемых запятыми. Даже если метод вызывается без параметров, все равно необходимо указать пустые скобки. Объект, для которого вызывается метод (объект, получающий запрос на вызов метода) носит название *объекта-получателя*, или просто *получателя*.

В качестве результата работы метода может возвращаться только одно значение. Чтобы метод возвращал несколько значений, следует создать специальный объект, единственное назначение которого — хранение возвращаемых значений, и вернуть этот объект.

Ниже приводится метод с именем distance, который входит в класс Point, использованный в предыдущих примерах. Метод distance принимает в качестве параметра еще один объект Point, вычисляет евклидово расстояние между двумя точками и возвращает результат в виде вещественного значения с двойной точностью:

```
public double distance(Point that) {
 double xdiff, ydiff;
 xdiff = x – that.x;
 ydiff = y – that.y;
 return Math.sqrt(xdiff * xdiff + ydiff * ydiff);
}
```

Для объектов lowerLeft и upperRight, которые были определены выше вызов метода distance может выглядеть так:

```
double d = lowerLeft.distance(upperRight);
```

После выполнения этого оператора переменная d будет содержать евклидово расстояние между точками lowerLeft и upperRight.

**Ссылка this.** Иногда объекту-получателю бывает необходимо знать ссылку на самого себя. Например, объект-получатель может захотеть внести себя в какой-нибудь список объектов. В каждом методе может использоваться this — ссылка на текущий объект (объект-получатель). Следующее определение clear эквивалентно приведенному выше:

```
public void clear() {
 this.x = 0;
 this.y = 0;
}
```

Ссылка this часто используется в качестве параметра для тех методов, которым нужна ссылка на объект. Кроме того, this также может применяться для именования членов текущего объекта. Вот еще один из методов Point, который называется move и служит для присвоения полям x и y определенных значений:

```
public void move(double x, double y) {
 this.x = x;
 this.y = y;
}
```

В методе move ссылка this помогает разобраться, о каких x и y идет речь. Присвоить аргументам move имена x и y вполне разумно, поскольку в этих параметрах методу передаются координаты x и y точки. Но тогда получается, что имена параметров совпадают с именами полей Point, и имена параметров *скрывают* имена полей. Если бы просто написали x = x, то значение параметра x было бы присвоено самому параметру, а не полю x. Выражение this.x определяет поле x объекта, а не параметр x метода move.

**Статические методы.** Как было сказано выше в классе могут присутствовать статические поля, относящиеся к классу в целом, а не к его конкретным экземплярам. По аналогии с ними могут существовать и статические методы, также относящиеся ко всему классу, которые часто называют *методами класса*. Статические методы обычно предназначаются для выполнения операций, специфичных для данного класса, и работают со статическими полями, а не с конкретными экземплярами класса. Методы класса объявляются с ключевым словом static и называются статическими методами.

В реализации метода distance использован статический метод Math.sqrt для вычисления квадратного корня. Класс Math содержит множество методов для часто встречающихся математических операций. Эти методы объявлены статическими, так как они работают не с каким-то определенным объектом, но составляют внутри класса группу со сходными функциями.

Статический метод не может напрямую обращаться к нестатическим членам. При вызове статического метода не существует ссылки на конкретный объект, для которого вызывается данный метод. Впрочем, это ограничение можно обойти, передавая ссылку на конкретный объект в качестве параметра статического метода. Тем не менее в общем случае статические методы выполняют свои функции на уровне всего класса, а нестатические методы работают с конкретными объектами.

**Задания к работе**

**Задание 1.** Реализуйте приведенный выше пример следующим образом.

**Этап 1**

1. Создайте новый Java-проект.
2. В этом проекте создайте новый класс **LabRob** с методом main.
3. В этом же проекте создайте еще один класс **Point**, в котором нет метода main.
4. В класс Point добавьте строку

```
public double x, y;
```

5. Проверьте работу класса Point. Для этого в метод main класса **LabRob** добавьте следующие строки

```
Point lowerLeft = new Point();
Point upperRight = new Point();
Point middlePoint = new Point();

lowerLeft.x = 0.0;
lowerLeft.y = 0.0;

upperRight.x = 1280.0;
upperRight.y = 1024.0;

middlePoint.x = 640.0;
middlePoint.y = 512.0;

System.out.println("lower left > "+lowerLeft.x+","+lowerLeft.y);
System.out.println("upper Right > "+upperRight.x+","+upperRight.y);
System.out.println("middle Point > "+middlePoint.x+","+middlePoint.y);
```

6. Теперь запустите на выполнение программу (класс **LabRob**). Созданный Java-код двух классов и результат выполнения занесите в отчет.

**Этап 2**

1. Добавьте в класс Point новое статическое поле при помощи следующей строки:

```
public static Point origin = new Point();
```

2. Проверьте работу класса Point и нового поля в нем. Для этого в метод main класса **LabRob** добавьте следующие строки

```
System.out.println("origin Point > "+Point.origin.x+","+Point.origin.y);
```

3. Запустите на выполнение программу. Результат выполнения занесите в отчет.

**Этап 3**

1. Добавьте в класс Point новый метод при помощи следующих строк:

```
public void clear() {
   x = 0;
   y = 0;
}
```

2. Проверьте работу класса Point и нового его метода. Для этого в метод main класса **LabRob** добавьте следующие строки

```
middlePoint.clear();
System.out.println("middle Point > "+middlePoint.x+","+middlePoint.y);
```

3. Запустите на выполните класс **LabRob** и сравните вывод результата для объекта middlePoint до применения метода clear и после него. Результат занесите в отчет.

**Этап 4**

1. Добавьте в класс Point новый метод при помощи следующих строк:

```
public double distance (Point that) {
        double xdiff, ydiff;
                System.out.println("x="+x);
                System.out.println("this.x="+this.x);
                System.out.println("that.x="+that.x);
        xdiff = this.x - that.x;
        ydiff = this.y - that.y;
        return Math.sqrt(xdiff * xdiff + ydiff * ydiff);
}
```

2. Проверьте работу класса Point и нового метода distance. Для этого в метод main класса **LabRob** добавьте следующие строки

```
        double d = upperRight.distance(middlePoint);
                System.out.println("d = "+d);
```

3. Запустите на выполнение программу. Результат выполнения занесите в отчет. Измените объекты, для которых выполняется метод distance. Что изменилось? Результат занесите в отчет.

**Задание 2.** Напишите простой класс Vehicle (транспортное средство) для хранения информации об автомашине, в котором предусмотрены (как минимум) поля для задания текущей скорости, текущего направления в градусах и имени владельца.

**Задание 3.** При решении практических задач иногда возникает необходимость реализовать математические функции или операции, или переопределить уже реализованные. Ниже приведен пример класса, в котором реализуются некоторые математические функции:

```
package mathdemo;

public class MyMath {
// Класс с математическими функциями:
// Интервал разложения в ряд Фурье:
      static double L=Math.PI;
// Экспонента:
      static double Exp(double x,int N){
            int i;
            double s=0,q=1;
            for(i=0;i<N;i++){
                  s+=q;
                  q*=x/(i+1);
                  }
          return s+q;
      }
// Синус:
      static double Sin(double x,int N){
            int i;
            double s=0,q=x;
            for(i=0;i<N;i++){
                  s+=q;
                  q*=(-1)*x*x/(2*i+2)/(2*i+3);
                  }
            return s+q;
      }
// Косинус:
      static double Cos(double x,int N){
            int i;
            double s=0,q=1;
            for(i=0;i<N;i++){
                  s+=q;
                  q*=(-1)*x*x/(2*i+1)/(2*i+2);
                  }
            return s+q;
            }
// Функция Бесселя:
      static double BesselJ(double x,int N){
            int i;
            double s=0,q=1;
            for(i=0;i<N;i++){
                  s+=q;
                  q*=(-1)*x*x/4/(i+1)/(i+1);
                  }
            return s+q;
            }
// Ряд Фурье по синусам:
      static double FourSin(double x,double[] a){
            int i,N=a.length;
            double s=0;
```

```
            for(i=0;i<N;i++){
                  s+=a[i]*Math.sin(Math.PI*x*(i+1)/L);
                  }
            return s;}
// Ряд Фурье по косинусам:
      static double FourCos(double x,double[] a){
            int i,N=a.length;
            double s=0;
            for(i=0;i<N;i++){
                  s+=a[i]*Math.cos(Math.PI*x*i/L);
                  }
            return s;
            }
}
package mathdemo;

public class MathDemo {

      public static void main(String[] args) {
                  System.out.println("Примеры вызова функций:");
                  // Вычисление экспоненты:
                  System.out.println("exp(1)="+MyMath.Exp(1,30));
                  // Вычисление синуса:
                  System.out.println("sin(pi)="+MyMath.Sin(Math.PI,100));
                  // Вычисление косинуса:
                  System.out.println("cos(pi/2)="+MyMath.Cos(Math.PI/2,100));
                  // Вычисление функции Бесселя:

      System.out.println("J0(mu1)="+MyMath.BesselJ(2.404825558,100));
      // Заполнение массивов коэффициентов рядов Фурье для функции y(x)=x:
                  int m=1000;
                  double[] a=new double[m];
                  double[] b=new double[m+1];
                  b[0]=MyMath.L/2;
                  for(int i=1;i<=m;i++){
                  a[i-1]=(2*(i%2)-1)*2*MyMath.L/Math.PI/i;
                  b[i]=-4*(i%2)*MyMath.L/Math.pow(Math.PI*i,2);}
                  // Вычисление функции y(x)=x через синус-ряд Фурье:
                  System.out.println("2.0->"+MyMath.FourSin(2.0,a));
                  // Вычисление функции y(x)=x через косинус-ряд Фурье:
                  System.out.println("2.0->"+MyMath.FourCos(2.0,b));
      }

}
```

В представленной программе каждая функция имеет по два аргумента: первый аргумент типа double определяет непосредственно аргумент математической функции, а второй целочисленный аргумент определяет количество слагаемых в ряде Тейлора, на основании которых вычисляется функция.
Для экспоненты использован ряд:

$$\exp(x)\approx\sum_{k=0}^{N}\frac{x^{k}}{k!}$$

Синус и косинус вычисляются соответственно по формулам:

$$\sin(x)\approx\sum_{k=0}^{N}\frac{(-1)^{k}x^{2k+1}}{(2k+1)!}$$

$$\cos(x) \approx \sum_{k=0}^{N} \frac{(-1)^k x^{2k}}{(2k)!}$$

Для функции Бесселя нулевого индекса использован ряд:

$$J_0(x) \approx \sum_{k=0}^{N} \frac{(-1)^k (x/2)^{2k}}{(k!)^2}$$

Для вычисления функции Бесселя в качестве первого аргумента указано значение нуля функции Бесселя $\mu_1 \approx 2.404825558$. Нулями $\mu_n$ (n = 1, 2, ... ) функции Бесселя нулевого индекса $J_0(x)$ называются неотрицательные решения уравнения $J_0(\mu_n)=0$. Поэтому для указанного аргумента значение функции должно быть нулевым в пределах точности вычислений. Проверьте результат вычислений функции в программе.

В классе MyMath также определены функции для вычисления рядов Фурье по базовым функциям $\sin(\frac{\pi n x}{L})$ (функция FourSin() ) и $\cos(\frac{\pi n x}{L})$ (функция FourCos() ). В частности, у функции FourSin() два аргумента: первый - это переменная *x* типа double , второй - массив *a* с элементами типа double. В качестве результата функция возвращает сумму вида:

$$\sum_{n=1}^{N} a_{n-1} \sin(\frac{\pi n x}{L}) \quad ,$$

где $a_n$ - элементы массива *a*; верхняя граница суммы N определяется по количеству элементов этого массива (команда N=a.length ), а сама сумма представляет собой разложение в ряд (точнее, ограниченное количество слагаемых ряда) на интервале от 0 до *L* некоторой функции с коэффициентами разложения, записанными в массив *a*. Параметр *L* объявлен в классе MyMath как статическое поле со значением *Math.PI*.

Дополните class MyMath тремя функциями или операциями и выполните демонстрацию их работы через MathDemo.