УДК 681.3

АБСТРАКТНЫЕ БАЗОВЫЕ КЛАССЫ

А.В. Пахунов*

Аннотация. Методы класса имеют шанс быть объявленными абстрактными. Абстрактные методы пишут модификатором abstract.

Абстрактный класс похож на обычный класс. В абстрактном классе можно определить поля и методы, но нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников, в то время как производные классы уже реализуют этот функционал.

Ключевые слова: абстрактные базовые классы, abstract, функционал.

ABSTRACT BASE CLASSES

A.V. Pakhunov

Annotation. Methods of any class can be declared as abstract. Abstract methods must follow the modifier abstract.

Abstract class is similar to a normal class. In abstract class you can define fields and methods but at the same time you can not create an object or an instance of such class. Abstract classes are designed to provide basic functionality for the derived classes that can already implement this functionality.

Keywords: abstract base classes, abstract, functionality.

Научная специальность 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Абстрактные базовые классы — это особый способ для документирования кода, который помогает ограничить (decouple) в программе классов взаимодействие между отдельными абстракциями.

В объективно-ориентированном программировании (ООП) абстрактный класс — базовый, не предполагающий создания экземпляров. Все абстрактные классы осуществляют на практике полиморфизм — один из главных принципов ООП. В абстрактном классе могут быть (или не быть) абстрактные свойства и методы. Для класса, где он объявлен, абстрактный метод не реализуется, но для его неабстрактных потомков его реализация обязательна. Абстрактные классы — это самые общие абстракции, в которых сочетается наименьшее содержание с наибольшим объемом.

В некоторых языках запрещается создание экземпляров абстрактных классов, в других же это может быть допустимо (пример тому ObjectPascal (Delphi), но если прибегнуть в процессе выполнения программы к абстрактному методу объекта такого класса, то произойдет ошибка. Многие языки допускают объявление абстрактным любого класса, даже если абстрактных методов в нем нет (как в Java), как раз для того, чтобы создание экземпляров запретить. Абстрактный класс часто рассматривается как интерфейс к порожденному им семейству классов, но у него могут иметься, в отличие от интерфейса классического, определенные свойства и методы.

Часто появляется необходимость создания базового класса, где для его производных классов опреде-

* АНТОН ВАЛЕРЬЕВИЧ ПАХУНОВ,
кандидат технических наук, доцент кафедры автоматизированных систем управления
Московский автомобильно-дорожный государственный технический университет
Адрес: Ленинградский проспект, д. 64, Москва, Россия, 125319
E-mail: Anton_Pakhunov@mail.ru

лена только одна общая форма, а каждый из классов должен сам наполнять его деталями. В подобном классе можно определить только характер методов, обязанных конкретно реализоваться не в самом базовом классе, а в производных. Так бывает, например, если невозможно получение в базовом классе содержательной реализации метода.

Создавая собственные библиотеки классов, можно на деле увидеть, что часто у метода нет содержательного определения в контексте базового класса. У такой ситуации есть два способа разрешения. Первый — это простая выдача предупредительного сообщения. Он годится в таких случаях, как отладка, хотя обычно не используется в практике программирования. Здесь используется другой способ — он дает гарантию переопределения всех нужных методов в производном классе — такой способ есть в C#, это простое применение абстрактного метода.

Абстрактный метод создается с помощью указываемого модификатора типа abstract. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования.

Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора virtual. В действительности совместное использование модификаторов virtual и abstract считается ошибкой. Для определения абстрактного метода служит приведенная общая форма: abstract тип имя (список_параметров);

У абстрактного метода нет тела. Здесь модификатор abstract применим только для методов экземпляра, а в статических методах (static) — нет. Свойства и индексаторы также могут быть абстрактными.

Если в классе имеется один и больше одного абстрактного метода, он должен быть объявлен абстрактным, для чего указывается модификатор abstract перед объявлением class. Но так как реализация абстрактного класса полностью не определяется, значит, объектов у него быть не может. По этой причине любая попытка создания с помощью оператора new объекта абстрактного класса приведет к появлению ошибки при компиляции.

Если производный класс становится наследником абстрактного класса, то в нем обязательна реализация всех абстрактных методов базового класса. Иначе производный класс определится как abstract.

Следовательно, пока не произойдет полная реализация класса, наследуется атрибут abstract.

Очень часто в программах не требуется создавать объекты базовых классов, т.е. базовые классы нужны только для того, чтобы построить иерархию классов и определить общие свойства для производных классов. Такие классы можно сделать абстрактными (abstract class). При попытке создания объекта абстрактного класса компилятор выдаст ошибку.

Абстрактные классы используются в качестве обобщенных концепций, на основе которых можно создавать более конкретные производные классы. Невозможно создать объект типа абстрактного класса; однако, можно использовать указатели и ссылки на типы абстрактного класса.

Класс, содержащий хотя бы одну чисто виртуальную функцию, считается абстрактным. Классы, производные от абстрактного класса, должны реализовать все его чисто виртуальные функции, иначе они также будут абстрактными классами.

Виртуальная функция объявляется как «чистая» с помощью синтаксиса спецификатора-чистоты. Рассмотрим пример, представленный в разделе «Виртуальные функции». Класс Account создан для того, чтобы предоставлять общие функции, но объекты типа Account имеют слишком общий характер для практического применения. Таким образом, класс Account является хорошим кандидатом в абстрактный класс:

```
// deriv_AbstractClasses.cpp
// compile with: /LD
class Account {
public:
Account( double d ); // Constructor.
virtual double GetBalance(); // Obtain balance.
virtual void PrintBalance() = 0; // Pure virtual function.
private:
double _balance;
};
```

Единственное различие между этим и предыдущим объявлениями состоит в том, что функция PrintBalance объявлена со спецификатором чисто виртуальной функции pure (= 0).

Чтобы сделать класс абстрактным, нужно объявить одну из виртуальных функций чистой.

Чистая виртуальная функция (pure virtual function) как бы намекает, что она будет реализована в производных классах.

Чтобы сделать виртуальную функцию чистой (pure), нужно добавить после заголовка функции символы =0 (знак равенства и ноль):

```
class Base
{
public:
virtual void method () =0;
virtual ~Base() =0;
};
```

В данном случае уже не нужно писать определение такой функции. Помимо этого теперь нельзя создавать объекты класса Base, так как он стал абстрактным.

Символы =0 необязательно добавлять ко всем виртуальным функциям, достаточно добавить к одной.

Отличие абстрактных классов от интерфейсов в том, что в абстрактном классе возможно задать базовый функционал, который будет наследован потомками этого класса. Интерфейс лишь определяет свойства/методы/пр, что обязаны реализовать классы, реализующие этот интерфейс.

Любой класс, который содержит один или более абстрактных методов, должен быть также объявлен как абстрактный. Для этого достаточно поместить ключевое слово abstract перед ключевым словом class в начале объявления класса. Абстрактный класс не может содержать какие-то объекты. А абстрактный класс не может быть непосредственно конкретизирован с помощью операции new. Такие объекты были бы бесполезны, поскольку абстрактный класс определен не полностью. Нельзя также объявлять абстрактные конструкторы или абстрактные статические методы. Любой подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо также быть объявлен абстрактным.

Ниже приведен простой пример класса, содержащего абстрактный метод, и класса, который реализует этот метод.

```
// Простой пример применения абстракции.
abstract class A {
abstract void callme ();
// абстрактные классы все же могут содержать конкретные методы
void callmetoo() {
System.out.println("Это конкретный метод.");
}
}
class B extends A {
void callme() {
System.out.println("Реализация метода callme класса B.");
}
}
class AbstractDemo {
public static void main(String args[]) {
B b = new B () ;
b.callme();
b.callmetoo();
}
}
```

Обратите внимание, что в этой программе класс A не содержит объявлений каких-либо объектов. Как уже было сказано, конкретизация абстрактного класса невозможна.

И еще один нюанс: класс A реализует конкретный метод callmetoo (). Это вполне допустимо. Абстрактные классы могут содержать любое необходимое количество конкретных реализаций.

Хотя абстрактные классы не могут быть использованы для конкретизации объектов, их можно применять для создания ссылок на объекты, поскольку в Java полиморфизм времени выполнения реализован посредством ссылок на суперкласс. Поэтому должна существовать возможность создания ссылки на абстрактный класс, которая может использоваться для указания на объект подкласса. Применение этого свойства показано в следующем примере.

Используя абстрактный класс, можно усовершенствовать созданный ранее класс Figure. Поскольку понятие площади неприменимо к неопределенной двумерной фигуре, следующая версия программы объявляет метод area () внутри класса Figure как abstract. Конечно, это означает, что все классы, производные от Figure, должны переопределять метод area ().

```
// Использование абстрактных методов и классов.
abstract class Figure {
double diml;
double dim2;
Figure(double a, double b) {
diml = a;
dim2 = b;
}
// теперь метод area является абстрактным
abstract double area();
}
class Rectangle extends Figure {
Rectangle(double a, double b) {
super(a, b);
}
// переопределение метода area для четырехугольника
double area() {
```

Таблица 1. Отличия abstract class от interface

Abstract Class	Interface
Абстрактные методы нужно указывать с помощью ключевого слова abstract	Все методы являются абстрактными
В силу отсутствия множественного наследования, класс может наследовать только один абстрактный класс, но может наследовать много интерфейсов, и все это одновременно	Может наследовать много интерфейсов, но не может наследовать класс
Абстрактные методы можно объявлять с идентификаторами доступа (public, protected, private). При реализации в классе-потомке методы должны иметь такой же модификатор или менее ограниченный	Все методы публичные
Может содержать: методы с реализацией, поля, члены данных, константы.	Может содержать константы
В наследуемом от класса классе константа может быть переопределена	В наследуемом от класса классе константа НЕ может быть переопределена

```
System.out.println("В области четырехугольника.");
return diml * dim2;
}
}
class Triangle extends Figure {
Triangle(double a, double b) {
super(a, b);
}
// переопределение метода area для четырехугольника double area() {
System.out.println ("В области треугольника.");
return diml * dim2 12;
}
}
class AbstractAreas {
public static void main(String args[]) {
// Figure f = new Figure(10, 10); // теперь недопустимо
Rectangle r = new Rectangle (9, 5);
Triangle t = new Triangle (10, 8);
Figure figref; // этот оператор допустим, никакой объект не создается
figref = r; System.out.println("Площадь равна " + figref. area ()) ;
figref = t;
System.out.println("Площадь равна " + figref.area());
}
}
```

Как видно из комментария, внутри метода main (), объявление объектов типа Figure более недопустимо, поскольку теперь этот класс является абстрактным. И все подклассы класса Figure должны переопределять метод area (). Чтобы убедиться в этом, попытайтесь создать подкласс, который не переопределяет метод area (). Это приведет к ошибке времени компиляции.

Хотя создание объекта типа Figure недопустимо, можно создать ссылочную переменную типа Figure. Переменная figref объявлена как ссылка на Figure; т.е. ее можно использовать для ссылки на объект любого класса, производного от Figure. Разрешение переопределенных методов во время выполнения осуществляется путем ссылки на суперкласс (табл. 1).

Методы класса имеют шанс быть объявленными абстрактными. Иными словами, реализации этих методов в этом классе нет. Абстрактные методы пишут модификатором abstract.

Если в классе имеется хоть один из абстрактных методов, его можно назвать абстрактным (в нем могут быть также и обычные методы). Создавать экземпляры абстрактного класса нельзя, потому что его можно использовать только как базовый класс для других классов. У потомка этого класса две возможности: реализация всех абстрактных методов базового класса (здесь возможно создание экземпляров для такого класса-потомка) или реализация только части абстрактных методов базового класса (здесь он тоже будет абстрактным классом, а единственный шанс его применения — производство классов-потомков от него).

Таким образом, абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

Литература

1. *Ирэ П.* Объектно-ориентированное программирование с использованием C++ / Пер. с англ. Киев, 2005.
2. *Фейсон Т.* Объектно-ориентированное программирование на Borland C++ 4.5 / Пер. с англ. М., 2007.