Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Разработка кроссплатформенных приложений»

Инструкция
по выполнению лабораторной работы
«Разработка, проектирование, отладка и испытание классов и объектов в программе»

Минск 2025 г.

Лабораторная работа № 7

Тема работы: «Разработка, проектирование, отладка и испытание классов и объектов в программе»

1 Цель работы

Сформировать умения разрабатывать классы, создавать объекты и использовать их в программах.

2 Задание

Номер варианта соответствует вашему номеру по списку, если номер по списку больше последнего варианта, то необходимо начать подсчет варианта с начала (например, учащийся с номером по списку 13 должен выполнить 1 вариант).

По вариантам (создать класс, в них предусмотреть различные члены классов и методы для работы):

1. Базовый класс — учащийся. Производные — школьник и студент. Создать класе Конференция, который может содержать оба вида учащихся. Предусмотреть метод подсчета участников конференции отдельно по школьникам и по студентам (использовать оператор instanceof).

2. Базовый класс - работник, Производные — работник на почасовой оплате и на окладе. Создать класс Предприятие, который может содержать оба вида работников. Предусмотреть метод подсчета работников отдельно на почасовой оплате и на окладе (использовать оператор instanceof).

3. Базовый класс — компьютер. Производные — ноутбук и смартфон. Создать класс Ремонт Сервис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно ремонтируемых ноутбуков и смартфонов (использовать оператор instanceof).

4. Базовый класс — печатные издания. Производные — книги и журналы. Создать класс Книжный Магазин, который момет содержать оба видз объектов. Предусмотреть метод подсчета отдельно книг и журналов (использовать оператор instanceof).

5. Базовый класс — помещения. Производные - квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор instanceof).

6. Базовый класс — файл. Производные — звуковой файл и видео-файл. Создать класс Каталог, который может содержать оба вида объектов.

Предусмотреть метод подсчета отдельно звуковых и видео-файлов (использовать оператор instanceof).

7. Базовый класс — летательный аппарат. Производные — самолет и вертолет. Создать класс Авизкомпания, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно самолетов и вертолетов (использовать оператор instanceof).

8. Базовый класс — соревнование. Производные - командные соревнования и личные, Создать класс Чемпионат, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно командных соревнований и личных (использовать оператор instanceof).

9. Базовый класс — мебель. Производные — диван и шкаф. Создать класе Комната, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно диванов и шкафов (использовать оператор instanceof).

10, Базовый класс — оружие. Производные — огнестрельное и холодное. Создать класс Оружейная Палата, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно огнестрельного и холодного оружия (использовать оператор instanceof).

11. Базовый класс — ортехника. Производные - принтер и сканер. Создать класс Офис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно принтеров и сканеров (использовать оператор instanceof).

12. Базовый клас - СМИ. Производные - телеканал и газета. Создать класс Холдинг, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно телеканалов и газет (использовать оператор instanceof).

3 Оснащение работы

Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4 Основные теоретические сведения

Основы классов

Мы пользовались классами с самого начала этого руководства. Однако до сих пор применялась только наиболее примитивная форма класса. Классы, созданные в предшествующих главах, служили только в качестве контейнеров метода main (), который мы использовали для ознакомления с основами синтаксиса Java. Как вы вскоре убедитесь, классы предоставляют значительно больше возможностей, чем те, которые были представлены до сих пор.

Вероятно, наиболее важное свойство класса то, что он определяет новый тип данных. После того как он определен, этот новый тип можно применять для создания объектов данного типа. Таким образом, класс — это шаблон объекта, а объект — это экземпляр класса. Поскольку объект является экземпляром класса, термины объект и экземпляр часто используются попеременно.

Общая форма класса

При определении класса объявляют его конкретную форму и сущность. Это выполняется путем указания данных, которые он содержит, и кода, воздействующего на эти данные. Хотя очень простые классы могут содержать только код или только данные, большинство классов, применяемых в реальных программах, содержит оба эти компонента. Как будет показано в дальнейшем, код класса определяет интерфейс к его данным.

Для объявления класса служит ключевое слово class. Использованные до сих пор классы в действительности представляли собой очень ограниченные примеры полной формы. Классы могут быть (и обычно являются) значительно более сложными. Упрощенная общая форма определения класса имеет следующий вид:

```
class имя_класса {
тип переменная_экземпляра1;
тип переменная_экземпляра2;
// ...
тип переменная_экземпляраN;
тип имя_метода 1 (список_параметров) {
// тело метода
}
типимя_метода2 (список_параметров) {
// тело метода
}
// ...
тип имя__методаN (список_параметров) {
// тело метода
}
}
```

Данные, или переменные, определенные внутри класса, называются переменными экземпляра. Код содержится внутри методов. Определенные внутри класса методы и переменные вместе называют членами класса. В большинстве классов действия с переменными экземпляров и доступ к ним выполняются через методы, определенные в этом классе. Таким образом, в общем случае именно методы определяют способ использования данных класса.

Определенные внутри класса переменные называют переменными экземпляра, поскольку каждый экземпляр класса (т.е. каждый объект класса) содержит собственные копии этих переменных. Таким образом, данные одного объекта отделены и отличаются от данных другого объекта. Вскоре мы

вернемся к рассмотрению этой концепции, но она слишком важна, чтобы можно было обойтись без хотя бы предварительного ознакомления с нею.

Все методы имеют ту же общую форму, что и метод main (), который мы использовали до сих пор. Однако большинство методов не будут указаны как static или public. Обратите внимание, что общая форма класса не содержит определения метода main (). Классы Java могут и не содержать этот метод. Его обязательно указывать только в тех случаях, когда данный класс служит начальной точкой программы. Более того, аплеты вообще не требуют использования метода main ().

На заметку! *Программисты на C++ обратят внимание, что объявление класса и реализация методов хранятся в одном месте, а не определены отдельно. Иногда эта особенность приводит к созданию очень больших файлов .java, поскольку любой класс должен быть полностью определен в одном файле исходного кода. Такая архитектура была принята для Java умышленно, поскольку разработчики посчитали, что хранение определения, объявления и реализации в одном файле упрощает сопровождение кода в течение длительного периода его эксплуатации.*

Простой класс

Изучение классов начнем с простого примера. Ниже приведен код класса Box (Параллелепипед), который определяет три переменных экземпляра: width (ширина), height (высота) и depth (глубина). В настоящий момент Box не содержит никаких методов (но вскоре мы добавим в него метод).

```
class Box {
double width;
double height;
double depth;
}
```

Как уже было сказано, класс определяет новый тип данных. В данном случае новый тип данных назван Box. Это имя будет использоваться для объявления объектов типа Box. Важно помнить, что объявление class создает только шаблон, но не действительный объект. Таким образом, приведенный код не приводит к появлению никаких объектов типа Box.

Чтобы действительно создать объект Box, нужно использовать оператор, подобный следующему:

```
Box mybox = new Box();
// создание объекта mybox типа Box
```

После выполнения этого оператора mybox станет экземпляром класса Box. То есть он обретет "физическое" существование. Но пока можете не задумываться о нюансах этого оператора.

Повторим еще раз: при каждом создании экземпляра класса мы создаем объект, который содержит собственную копию каждой переменной экземпляра, определенной классом. Таким образом, каждый объект Box будет содержать собственные копии переменных экземпляра width, height и depth. Для получения доступа к этим переменным применяется операция точки (.). Эта

операция связывает имя объекта с именем переменной экземпляра. Например, чтобы присвоить переменной width объекта mybox значение 100, нужно было бы использовать следующий оператор:

```
mybox.width = 100;
```

Этот оператор указывает компилятору, что копии переменной width, хранящейся внутри объекта mybox, нужно присвоить значение, равное 100. В общем случае операцию точки используют для доступа как к переменным экземпляра, так и к методам внутри объекта.

Ниже приведена полная программа, в которой используется класс Box.

```
/* Программа, использующая класс Box.
Назовите этот файл BoxDemo.java
*/
class Box {
double width;
double height;
double depth;
}
// Этот класс объявляет объект типа Box.
class BoxDemo {
public static void main(String args[]) {
Box mybox = new Box () ;
double vol;
// присваивание значений переменным экземпляра mybox
mybox.width = 10;
mybox.height = 20;
mybox.depth = 15;
// вычисление объема параллелепипеда
vol = mybox.width * mybox.height * mybox.depth;
System.out.println("Объем равен " + vol);
}
}
```

Файлу этой программы нужно присвоить имя BoxDemo. j ava, поскольку метод main () определен в классе, названном BoxDemo, а не Box. Выполнив компиляцию этой программы, вы убедитесь в создании двух файлов . class: одного для Box и одного для BoxDemo. Компилятор Java автоматически помещает каждый класс в отдельный файл с расширением .class. В действительности классы Box и BoxDemo не обязательно должны быть объявлены в одном и том же исходном файле. Каждый класс можно было бы поместить в отдельный файл, названный соответственно Box. j ava и BoxDemo. j ava.

Чтобы запустить эту программу, нужно выполнить файл BoxDemo. class. В результате будет получен следующий вывод:

```
Объем равен 3000.0
```

Как было сказано ранее, каждый объект содержит собственные копии переменных экземпляра. Это означает, что при наличии двух объектов Box каждый из них будет содержать собственные копии переменных depth, width и height. Важно понимать, что изменения переменных экземпляра одного объекта не влияют на переменные экземпляра другого. Например, в следующем программе объявлены два объекта Box:

```
// Эта программа объявляет два объекта Box.
class Box {
double width;
double height;
double depth;
}
class BoxDemo2 {
public static void main(String args[]) {
Box myboxl = new Box() ;
Box mybox2 = new Box () ;
double vol;
// присваивание значений переменным экземпляра myboxl
myboxl.width = 10;
myboxl.height =20;
myboxl.depth = 15;
// присваивание других значений переменным экземпляра
mybox2
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9 ;
// вычисление объема первого параллелепипеда
vol = myboxl.width * myboxl.height * myboxl.depth;
System.out.println("Объем равен " + vol);
// вычисление объема второго параллелепипеда
vol = mybox2.width * mybox2.height * mybox2.depth;
System.out.println("Volume is " + vol);
}
}
```

Эта программа генерирует следующий вывод:

```
Объем равен 3000.0 Объем равен 162.0
```

Как видите, данные объекта myboxl полностью изолированы от данных, содержащихся в объекте mybox2.

Объявление объектов

Как мы уже отмечали, при создании класса вы создаете новый тип данных. Этот тип можно использовать для объявления объектов данного типа. Однако создание объектов класса — двухступенчатый процесс. Вначале необходимо объявить переменную типа класса. Эта переменная не определяет объект. Она представляет собой всего лишь переменную, которая может

ссылаться на объект. Затем потребуется получить действительную, физическую копию объекта и присвоить ее этой переменной. Это можно выполнить с помощью операции new. Эта операция динамически (т.е. во время выполнения) распределяет память под объект и возвращает ссылку на него. В общих чертах эта ссылка представляет собой адрес объекта в памяти, распределенной операцией new. Затем эта ссылка сохраняется в переменной. Таким образом, в Java все объекты классов должны распределяться динамически. Рассмотрим эту процедуру более подробно.

В приведенном ранее примере программы строка, подобная следующей, используется для объявления объекта типа Box:

```
Box mybox = new Box() ;
```

Этот оператор объединяет только что описанные шаги. Чтобы каждый из шагов был более очевидным, его можно было переписать следующим образом:

```
Box mybox; // объявление ссылки на объект
mybox = new Box();
// распределение памяти для объекта Box
```

Первая строка объявляет mybox в качестве ссылки на объект типа Box. После выполнения этой строки переменная mybox содержит значение null, свидетельствующее о том, что она еще не указывает на реальный объект. Любая попытка использования mybox на этом этапе приведет к возникновению ошибки времени компиляции. Следующая строка распределяет память под реальный объект и присваивает переменной mybox ссылку на этот объект. После выполнения второй строки переменную mybox можно использовать, как если бы она была объектом Box. Но в действительности переменная mybox просто содержит адрес памяти реального объекта Box. Эффект выполнения этих двух строк кода показан на рисунке. 7.1.

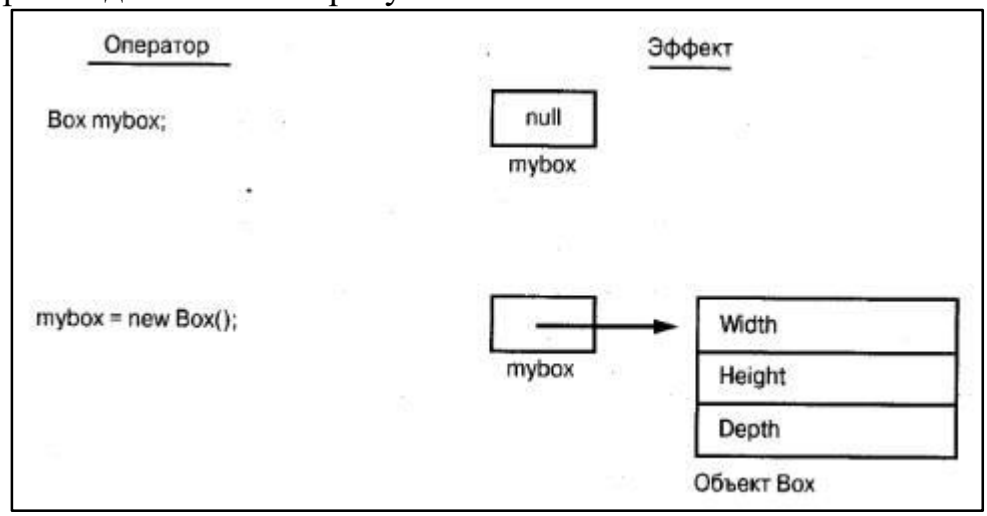

Рис. 7.1. Объявление объекта типа Box

Более подробное рассмотрение операции new

Как было сказано, операция new динамически распределяет память для объекта. Общая форма этой операции имеет следующий вид:

```
переменная_класса = new имя_класса();
```

Здесь переменная_класса — переменная создаваемого типа класса. Имя_класса — это имя класса, конкретизация которого выполняется. Имя класса, за которым следуют круглые скобки, указывает конструктор данного класса. Конструктор определяет действия, выполняемые при создании объекта класса. Конструкторы — важная часть всех классов, и они обладают множеством важных атрибутов. Большинство классов, используемых в реальных программах, явно определяют свои конструкторы внутри своего определения класса. Однако если никакой явный конструктор не указан, Java автоматически предоставит конструктор, используемый по умолчанию. Это же происходит в случае объекта Box. Пока мы будем пользоваться конструктором, заданным по умолчанию. Вскоре читатели научатся определять собственные конструкторы.

У читателей может возникнуть вопрос, почему не требуется использовать операцию new для таких элементов, как целые числа или символы. Это обусловлено тем, что элементарные типы Java реализованы в не виде объектов, а в виде "обычных" переменных. Это сделано для повышения эффективности. Как вы убедитесь, объекты обладают множеством свойств и атрибутов, которые требуют, чтобы Java-программа обрабатывала их иначе, чем элементарные типы. Отсутствие накладных расходов, связанных с обработкой объектов, при обработке элементарных типов позволяет эффективнее реализовать элементарные типы. Несколько позже мы приведем объектные версии элементарных типов, которые могут пригодиться в ситуациях, когда требуются полноценные объекты этих типов.

Важно понимать, что операция new распределяет память для объекта во время выполнения. Преимущество этого подхода состоит в том, что программа может создавать ровно столько объектов, сколько требуется во время ее выполнения. Однако поскольку объем памяти ограничен, возможна ситуация, когда операция new не сможет выделить память для объекта из-за ее нехватки. В этом случае возникнет исключение времени выполнения. В примерах программ, приведенных в этой книге, можно не беспокоиться по поводу недостатка объема памяти, но в реальных программах эту возможность придется учитывать.

Еще раз рассмотрим различие между классом и объектом. Класс создает новый тип данных, который можно использовать для создания объектов. То есть класс создает логический каркас, определяющий взаимосвязь между его членами. При объявлении объекта класса мы создаем экземпляр этого класса. Таким образом, класс — это логическая конструкция. А объект обладает физической сущностью. (То есть объект занимает область в памяти.) Важно помнить об этом различии.

Присваивание переменных объектных ссылок

При выполнении присваивания переменные объектных ссылок действуют иначе, чем можно было бы представить. Например, какие действия, по вашему мнению, выполняет следующий фрагмент?

```
Box b1 = new Box ();
```

```
Box b2 = b1;
```

Можно подумать, что переменной b2 присваивается ссылка на копию объекта, на которую ссылается переменная b1. То есть может показаться, что b1 и b2 ссылаются на отдельные и различные объекты. Однако это не так. После выполнения этого фрагмента кода обе переменные b1 и b2 будут ссылаться на один и тот же объект. Присваивание b1 переменной b2 не привело к распределению какой-то памяти или копированию какой-либо части исходного объекта. Эта операция присваивания приводит лишь к тому, что переменная b2 ссылается на тот же объект, что и переменная b1. Таким образом, любые изменения, выполненные в объекте через переменную b2, окажут влияние на объект, на который ссылается переменная b1, поскольку это — один и тот же объект.

Эта ситуация отражена на рисунке 7.2.

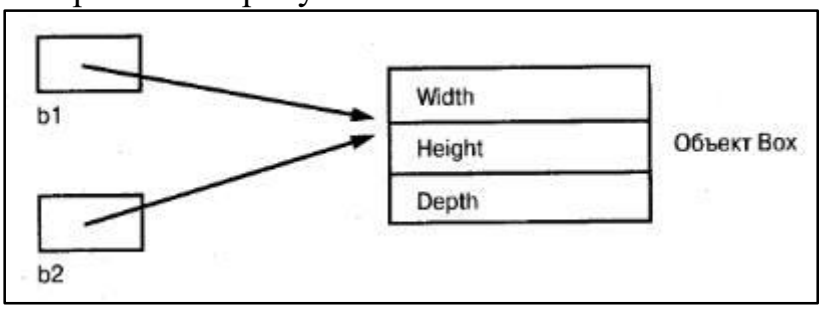

Рис. 2. Использование переменных объектных ссылок

Хотя и b1 и b2 ссылаются на один и тот же объект, эти переменные не связаны между собой никаким другим образом. Например, следующая операция присваивания значения переменной b1 просто разорвет связь переменной Ы с исходным объектом, не оказывая влияния на сам объект или на переменную b2:

```
Box b1 = new Box () ;
Box b2 = b1;
// ...
b1 = null;
```

В этом примере значение b1 установлено равным null, но переменная b2 по-прежнему указывает на исходный объект.

Помните! *Присваивание ссылочной переменной одного объекта ссылочной переменной другого объекта не ведет к созданию копии объекта, а лишь создает копию ссылки.*

Знакомство с методами

Как было сказано в начале этой главы, обычно классы состоят из двух элементов: переменных экземпляра и методов. Поскольку язык Java предоставляет им столь большие возможности и гибкость, тема методов очень обширна. Фактически многие последующие главы посвящены методам. Однако чтобы можно было приступить к добавлению методов к своим классам, необходимо ознакомиться с рядом их основных характеристик.

Общая форма объявления метода выглядит следующим образом:

```
тип имя (список_параметров) {
```

```
// тело метода
}
```

Здесь тип указывает тип данных, возвращаемых методом. Он может быть любым допустимым типом, в том числе типом класса, созданным программистом. Если метод не возвращает значение, типом его возвращаемого значения должен быть void. Имя служит для указания имени метода. Оно может быть любым допустимым идентификатором, кроме тех, которые уже используются другими элементами в текущей области определения. Список_параметров — последовательность пар "тип-идентификатор", разделенных запятыми. По сути, параметры — это переменные, которые принимают значения аргументов, переданных методу во время его вызова. Если метод не имеет параметров, список параметров будет пустым. Методы, тип возвращаемого значения которых отличается от void, возвращают значение вызывающей процедуре с помощью следующей формы оператора return: `return значение;` Здесь значение — это возвращаемое значение. В нескольких последующих разделах мы рассмотрим создание различных типов методов, в том числе как принимающие параметры, так и возвращающие значения.

Добавление метода к классу Box

Хотя было бы весьма удобно создать класс, который содержит только данные, в реальных программах подобное встречается редко. В большинстве случаев для осуществления доступа к переменным экземпляра, определенным классом, придется использовать методы. Фактически методы определяют интерфейсы большинства классов. Это позволяет программисту, который реализует класс, скрывать конкретную схему внутренних структур данных за более понятными абстракциями метода. Кроме определения методов, которые обеспечивают доступ к данным, можно определять также методы, используемые внутренне самим классом.

Теперь приступим к добавлению метода в класс Box. Просматривая предшествующие программы, легко прийти к выводу, что класс Box мог бы лучше справиться с вычислением объема параллелепипеда, чем класс BoxDemo. В конце концов, поскольку объем параллелепипеда зависит от его размеров, вполне логично, чтобы его вычисление выполнялось в классе Box. Для этого в класс Box нужно добавить метод, как показано в следующем примере:

```
// Эта программа содержит метод внутри класса box.
class Box {
double width;
double height;
double depth;
// отображение объема параллелепипеда
void volume () {
System.out.print("Объем равен ") ;
System.out.println(width * height * depth);
}
```

```
}
class BoxDemo3 {
public static void main(String args[]) {
Box myboxl = new Box() ;
Box mybox2 = new Box();
// присваивание значений переменным экземпляра myboxl
myboxl.width = 10;
myboxl.height = 20;
myboxl.depth = 15;
/* присваивание других значений переменным
экземпляра mybox2 */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
// отображение объема первого параллелепипеда
myboxl .volume () ;
// отображение объема второго параллелепипеда
mybox2.volume();
}
}
```

Эта программа генерирует следующий вывод, совпадающий с выводом предыдущей версии:

```
Объем равен 3000.0 Объем равен 162.0
```

Внимательно взгляните на следующие две строки кода:

myboxl.volume() ;

mybox2.volume();

В первой строке присутствует обращение к методу volume (), определенному в myboxl. То есть она вызывает метод volume () по отношению к объекту myboxl, для чего было использовано имя объекта, за которым следует символ операции точки. Таким образом, обращение к myboxl .volume () отображает объем параллелепипеда, определенного объектом myboxl, а обращение к mybox2 .volume () — объем параллелепипеда, определенного объектом mybox2. При каждом вызове метода volume () он отображает объем указанного параллелепипеда.

Соображения, приведенные в следующих абзацах, облегчат понимание концепции вызова метода. При вызове метода myboxl .volume О система времени выполнения Java передает управление коду, определенному внутри метода volume (). По завершении выполнения всех операторов внутри метода управление возвращается вызывающей программе и ее выполнение продолжается со строки, которая следует за вызовом метода. В самом общем смысле можно сказать, что метод — способ реализации подпрограмм в Java.

В методе volume () следует обратить внимание на один очень важный нюанс: ссылка на переменные экземпляра width, height и depth выполняется непосредственно, без указания перед ними имени объекта или операции точки.

Когда метод использует переменную экземпляра, которая определена его классом, он выполняет это непосредственно, без указания явной ссылки на объект и без применения операции точки. Это становится понятным, если немного подумать. Метод всегда вызывается по отношению к какому-то объекту его класса. Как только этот вызов выполнен, объект известен. Таким образом, внутри метода вторичное указание объекта совершенно излишне. Это означает, что width, height и depth неявно ссылаются на копии этих переменных, хранящиеся в объекте, который вызывает метод volume ().

Подведем краткие итоги. Когда обращение к переменной экземпляра выполняется кодом, не являющимся частью класса, в котором определена переменная экземпляра, оно должно выполняться посредством объекта с применением операции точки. Однако когда это обращение осуществляется кодом, который является частью того же класса, где определена переменная экземпляра, ссылка на переменную может выполняться непосредственно. Эти же правила применимы и к методам.

Добавление метода, принимающего параметры

Хотя некоторые методы не нуждаются в параметрах, большинство требует их передачи. Параметры позволяют обобщать метод. То есть метод с параметрами может работать с различными данными и/или применяться в ряде несколько различных ситуаций. В качестве иллюстрации рассмотрим очень простой пример. Ниже показан метод, который возвращает квадрат числа 10.

```
int square () {
return 10 * 10;
}
```

Хотя этот метод действительно возвращает 102, его применение очень ограничено. Однако если его изменить так, чтобы он принимал параметр, как показано в следующем примере, метод square () может стать значительно более полезным.

```
int square(int i) {
return i * i;
}
```

Теперь метод square () будет возвращать квадрат любого значения, с которым он вызван. То есть теперь метод square () является методом общего назначения, который может вычислять квадрат любого целочисленного значения, а не только числа 10.

Приведем примеры:

```
int x, y;
x = square(5);
// x равно 25 x = square (9);
//x равно 81 У = 2;
x = square (y) ;
// x равно 4
```

В первом обращении к методу square () значение 5 будет передано параметру i. Во втором обращении параметр i примет значение, равное 9.

Третий вызов метода передает значение переменной y, которое в этом примере составляет 2. Как видно из этих примеров, метод square () способен возвращать квадрат любых переданных ему данных.

Важно различать два термина: параметр и аргумент. Параметр — это переменная, определенная методом, которая принимает значение при вызове метода. Например, в методе square () параметром является i. Аргумент — это значение, передаваемое методу при его вызове. Например, aquare (100) передает 100 в качестве аргумента. Внутри метода square () параметр i получает это значение.

Методом с параметрами можно воспользоваться для усовершенствования класса Box. В предшествующих примерах размеры каждого параллелепипеда нужно было устанавливать отдельно, используя последовательность операторов вроде следующей:

```
myboxl.width = 10;
myboxl.height =20;
myboxl.depth = 15;
```

Хотя этот код работает, он не очень удобен по двум причинам. Во-первых, он громоздок и чреват ошибками. Например, вполне можно забыть определить один из размеров. Во-вторых, в правильно спроектированных Java-программах доступ к переменным экземпляра должен осуществляться только через методы, определенные их классом. В будущем поведение метода можно изменить, но нельзя изменить поведение раскрытой переменной экземпляра.

Поэтому более рациональный подход установки размеров параллелепипеда — создание метода, который принимает размеры параллелепипеда в виде своих параметров и соответствующим образом устанавливает значение каждой переменной экземпляра. Эта концепция реализована в приведенной ниже программе.

```
// Эта программа использует метод с параметрами.
class Box {
double width;
double height;
double depth;
// вычисление и возвращение объема
double volume() {
return width * height * depth;
// установка размеров параллелепипеда
void setDim(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
}
class BoxDemo5 {
public static void main(String args[]) {
```

```
Box mybox1 = new Box() ;
Box mybox2 = new Box();
double vol;
// инициализация каждого экземпляра Box
mybox1.setDim(10, 20, 15);
mybox2.setDim(3, 6, 9) ;
// получение объема первого параллелепипеда
vol = mybox1.volume () ;
System.out.println("Объем равен " + vol) ;v // получение
объема второго параллелепипеда
vol = mybox2.volume();
System.out.println("Объем равен " + vol);
}
}
```

Как видите, метод setDim () использован для установки размеров каждого параллелепипеда. Например, при выполнении такого оператора:

```
mybox1.setDim(10, 20, 15);
```

значение 10 копируется в параметр w, 20 — в h и 15 — в d. Затем внутри метода setDim () значения w, h и d присваиваются соответственно переменным width, height и depth.

Многим читателям представленные в предшествующих разделах концепции будут знакомы. Однако если вы еще не знакомы с такими понятиями, как вызовы методов, аргументы и параметры, можете немного поэкспериментировать с ними, прежде чем продолжить изучение материала, изложенного в последующих разделах. Концепции вызова метода, параметров и возвращаемых значений являются основополагающими в программировании на языке Java.

Конструкторы

Инициализация всех переменных класса при каждом создании его экземпляра может оказаться утомительным процессом. Даже при добавлении функций, предназначенных для увеличения удобства работы, таких как setDim (), было бы проще и удобнее, если бы все действия по установке переменных выполнялись при первом создании объекта. Поскольку необходимость инициализации возникает столь часто, Java позволяет объектам выполнять собственную инициализацию при их создании. Эта автоматическая инициализация осуществляется с помощью конструктора.

Конструктор инициализирует объект непосредственно во время создания. Его имя совпадает с именем класса, в котором он находится, а синтаксис аналогичен синтаксису метода. Как только он определен, конструктор автоматически вызывается непосредственно после создания объекта, перед завершением выполнения операции new. Конструкторы выглядят несколько непривычно, поскольку не имеют ни возвращаемого типа, ни даже типа void. Это обусловлено тем, что неявно заданный возвращаемый тип конструктора класса — тип самого класса. Именно конструктор инициализирует внутреннее

состояние объекта так, чтобы код, создающий экземпляр, с самого начала содержал полностью инициализированный, пригодный к использованию объект.

Пример класса Box можно изменить, чтобы значения размеров параллелепипеда присваивались при конструировании объекта. Для этого потребуется заменить метод setDim () конструктором. Вначале определим простой конструктор, который просто устанавливает одинаковые значения размеров для всех параллелепипедов. Эта версия программы имеет вид:

```
/* В этом примере класс Box использует конструктор
для инициализации размеров параллелепипеда. */
class Box {
double width;
double height;
double depth;
// Это конструктор класса Box.
Box () {
System.out.println("Конструирование объекта Box");
width =10;
height =10;
depth =10;
}
// вычисление и возвращение объема
double volume () {
return width * height * depth;
}
}
class BoxDemo6 {
public static void main(String args[]) {
// объявление, распределение и инициализация объектов Box
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// получение объема первого параллелепипеда
vol = mybox1.volume () ;
System.out.println("Объем равен " + vol);
// получение объема второго параллелепипеда
vol = mybox2.volume ();
System.out.println("Объем равен " + vol);
}
}
```

Эта программа генерирует следующий вывод:

```
Конструирование объекта Box
Конструирование объекта Box
Объем равен 1000.0
```

```
Объем равен 1000.0
```

Как видите, и myboxl, и mybox2 были инициализированы конструктором Box () при их создании. Поскольку конструктор присваивает всем параллелепипедам одинаковые размеры 10x10x10, и myboxl, и mybox2 будут иметь одинаковый объем. Оператор println () внутри конструктора Box () служит исключительно иллюстративным целям.

Большинство конструкторов не выводят никакой информации, а лишь выполняют инициализацию объекта. П

режде чем продолжить, еще раз рассмотрим операцию new. Как вы уже знаете, при распределении памяти для объекта используют следующую общую форму:

```
переменная_класса = new имя_класса() ;
```

Теперь вам должно быть ясно, почему после имени класса требуются круглые скобки. В действительности этот оператор вызывает конструктор класса. Таким образом, в строке:

```
Box myboxl = new Box() ;
```

операция new Box () вызывает конструктор Box (). Если конструктор класса не определен явно, Java создает для класса конструктор, который будет использоваться по умолчанию. Именно поэтому приведенная строка кода работала в предшествующих версиях класса Box, в которых конструктор не был определен. Конструктор, используемый по умолчанию, инициализирует все переменные экземпляра нулевыми значениями. Зачастую конструктора, используемого по умолчанию, вполне достаточно для простых классов, чего обычно нельзя сказать о более сложных. Как только конструктор определен, конструктор, заданный по умолчанию, больше не используется.

Ключевое слово this

Иногда будет требоваться, чтобы метод ссылался на вызвавший его объект. Чтобы это было возможно, в Java определено ключевое слово this. Оно может использоваться внутри любого метода для ссылки на текущий объект. То есть this всегда служит ссылкой на объект, для которого был вызван метод. Ключевое слово this можно использовать везде, где допускается ссылка на объект типа текущего класса.

Для пояснения рассмотрим следующую версию конструктора Box ():

```
// Избыточное применение ключевого слова this.
Box(double w, double h, double d) {
this.width = w;
this.height = h;
this.depth = d;
}
```

Эта версия конструктора Box () действует точно так же, как предыдущая. Применение ключевого слова this избыточно, но совершенно правильно. Внутри метода Box () ключевое слово this всегда будет ссылаться на вызывающий объект. Хотя в данном случае это и излишне, в других случаях,

один из которых рассмотрен в следующем разделе, ключевое слово this весьма полезно.

Сокрытие переменной экземпляра

Как вы знаете, в Java не допускается объявление двух локальных переменных с одним и тем же именем в одной и той же или во включающих одна другую областях определения. Интересно отметить, что могут существовать локальные переменные, в том числе формальные параметры методов, которые перекрываются с именами переменных экземпляра класса. Однако когда имя локальной переменной совпадает с именем переменной экземпляра, локальная переменная скрывает переменную экземпляра. Именно поэтому внутри класса Box переменные width, height и depth не были использованы в качестве имен параметров конструктора Box (). В противном случае переменная width ссылалась бы на формальный параметр, скрывая переменную экземпляра width. Хотя обычно проще использовать различные имена, существует и другой способ выхода из подобной ситуации. Поскольку ключевое слово this позволяет ссылаться непосредственно на объект, его можно применять для разрешения любых конфликтов пространства имен, которые могут возникать между переменными экземпляра и локальными переменными. Например, ниже показана еще одна версия метода Box (), в которой имена width, height и depth использованы в качестве имен параметров, а ключевое слово this служит для обращения к переменным экземпляра по этим же именам.

```
// Этот код служит для разрешения конфликтов пространства имен.
Box(double width, double height, double depth) {
this.width = width;
this.height = height;
this.depth = depth;
}
```

Небольшое предостережение: иногда подобное применение ключевого слова this может приводить к недоразумениям, и некоторые программисты стараются не применять имена локальных переменных и параметров, скрывающие переменные экземпляров. Конечно, множество программистов придерживаются противоположного мнения и считают целесообразным в целях облегчения понимания программ использовать одни и те же имена, а для предотвращения скрытия переменных экземпляров применяют ключевое слово this. Выбор того или иного подхода — дело личного вкуса.

5 Порядок выполнения работы

1. Выделить ключевые моменты задачи;
2. Построить алгоритм и теоритическую объектную модель решения задачи;
3. Запрограммировать полученный алгоритм и объектную модель;
4. Провести тестирование полученной программы.

6 Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

_____

_____

7. Контрольные вопросы и задания

1. Что представляет собой класс?
2. Что может содержать класс?
3. Назовите назначение конструкторов.
4. Как создать объект класс?
5. Как вызвать метод через объект класса?
6. Как скрыть поля класс за его пределами?

8. Рекомендуемая литература

1. **Урванов, Ф. В.** Java. Состояние языка и его перспективы. / Ф.В. Урванов. - Санкт-Петербург : БХВ-Петербург, 2023. - 368 с.

2. **Копец Дэвид.** Классические задачи Computer Science на языке Java. - Санкт-Петербург : Питер, 2022. - 288 с

3. **Васильев А. Н.** Java. Объектно-ориентированное программирование: Учебное пособие / А.Н. Васильев. - Санкт-Петербург : Питер, 2021. - 400 с.

4. **Эванс Бенджамин**. Java для опытных разработчиков. 2-е изд. — (Серия «Библиотека программиста»). - Санкт-Петербург : Питер, 2024. - 736 с.

5. **Лой Марк.** Программируем на Java. 5-е межд. изд. . - Санкт-Петербург : Питер, 2023. - 544 с.

6. **Гетц Брайан**. Java Concurrency на практике. - Санкт-Петербург : Питер, 2021. - 464 с.