Программирование на языке Java. Классы

Картузов А.В.

Базовым элементом объектно-ориентированного программирования в языке Java является класс. В этой главе Вы научитесь создавать и расширять свои собственные классы, работать с экземплярами этих классов и начнете использовать мощь объектно-ориентированного подхода. Напомним, что классы в Java не обязательно должны содержать метод main. Единственное назначение этого метода—указать интерпретатору Java, откуда надо начинать выполнение программы. Для того, чтобы создать класс, достаточно иметь исходный файл, в котором будет присутствовать ключевое слово class, и вслед за ним—допустимый идентификатор и пара фигурных скобок для его тела.

```
class Point {
        }
```

Замечание:

Имя исходного файла Java должно соответствовать имени хранящегося в нем класса. Регистр букв важен и в имени класса, и в имени файла.

Как вы помните из главы 2, класс—это шаблон для создания объекта. Класс определяет структуру объекта и его методы, образующие функциональный интерфейс. В процессе выполнения Java-программы система использует определения классов для создания представителей классов. Представители являются реальными объектами. Термины <представитель>, <экземпляр> и <объект> взаимозаменяемы. Ниже приведена общая форма определения класса.

```
class имя_класса extends имя_суперкласса {
        type переменная1_объекта:
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
type переменная2_объекта:
type переменнаяN_объекта:
type имяметода1(список_параметров) {
        тело метода;
        }
type имяметода2(список_параметров) {
        тело метода;
        }
type имя методаM(список_параметров) {
        тело метода;
        }
}
```

Ключевое слово extends указывает на то, что <имя_класса>—это подкласс класса <имя_суперкласса>. Во главе классовой иерархии Java стоит единственный ее встроенный класс—Object. Если вы хотите создать подкласс непосредственно этого класса, ключевое слово extends и следующее за ним имя суперкласса можно опустить—транслятор включит их в ваше определение автоматически. Примером может служить класс Point, приведенный ранее.

1. Переменные представителей (instance variables)

Данные инкапсулируются в класс путем объявления переменных между открывающей и закрывающей фигурными скобками, выделяющими в определении класса его тело. Эти переменные объявляются точно так же, как объявлялись локальные переменные в предыдущих примерах. Единственное отличие состоит в том, что их надо объявлять вне методов, в том числе вне метода main. Ниже приведен фрагмент кода, в котором объявлен класс Point с двумя переменными типа int.

```
class Point { int x, y;
}
```

В качестве типа для переменных объектов можно использовать как любой из простых типов, описанных в главе 4, так и классовые типы. Скоро мы добавим к приведенному выше классу метод main, чтобы его можно было запустить из командной строки и создать несколько объектов.

Copyright © 1999-2004 Javable.com. All rights reserved.

2. Оператор new

Оператор new создает экземпляр указанного класса и возвращает ссылку на вновь созданный объект. Ниже приведен пример создания и присваивание переменной p экземпляра класса Point.

```
Point p = new Point();
```

Вы можете создать несколько ссылок на один и тот же объект. Приведенная ниже программа создает два различных объекта класса Point и в каждый из них заносит свои собственные значения. Оператор точка используется для доступа к переменным и методам объекта.

```
class TwoPoints {
        public static void main(String args[]) {
                Point p1 = new Point();
                Point p2 = new Point();
                p1.x = 10;
                p1.y = 20;
                p2.x = 42;
                p2.y = 99;
                System.out.println("x = " + p1.x + " y = " + p1.y);
                System.out.println("x =  " + p2.x + " y = " + p2.y);
                }
        }
```

В этом примере снова использовался класс Point, было создано два объекта этого класса, и их переменным x и y присвоены различные значения. Таким образом мы продемонстрировали, что переменные различных объектов независимы на самом деле. Ниже приведен результат, полученный при выполнении этой программы.

```
C:\> Java TwoPoints
x = 10 y = 20
x = 42 y = 99
```

Замечание:

Поскольку при запуске интерпретатора мы указали в командной строке не класс Point, а класс TwoPoints, метод main класса Point был полностью проигнорирован. Добавим в класс Point метод main и, тем самым,

Copyright © 1999-2004 Javable.com. All rights reserved.

получим законченную программу.

```
class Point { int x, y;
        public static void main(String args[]) {
                Point p = new Point();
                p.x = 10;
                p.y = 20;
                System.out.println("x =  " + p.x + " y = " + p.y);
                }
        }
```

3. Объявление методов

Методы—это подпрограммы, присоединенные к конкретным определениям классов. Они описываются внутри определения класса на том же уровне, что и переменные объектов. При объявлении метода задаются тип возвращаемого им результата и список параметров. Общая форма объявления метода такова:

```
        тип имя_метода (список формальных параметров) {
                тело метода:
                }
```

Тип результата, который должен возвращать метод может быть любым, в том числе и типом void—в тех случаях, когда возвращать результат не требуется. Список формальных параметров—это последовательность пар тип-идентификатор, разделенных запятыми. Если у метода параметры отсутствуют, то после имени метода должны стоять пустые круглые скобки.

```
        class Point { int x, y;
                void init(int a, int b) {
                x = a;
                y = b;
                }
        }
```

4. Вызов метода

Copyright © 1999-2004 Javable.com. All rights reserved.

В Java отсутствует возможность передачи параметров по ссылке на примитивный тип. В Java все параметры примитивных типов передаются по значению, а это означает, что у метода нет доступа к исходной переменной, использованной в качестве параметра. Заметим, что все объекты передаются по ссылке, можно изменять содержимое того объекта, на который ссылается данная переменная. В главе 12 Вы узнаете, как предать переменные примитивных типов по ссылке (через обрамляющие классы-оболочки).

5. Скрытие переменных представителей

В языке Java не допускается использование в одной или во вложенных областях видимости двух локальных переменных с одинаковыми именами. Интересно отметить, что при этом не запрещается объявлять формальные параметры методов, чьи имена совпадают с именами переменных представителей. Давайте рассмотрим в качестве примера иную версию метода init, в которой формальным параметрам даны имена x и y, а для доступа к одноименным переменным текущего объекта используется ссылка this.

```
class Point { int x, y;
        void init(int x, int y) {
                this.x = x;
                this.y = y }
        }

class TwoPointsInit {
        public static void main(String args[]) {
                Point p1 = new Point();
                Point p2 = new Point();
                p1.init(10,20);
                p2.init(42,99);
                System.out.println("x = " + p1.x + " y = " + p-l.y);
                System.out.printlnC'x = " + p2.x + " y = " + p2.y);
                }
}
```

6. Конструкторы

Инициализировать все переменные класса всякий раз, когда создается его очередной

Copyright © 1999-2004 Javable.com. All rights reserved.

представитель—довольно утомительное дело даже в том случае, когда в классе имеются функции, подобные методу init. Для этого в Java предусмотрены специальные методы, называемые конструкторами. Конструктор—это метод класса, который инициализирует новый объект после его создания. Имя конструктора всегда совпадает с именем класса, в котором он расположен (также, как и в C++). У конструкторов нет типа возвращаемого результата—никакого, даже void. Заменим метод init из предыдущего примера конструктором.

```
class Point {
        int x, y;
        Point(int x, int y) {
                this.x = x;
                this.y = y;
                }
        }

class PointCreate {
        public static void main(String args[]) {
                Point p = new Point(10,20);
                System.out.println("x = " + p.x + " y = " + p.y);
                }
        }
```

Программисты на Pascal (Delphi) для обозначения конструктора используют ключевое слово constructor.

7. Совмещение методов

Язык Java позволяет создавать несколько методов с одинаковыми именами, но с разными списками параметров. Такая техника называется совмещением методов (method overloading). В качестве примера приведена версия класса Point, в которой совмещение методов использовано для определения альтернативного конструктора, который инициализирует координаты x и y значениями по умолчанию (-1).

```
class Point {
        int x, y;
        Point(int x, int y) {
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                this.x = x;
                this.y = y;
                }

        Point() {
                x = -1;
                y = -1;
                }
        }

class PointCreateAlt {
        public static void main(String args[]) {
                Point p = new Point();
                System.out.println("x = " + p.x + " y = " + p.y);
                }
        }
```

В этом примере объект класса Point создается не при вызове первого конструктора, как это было раньше, а с помощью второго конструктора без параметров. Вот результат работы этой программы:

```
C:\> java PointCreateAlt
x = -1 y = -1
```

Замечание:

Решение о том, какой конструктор нужно вызвать в том или ином случае, принимается в соответствии с количеством и типом параметров, указанных в операторе new. Недопустимо объявлять в классе методы с одинаковыми именами и сигнатурами. В сигнатуре метода не учитываются имена формальных параметров учитываются лишь их типы и количество.

8. this в конструкторах

Очередной вариант класса Point показывает, как, используя this и совмещение методов, можно строить одни конструкторы на основе других.

```
class Point {
        int x, y;
        Point(int x, int y) {
                this.x = x;
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                this.y = y;
                }
        Point() {
                this(-1, -1);
        }
}
```

В этом примере второй конструктор для завершения инициализации объекта обращается к первому конструктору.

Методы, использующие совмещение имен, не обязательно должны быть конструкторами. В следующем примере в класс Point добавлены два метода distance. Функция distance возвращает расстояние между двумя точками. Одному из совмещенных методов в качестве параметров передаются координаты точки x и y, другому же эта информация передается в виде параметра-объекта Point.

```
class Point { int x, y;
        Point(int x, int y) {
                this.x = x;
                this. y = y;
                }

        double distance(int x, int y) {
                int dx = this.x—x;
                int dy = this.y—y;
                return Math.sqrt(dx*dx + dy*dy);
                }

        double distance(Point p) {
        return distance(p.x, p.y);
        }
}

class PointDist {
        public static void main(String args[]) {
                Point p1 = new Point(0, 0);
                Point p2 = new Point(30, 40);
                System.out.println("p1 = " + pi.x + ", " + p1.y);
                System.out.println("p2 = " + p2.x + ", " + p2.y);
                System.out.println("p1.distance(p2) = " +
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                        p1.distance(p2));
                System.out.println("p1.distance(60, 80) = " +
                        p1.distance(60, 80));
                }
        }
```

Обратите внимание на то как во второй фороме метода distance для получения результата вызывается его первая форма. Ниже приведен результат работы этой программы:

```
C:\> java PointDist
p1 = 0, 0
p2 = 30, 40
p1.distance(p2) = 50.0
p1.distance(60, 80) = 100.0
```

9. Наследование

Вторым фундаментальным свойством объектно-ориентированного подхода является наследование (первый—инкапсуляция). Классы-потомки имеют возможность не только создавать свои собственные переменные и методы, но и наследовать переменные и методы классов-предков. Классы-потомки принято называть подклассами. Непосредственного предка данного класса называют его суперклассом. В очередном примере показано, как расширить класс Point таким образом, чтобы включить в него третью координату z.

```
class Point3D extends Point {
        int z;
        Point3D(int x, int y, int z) {
                this.x = x;
                this.y = y;
                this.z = z; }
        Point3D() {
                this(-1,-1,-1);
                }
        }
```

В этом примере ключевое слово extends используется для того, чтобы сообщить транслятору о намерении создать подкласс класса Point. Как видите, в этом классе не

Copyright © 1999-2004 Javable.com. All rights reserved.

понадобилось объявлять переменные x и y, поскольку Point3D унаследовал их от своего суперкласса Point.

Замечание:

Вероятно, программисты, знакомые с C++, очевидно ожидают, что сейчас мы начнем обсуждать концепцию множественного наследования. Под множественным наследованием понимается создание класса, имеющего несколько суперклассов. Однако в языке Java ради обеспечения высокой производительности и большей ясности исходного кода множественное наследование реализовано не было. В большинстве случаев, когда требуется множественное наследование, проблему можно решить с помощью имеющегося в Java механизма интерфейсов, описанного в следующей главе.

10. super

В примере с классом Point3D частично повторялся код, уже имевшийся в суперклассе. Вспомните, как во втором конструкторе мы использовали this для вызова первого конструктора того же класса. Аналогичным образом ключевое слово super позволяет обратиться непосредственно к конструктору суперкласса (в Delphi / C++ для этого используется ключевое слово inherited).

```
class Point3D extends Point {
        int z;
        Point3D(int x, int y, int z) {
                super(x, y);     // Здесь мы вызываем конструктор
                                 // суперкласса this.z=z;
                public static void main(String args[]) {
                        Point3D p = new Point3D(10, 20, 30);
                        System.out.println( " x = " +  p.x +
                        " y = " + p.y + " z = " + p.z);
                        }
                }
```

Вот результат работы этой программы:

```
C:\> java Point3D
x = 10 y = 20 z = 30
```

11. Замещение методов

Copyright © 1999-2004 Javable.com. All rights reserved.

Новый подкласс Point3D класса Point наследует реализацию метода distance своего суперкласса (пример PointDist.java). Проблема заключается в том, что в классе Point уже определена версия метода distance(mt x, int y), которая возвращает обычное расстояние между точками на плоскости. Мы должны заместить (override) это определение метода новым, пригодным для случая трехмерного пространства. В следующем примере проиллюстрировано и совмещение (overloading), и замещение (overriding) метода distance.

```
class Point {
        int x, y;
        Point(int x, int y) {
                this.x = x;
                this.y = y;
                }

        double distance(int x, int y) {
                int dx = this.x—x;
                int dy = this.y—y:
                return Math,sqrt(dx*dx + dy*dy);
                }

        double distance(Point p) {
                return distance(p.x, p.y);
                }
        }

class  Point3D extends Point {
        int z;
        Point3D(int x, int y, int z)   {
                super(x, y);
                this.z = z;
                }

        double distance(int x, int y,  int z) {
                int dx = this.x—x;
                int dy = this.y—y;
                int dz = this.z— z;
                return Math.sqrt(dx*dx + dy*dy + dz*dz);
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                }

        double distance(Point3D other) {
                return distance(other.x, other.y, other.z);
                }

        double distance(int x, int y)  {
                double dx = (this.x / z)—x;
                double dy = (this.y / z)—y;
                return Math.sqrt(dx*dx + dy*dy);
                }
        }

class Point3DDist {
        public static void main(String args[]) {
                Point3D p1 = new Point3D(30, 40, 10);
                Point3D p2 = new Point3D(0, 0, 0);
                Point p = new Point(4, 6);
                System.out.println("p1="+p1.x+",
                 "+p1.y+","+p1.z);
                System.out.println("p2="+p2.x+",
                 "+p2.y+","+p2.z);
                System.out.println("p="+p.x+","+p.y);
                System.out.println("p1.distance(p2)=
                 "+p1.distance(p2));
                System.out.println("p1.distance(4,6)=
                 "+p1.distance(4,6));
                System.out.println("p1.distance(p)=
                 "+p1.distance(p));
                }
        }
```

Ниже приводится результат работы этой программы:

```
C:\> Java Point3DDist
p1 = 30, 40, 10
p2 = 0, 0, 0
p = 4, 6
p1.distance(p2) = 50.9902
p1.distance(4, 6) = 2.23607
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
        p1.distance(p) = 2.23607
```

Обратите внимание—мы получили ожидаемое расстояние между трехмерными точками и между парой двумерных точек. В примере используется механизм, который называется динамическим назначением методов (dynamic method dispatch).

12. Динамическое назначение методов

Давайте в качестве примера рассмотрим два класса, у которых имеют простое родство подкласс / суперкласс, причем единственный метод суперкласса замещен в подклассе.

```
class A {
        void callme() {
                System.out.println("Inside A's callrne method");
        }}
class B extends A {
        void callme() {
                System.out.println("Inside B's callme method");
        }}

class Dispatch {
        public static void main(String args[]) {
                A a = new B();
                a.callme();
                }
        }
```

Обратите внимание—внутри метода main мы объявили переменную a класса A, а проинициализировали ее ссылкой на объект класса B. В следующей строке мы вызвали метод callme. При этом транслятор проверил наличие метода callme у класса A, а исполняющая система, увидев, что на самом деле в переменной хранится представитель класса B, вызвала не метод класса A, а callme класса B. Ниже приведен результат работы этой программы:

```
C:\> Java Dispatch
Inside B's calime method
```

Замечание:

Программистам Delphi / C++ следует отметить, что все Java по умолчанию являются виртуальными

Copyright © 1999-2004 Javable.com. All rights reserved.

функциями (ключевое слово virtual).

Рассмотренная форма динамического полиморфизма времени выполнения представляет собой один из наиболее мощных механизмов объектно-ориентированного программирования, позволяющих писать надежный, многократно используемый код.

13. final

Все методы и переменные объектов могут быть замещены по умолчанию. Если же вы хотите объявить, что подклассы не имеют права замещать какие-либо переменные и методы вашего класса, вам нужно объявить их как final (в Delphi / C++ не писать слово virtual).

```
        final int FILE_NEW = 1;
```

По общепринятому соглашению при выборе имен переменных типа final—используются только символы верхнего регистра (т.е. используются как аналог препроцерных констант C++). Использование final-методов порой приводит к выигрышу в скорости выполнения кода—поскольку они не могут быть замещены, транслятору ничто не мешает заменять их вызовы встроенным (in-line) кодом (байт-код копируется непосредственно в код вызывающего метода).

14. finalize

В Java существует возможность объявлять методы с именем finalize. Методы finalize аналогичны деструкторам в C++ (ключевой знак ~) и Delphi (ключевое слово destructor). Исполняющая среда Java будет вызывать его каждый раз, когда сборщик мусора соберется уничтожить объект этого класса.

15. static

Иногда требуется создать метод, который можно было бы использовать вне контекста какого-либо объекта его класса. Так же, как в случае main, все, что требуется для создания такого метода—указать при его объявлении модификатор типа static.

Copyright © 1999-2004 Javable.com. All rights reserved.

Статические методы могут непосредственно обращаться только к другим статическим методам, в них ни в каком виде не допускается использование ссылок this и super. Переменные также могут иметь тип static, они подобны глобальным переменным, то есть доступны из любого места кода. Внутри статических методов недопустимы ссылки на переменные представителей. Ниже приведен пример класса, у которого есть статические переменные, статический метод и статический блок инициализации.

```
        class Static {
                static int a = 3;
                static int b;
                static void method(int x) {
                        System.out.println("x = " + x);
                        System.out.println("a = " + a);
                        System.out.println("b = " + b);
                        }
                static {
                        System.out.println("static block initialized");
                        b = a * 4;
                        }
                public static void main(String args[]) {
                        method(42);
                        }
                }
```

Ниже приведен результат запуска этой программы.

```
C:\> java Static static block initialized
        X = 42
        A = 3
        B = 12
```

В следующем примере мы создали класс со статическим методом и несколькими статическими переменными. Второй класс может вызывать статический метод по имени и ссылаться на статические переменные непосредственно через имя класса.

```
        class StaticClass {
                static int a = 42;
                static int b = 99;
                static void callme() {
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                System.out.println("a = " + a);
                }
        }
        class StaticByName {
                public static void main(String args[]) {
                StaticClass.callme();
                System.out.println("b = " + StaticClass.b);
                }
        }
```

А вот и результат запуска этой программы:

```
C:\> Java StaticByName
        a = 42 b = 99
```

16. abstract

Бывают ситуации, когда нужно определить класс, в котором задана структура какой-либо абстракции, но полная реализация всех методов отсутствует. В таких случаях вы можете с помощью модификатора типа abstract объявить, что некоторые из методов обязательно должны быть замещены в подклассах. Любой класс, содержащий методы abstract, также должен быть объявлен, как abstract. Поскольку у таких классов отсутствует полная реализация, их представителей нельзя создавать с помощью оператора new. Кроме того, нельзя объявлять абстрактными конструкторы и статические методы. Любой подкласс абстрактного класса либо обязан предоставить реализацию всех абстрактных методов своего суперкласса, либо сам должен быть объявлен абстрактным.

```
        abstract class A {
                abstract void callme();
                void metoo() {
                        System.out.println("Inside A's metoo method");
                        }
                }
        class B extends A {
                void callme() {
                        System.out.println("Inside B's callme method");
                        }
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
                }
        class Abstract {
                public static void main(String args[]) {
                        A a = new B():
                        a.callme():
                        a.metoo():
                        }
                }
```

В нашем примере для вызова реализованного в под-классе класса A метода callme и реализованного в классе A метода metoo используется динамическое назначение методов, которое мы обсуждали раньше.

```
C:\> Java Abstract
        Inside B's callrne method Inside A's metoo method
```

17. Классическое заключение

В этой главе вы научились создавать классы, конструкторы и методы. Вы осознали разницу между совмещением (overloading) и замещением (overriding) методов. Специальные переменные this и super помогут вам сослаться на текущий объект и на его суперкласс. В ходе эволюции языка Java стало ясно, что в язык нужно ввести еще несколько организационных механизмов—возможности более динамичного назначения методов и возможности более тонкого управления пространством имен класса и уровнями доступа к переменным и методам объектов. Оба этих механизма—интерфейсы и пакеты, описаны в следующей главе.

Copyright © 1999-2004 Javable.com. All rights reserved.