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, отладка и испытание алгоритмов и программ с перегруженными методами класса»

Минск 2025 г.

Лабораторная работа № 8

Тема работы: «Разработка, отладка и испытание алгоритмов и программ с перегруженными методами класса»

1 Цель работы

Обучить разработке и перегрузке методов с параметрами различного статуса, использованию встроенных методов.

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 22 должен выполнить 2 вариант в первом задании и 3 вариант во втором задании).

Необходимо в каждом классе из лабораторной работы № 7 реализовать по 4 перегруженных метода (необходимых по логике).

3 Оснащение работы

Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4 Основные теоретические сведения

Перегрузка конструкторов

Наряду с перегрузкой обычных методов можно также выполнять перегрузку методов конструкторов. Фактически перегруженные конструкторы станут нормой, а не исключением, для большинства классов, которые вам придется создавать для реальных программ. Чтобы это утверждение было понятным, вернемся к классу Box, разработанному в предыдущей главе. Его последняя версия имеет следующий вид:

```
class Box {
double width;
double height;
double depth;
// Это конструктор класса Box.
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// вычисление и возврат значения
double volume() {
return width * height * depth;
}
}
```

Как видите, конструктор Box () требует передачи трех параметров. Это означает, что все объявления объектов Box должны передавать конструктору Box () три аргумента. Например, следующий оператор недопустим:

```
Box ob = new Box () ;
```

Поскольку конструктор Box () требует передачи трех аргументов, его вызов без аргументов — ошибка. Эта ситуация порождает три важных вопроса. Что если нужно было просто определить параллелепипед и его начальные размеры не имели значения (или не были известны)? Или, нужно иметь возможность инициализировать куб, указывая только один размер, который должен использоваться для всех трех измерений? При текущем определении класса Box все эти дополнительные возможности недоступны.

К счастью, решение подобных проблем достаточно просто: достаточно перегрузить конструктор Box, чтобы он учитывал только что описанные ситуации. Ниже приведена программа, которая содержит усовершенствованную версию класса Box, выполняющую эту задачу.

```
/* В этом примере класс Box определяет три конструктора
для инициализации размеров параллелепипеда различными
способами. */
class Box {
double width;
double height;
double depth;
// конструктор, используемый при указании всех измерений
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// конструктор, используемый, когда ни один из размеров
не указан
Box() {
width = -1; // значение -1 используется для указания
height = -1; // неинициализированного
depth = -1; // параллелепипеда
}
// конструктор, используемый при создании куба
Box(double len) {
width = height = depth = len;
}
// вычисление и возврат объема
double volume () {
return width * height * depth;
class OverloadCons {
public static void main(String args[]) {
```

```
// создание параллелепипедов с применением различных
конструкторов
Box mybox1 = new Box(10, 20, 15) ;
Box mybox2 = new Box();
Box mycube = new Box(7) ;
double vol;
// получение объема первого параллелепипеда
vol = mybox1.volume();
System.out.println("Объем mybox1 равен " + vol);
// получение объема второго параллелепипеда
vol = mybox2.volume () ;
System.out.println("Объем mybox2 равен " + vol);
// получение объема куба
vol = mycube.volume ();
System.out.println("Объем mycube равен " + vol);
}
}
```

Эта программа создает следующий вывод:

```
Объем mybox1 равен 3000.0
Объем mybox2 равен -1.0
Объем mycube равен 343.0
```

Как видите, соответствующий перегруженный конструктор вызывается в зависимости от параметров, указанных при выполнении операции new.

Использование объектов в качестве параметров

До сих пор в качестве параметров методов мы использовали только простые типы. Однако передача методам объектов — и вполне допустима, и достаточно распространена. Например, рассмотрим следующую короткую программу:

```
// Методам можно передавать объекты. class Test { int a,
b;
Test(int i, int j) {
a = i;
b = j;
}
// возврат значения true, если параметр о равен
вызывающему объекту
boolean equals(Test o) {
if(o.a == a && o.b == b) return true;
else return false;
class PassOb {
public static void main(String args[]) {
Test obi = new Test(100, 22);
Test ob2 = new Test(100, 22);
```

```
Test ob3 = new Test(-1, -lb-System, out .println ("obi ==
ob2: " + obi.equals(ob2) ) ;
System.out.println("obi == ob3 : " + obi.equals(ob3));
}
}
```

Эта программа создает следующий вывод:

```
obi == ob2: true
obi == oЬ3: false
```

Как видите, метод equals () внутри метода Test проверяет равенство двух объектов и возвращает результат этой проверки. То есть он сравнивает вызывающий объект с тем, который был ему передан. Если они содержат одинаковые значения, метод возвращает значение true. В противном случае он возвращает значение false. Обратите внимание, что параметр о в методе equals () указывает Test в качестве типа. Хотя Test — тип класса, созданный программой, он используется совершенно так же, как встроенные типы Java.

Одно из наиболее часто встречающихся применений объектов-параметров — в конструкторах. Часто приходится создавать новый объект так, чтобы вначале он не отличался от какого-то существующего объекта. Для этого потребуется определить конструктор, который в качестве параметра принимает объект его класса. Например, следующая версия класса Box позволяет выполнять инициализацию одного объекта другим:

```
//В этой версии Box допускает инициализацию одного
объекта другим.
class Box {
double width;
double height;
double depth;
// Обратите внимание на этот конструктор. Он использует
объект типа Bo:
Box(Box ob) { // передача объекта конструктору
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// конструктор, используемый при указании всех измерений
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// конструктор, используемый, если ни одно из изменений
не указано
Box () {
width = -1; // значение -1 используется для указания
```

```
height = -1; //не инициализированного
depth = -1; // параллелепипеда
}
// конструктор, используемый при создании куба
Box(double len) {
width = height = depth = len;
}
// вычисление и возврат объема
double volume () {
return width * height * depth;
}
}
class 0verloadCons2 {
public static void main(String args[]) {
// создание параллелепипедов с применением различных
конструкторов
Box myboxl = new Box(10, 20, 15);
Box mybox2 = new Box() ;
Box mycube = new Box (7) ;
Box myclone = new Box(myboxl) ; // создание копии объекта
myboxl double vol;
// получение объема первого параллелепипеда
vol = myboxl.volume () ;
System.out.println("Объем myboxl равен " + vol);
// получение объема второго параллелепипеда
vol = mybox2.volume();
System.out.println("Объем mybox2 равен " + vol);
// получение объема куба
vol = mycube.volume O ;
System.out.println("Объем куба равен " + vol);
// получение объема клона
vol = myclone.volume O;
System.out.println("Объем клона равен " + vol);
}
}
```

Как вы убедитесь, приступив к созданию собственных классов, чтобы объекты можно было конструировать удобным и эффективным образом, нужно располагать множеством форм конструкторов.

Возврат объектов

Метод может возвращать любой тип данных, в том числе созданные типы классов. Например, в следующей программе метод icrByTen () возвращает объект, в котором значение переменной a на 10 больше значения этой переменной в вызывающем объекте.

// Возвращение объекта.

```
class Test {
int a;
Test(int i) { a = i;
Test incrByTen () {
Test temp = new Test(a+10);
return temp;
}
}
class RetOb {
public static void main(String args[]) {
Test ob1 = new Test (2) ;
Test ob2;
ob2 = ob1.incrByTen();
System.out.println ("ob1.a: " + ob1.a);
System.out.println("ob2.a: " + ob2.a);
ob2 = ob2.incrByTen();
System.out.println("ob2.a после второго увеличения значения: " + ob2.a);
}
}
```

Эта программа генерирует следующий вывод:

```
ob1.a: 2
ob2.a: 12
ob2.a после второго увеличения значения: 22
```

Как видите, при каждом вызове метода incrByTen () программа создает новый объект и возвращает ссылку на него вызывающей процедуре.

Приведенная программа иллюстрирует еще один важный момент: поскольку все объекты распределяются динамически с помощью операции new, программисту не нужно беспокоиться о том, чтобы объект не вышел за пределы области определения, т.к. выполнение метода, в котором он был создан, прекращается. Объект будет существовать до тех пор, пока где-либо в программе будет существовать ссылка на него. При отсутствии какой-либо ссылки на него объект будет уничтожен во время следующей уборки мусора.

Что такое static

В некоторых случаях желательно определить член класса, который будет использоваться независимо от любого объекта этого класса. Обычно обращение к члену класса должно выполняться только в сочетании с объектом его класса. Однако можно создать член класса, который может использоваться самостоятельно, без ссылки на конкретный экземпляр. Чтобы создать такой член, в начало его объявления нужно поместить ключевое слово static. Когда член класса объявлен как static (статический), он доступен до создания каких-либо объектов его класса и без ссылки на какой-либо объект. Статическими могут быть объявлены как методы, так и переменные. Наиболее распространенный пример статического члена — метод main (). Этот метод объявляют как static, поскольку он должен быть объявлен до создания любых объектов.

Переменные экземпляров, объявленные как static, по существу являются глобальными переменными. При объявлении объектов их класса программа не создает никаких копий переменной static. Вместо этого все экземпляры класса совместно используют одну и ту же статическую переменную.

На методы, объявленные как static, накладывается ряд ограничений.

- ✓ Они могут вызывать только другие статические методы.
- ✓ Они должны осуществлять доступ только к статическим переменным.
- ✓ Они ни коим образом не могут ссылаться на члены типа this или super. (Ключевое слово super связано с наследованием и описывается в следующей главе.)

Если для инициализации переменных типа static нужно выполнить вычисления, можно объявить статический блок, который будет выполняться только один раз при первой загрузке класса. В следующем примере показан класс, который содержит статический метод, несколько статических переменных и статический блок инициализации:

```
// Демонстрация статических переменных, методов и блоков.
class UseStatic {
static int a = 3;
static int b;
static void meth(int x) {
System.out.println ("x = " + x) ;
System.out.println ("a = " + a);
System.out.println("b = " + b) ;
}
static {
System.out.println("Статический блок инициализирован.");
b = a * 4;
}
public static void main(String args[]) {
meth(42);
}
}
```

Сразу после загрузки класса UseStatic программа выполняет все операторы static. Вначале значение a устанавливается равным 3, затем программа выполняет блок static, который выводит сообщение, а затем инициализирует переменную b значением a*4, или 12. Затем программа вызывает метод main (), который обращается к методу meth (), передавая параметру x значение 42. Три оператора println () ссылаются на две статических переменные a и b на локальную переменную x.

Вывод этой программы имеет такой вид:

```
Статический блок инициализирован, x = 42 a = 3 b = 12
```

За пределами класса, в котором они определены, статические методы и переменные могут использоваться независимо от какого-либо объекта. Для

этого достаточно указать имя их класса, за которым должна следовать операция точки. Например, если метод типа static нужно вызвать извне его класса, это можно выполнить, используя следующую общую форму:

```
имя_класса.метод( )
```

Здесь имя_класса — имя класса, в котором объявлен метод тип static. Как видите, этот формат аналогичен применяемому для вызова нестатических методов через переменные объектных ссылок. Статическая переменная доступна аналогичным образом — посредством операции точки, следующей за именем класса. Так в Java реализованы управляемые версии глобальных методов и переменных.

Приведем пример. Внутри метода main () обращение к статическому методу callme () и статической переменной b осуществляется посредством имени их класса StaticDemo.

```
class StaticDemo {
static int a = 42;
static int b = 99;
static void callme () {
System.out.println("a = " + a);
}
}
class StaticByName {
public static void main(String args[]) {
StaticDemo.callme () ;
System.out.println("b = " + StaticDemo.b);
}
}
```

Вывод этой программы выглядит следующим образом:

```
a = 42
b = 99
```

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

Как можно перегрузить конструктор?

2. Можно ли в качестве параметра отправлять объект?

3. Можно ли вернуть из метода объект?

4. Назначение ключевого слова static.

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.