Практическая работа № 4. Перечисления и их использование в Джава программах

Цель данной практической работы – познакомиться с новым ссылочным типом данных перечислением, научиться разрабатывать перечисления и использовать их в своих программах.

Теоретические сведения

Перечисления это один из объектных типов в Джава. Они являются безопасными типами, поскольку переменная тип перечисление может принимать значение только константу из перечисления. Рассмотрим пример объявления перечисления в языке Джва:

```
public enum Level {
    HIGH,
    MEDIUM,
    LOW
}
```

Обратите внимание на enum - ключевое слово перед именем перечисления, которое используется вместо class или interface. Ключевое слово enum сигнализирует компилятору Java, что это определение типа является перечислением. Вы можете ссылаться на константы в приведенном выше перечислении следующим образом:

```
Level level = Level.HIGH;
```

В этом случае переменная level принимает значение HIGH.

Обратите внимание, что level переменная имеет тип, Level который является типом перечисления Java, определенным в приведенном выше примере. Level.

Переменная может принимать одно значение из констант перечисления Level, то есть в качестве значения может принимать значения HIGH, MEDIUM, или LOW.

Использование перечислений в операторах ветвления if () else

Поскольку все перечисления в Джава являются константами, то вам часто придется сравнивать переменную, указывающую на константу перечисления, с возможными константами в типе перечисления. Вот пример использования перечисления в операторе if () else:

```
Level level = ...
//Присваиваем некоторое значение из перечисления Level
if( level == Level.HIGH) {
```

```
} else if( level == Level.MEDIUM) {
} else if( level == Level.LOW) {
}
```

В этом коде переменную level сравнивается с каждой из возможных констант перечисления в перечислении Level. Если оно соответствует одному из значений в перечислении, то проверка этого значения в первом операторе if приведет к повышению производительности, так как в среднем выполняется меньше сравнений.

Использование перечислений в операторах множественного выбора switch

Если ваши типы перечисления Java содержат много констант и вам нужно проверять переменную на соответствие нескольким значениям из перечисления, то воспользуйтесь оператором идеей switch.

Вы можете использовать перечисления в операторах switch следующим образом:

```
Level level = ...
// присвоить ему некоторую константу Level

switch (level) {
    case HIGH: ...;break;
    case MEDIUM: ...; break;
    case LOW: ...; break;
}
```

Замените многоточие ... в коде выше на то значение, которое вам нужно для выполнения кода, если переменная level истинно, то выполнится соответствующая ветка switch, т.е. в выражении переменная совпадет со значением константы соответствует заданному значению константы присвоенному level. Код, который соответствует определенной ветви switch может быть просто блок выражений или например может выполняться вызов метода и т.д.

Итерация перечислений

Вы можете получить массив всех возможных значений типа перечисления Джава, вызвав его статический метод values(). Все типы перечислений автоматически получают статический метод values() с помощью компилятора Джава. Вот пример итерации всех значений перечисления:

```
for (Level level : Level.values()) {
    System.out.println(level);
```

```
}
```

Запуск кода выше распечатает все значения перечисления.

Вот результат выполнения этого кода:

```
HIGH
MEDIUM
LOW
```

Обратите внимание, как печатаются имена самих констант. Это одно из отличий работы с перечислениями, перечисления Java отличаются от static final констант.

Использование метода toString () с перечислениями

Класс перечисления автоматически получает метод toString()при компиляции. Метод toString() возвращает строковое значение для данного экземпляра перечисления.

Пример:

```
Строка levelText = Level.HIGH.toString ();
```

Значением переменной levelText после выполнения вышеуказанного оператора будет текст HIGH.

Печать перечислений

Вы можете вывести на значение перечисления обычным способом, с помощью метода println(), например таким образом:

```
System.out.println (Level.HIGH);
```

Затем будет вызван метод toString(), поэтому значение, которое будет распечатано, будет именем экземпляра перечисления - текст. Другими словами, в приведенном выше примере будет напечатан текст HIGH.

Использование метода valueOf () с перечислениями

Класс перечисление автоматически получает статический метод valueOf() в классе при компиляции. Метод valueOf() может быть использован для получения экземпляра класса перечислимого для заданного значения String. Вот пример:

```
Level level = Level.valueOf ("BHIGH");
```

Переменная типа Level будет указывать на Level.HIGH после выполнения этой строки.

Поля перечислений

Вы можете добавлять поля данных в перечисление Джава, также как в классы. Таким образом, каждое константа перечисления получает эти поля.

Значения полей должны быть переданы конструктору перечисления при определении констант. Пример приведен на листинге 4.1.

Лстинг 4.1 – Пример перечисления Level

```
public enum Level {
    HIGH(3),  //вызов конструктора со значением 3
    MEDIUM(2),  // вызов конструктора со значением 2
    LOW(1)   // вызов конструктора со значением 1
/* нужна точка с запятой, если после констант следуют
поля и методы*.
    ;
 //определение полей перечисления
    private final int levelCode;
    private Level(int levelCode) {
        this.levelCode = levelCode;
    }
}
```

Обратите внимание, как у перечисления в приведенном выше примере есть конструктор, который принимает параметр типа int. Конструктор перечисления устанавливает поле int. Когда определены постоянные значения перечисления, int значение передается конструктору перечисления. Конструктор для перечисления должен быть объявлен с модификатором private. Вы не можете использовать конструкторы с модификаторами public или protected для перечислений в Джава. Если вы не укажете модификатор доступа в конструкторе перечисления, то он будет объявлен неявным образом с модификатором private.

Поля перечислений

Вы также можете добавлять методы в перечисление Джава. Пример приведен на листинге 4.2

Лстинг 4.2 – Пример перечисления Level с полями и методами

```
public enum Level {
  HIGH (3), // вызывает конструктор со значением 3
  MEDIUM (2),// вызывает конструктор со значением 2
  LOW (1) // вызывает конструктор со значением 1
    ; //здесь нужна точка с запятой
//поле для перечисления
  private final int levelCode;
//конструктор
  private Level (int levelCode) {
```

```
    this.levelCode = levelCode;
    }
// геттер для поля levelCode
     public int getLevelCode () {
         return this.levelCode;
     }
}
```

Вы вызываете метод перечисления Java через ссылку на одно из постоянных значений. Вот пример вызова метода перечисления Java:

```
Level level = Level.HIGH;
System.out.println (level.getLevelCode ());
```

Этот код распечатает значение, 3 которое является значением levelCode для поля HIGH константы перечисления

Кроме конструкторов, геттеров и сеттеров вы можете добавлять свои собственные методы в классы перечисления, которые будут производить вычисления на основе значений полей константы перечисления. Если ваши поля не объявлены как final, вы даже можете изменить значения полей (хотя это может быть не очень хорошая идея, учитывая, что перечисления должны быть константами).

Задания на практическую работу № 4

Задание 1. Времена года

Создать перечисление, содержащее названия времен года.

1) Создать переменную, содержащую ваше любимое время года и распечатать всю информацию о нем.
2) Создать метод, который принимает на вход переменную созданного вами enum типа. Если значение равно Лето, выводим на консоль “Я люблю лето” и так далее. Используем оператор switch.
3) Перечисление должно содержать переменную, содержащую среднюю температуру в каждом времени года.
4) Добавить конструктор, принимающий на вход среднюю температуру.
5) Создать метод getDescription, возвращающий строку “Холодное время года”. Переопределить метод getDescription - для константы Лето метод должен возвращать “Теплое время года”.
6) В цикле распечатать все времена года, среднюю температуру и описание времени года

Задание 2. Ателье

Создать перечисление, содержащее размеры одежды (XXS, XS, S, M, L). Перечисление содержит метод getDescription, возвращающий строку “Взрослый размер”. Переопределить метод getDescription - для константы XXS метод должен возвращать строку “Детский размер”. Также перечисление должно содержать числовое значение euroSize(32, 34, 36, 38, 40), соответствующее каждому размеру. Создать конструктор, принимающий на вход euroSize.

1) Создать интерфейсы MenClothing (мужская одежда) с методом dressMan() (одеть мужчину) и WomenClothing (женская одежда) с методом dressWomen() (одеть женщину).
2) Создать абстрактный класс Clothes (одежда), содержащий в качестве переменных класса - размер одежды, стоимость, цвет.
3) Создать классы наследники класса Clothes – класс TShirt (футболка) (реализует интерфейсы MenClothing и WomenClothing), класс Pants (штаны) (реализует интерфейсы MenClothing и WomenClothing), класс Skirt (реализует интерфейсы WomenClothing), класс Tie (галстук) (реализует интерфейсы MenClothing).
4) Создать массив, содержащий все типы одежды. Создать класс Atelier (Ателье), содержащий методы dressWomen, dressMan, на вход которых будет поступать массив, содержащий все типы одежды (подумайте какой тип будет у массива). Переопределите метод dressWomen() для вывода на консоль всей информации о женской одежде. То же самое сделайте для метода dressMan().

Задание 3. Интернет-магазин

Создать мини приложение - интернет-магазин. Должны быть реализованы следующие возможности:

1) Аутентификация пользователя. Пользователь вводит логин и пароль с клавиатуры.
2) Просмотр списка каталогов товаров.
3) Просмотр списка товаров определенного каталога.
4) Выбор товара в корзину.
5) Покупка товаров, находящихся в корзине.

Для выполнения заданий необходимо создать перечисление согласно заданию, можете добавить свои операции или изменить что-то по своему усмотрению.

Задание 4

Создать класс, описывающий сущность компьютер (Computer). Для описания составных частей компьютера использовать отдельные классы (Processor, Memory, Monitor). Описать необходимые свойства и методы для каждого класса. Для названий марок компьютера используйте перечисления (enum)

Практическая работа № 4.1. Наследование в Джава. Абстрактные классы.

Цель: познакомиться на практике с реализацией принципа ООП Наследование в Джава и освоить на практике работу с наследованием от абстрактных классов.

Теоретические сведения об абстрактных классах и наследовании

Наследование один из основных принципов разработки объектно-ориентированных программ. В терминологии Джава базовый класс, от которого производится наследование называется суперкласс. Производные классы, называются подклассы наследуют все компоненты родителей (поля и методы) кроме конструкторов и статических компонентов. Обратится к методам родительского класса можно, используя служебное слово super. Основное преимущество наследования — это возможность повторного использования кода. Наследование поддерживает концепцию «возможности повторного использования», т.е. когда мы хотим создать новый класс и уже существует класс, который включает в себя часть кода, который нам нужен, мы можем получить наш новый класс из уже существующего класса. Таким образом, мы повторно используем поля и методы уже существующего класса. Для организации наследования в Джава, используется ключевое слово – extends, что в переводе на русский означает расширяет. Таким образом создавая производный класс с помощью наследования, мы расширяем родительский класс как новыми свойствами – поля данных, так добавляем к нему новое поведение – методы класса. Пример на листинге 4.1.1

Листинг 4.1.1 – Пример наследования

```
// базовый класс велосипед
class Bicycle {
public int gear; // поле
public int speed; // поле

// конструктор класса
public Bicycle(int gear, int speed){
this.gear = gear;
this.speed = speed;
}
//метод класса
public void applyBrake(int decrement){
```

```
speed -= decrement;
}
//метод класса
public void speedUp(int increment){
speed += increment;
}
// метод toString()чтобы печатать объекты Bicycle
public String toString(){
return ("No of gears are " + gear + "\n"
+ "speed of bicycle is " + speed);
}
} // end of class

// производный класс горный велосипед
class MountainBike extends Bicycle {
public int seatHeight; //новое поле произв. класса
//конструктор производного класса
public MountainBike(int gear, int speed,
int startHeight){
// здесь вызов конструктора класса родителя
super(gear, speed);
seatHeight = startHeight;
}
// новый метод производного класса
public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }=
// переопределенный метод toString()класса Bicycle
@Override public String toString(){
return (super.toString() + "\nseat height is "
+ seatHeight);
  }
}
// класс тестер Main
public class Main {
public static void main(String args[]){
// создаем объект родительского класса
```

```
Bicycle bl = new Bicycle(5,200);
System.out.println(bl.toString());

// создаем объект дочернего класса
MountainBike mb = new MountainBike(3, 100, 25);
System.out.println(mb.toString());
    }
}
```

Абстрактные классы

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы при определении помечаются ключевым словом abstract.

Абстрактный метод внутри абстрактного класса не имеет тела, только прототип. Он состоит только из объявления и не имеет тела:

```
abstract void yourMethod();
```

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придётся предоставить определения для всех абстрактных методов базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также абстрактные конструкторы и абстрактные статические методы. Любой подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо сам быть объявлен абстрактным. Пример приведен на листинге 4.1.2

Листинг 4.1.2 – Пример абстрактного класса Swim

```
public abstract class Swim {
// абстрактный метод
abstract void swim()
// абстрактный класс может содержать иобычный метод
void run() {
System.out.println(“Куда идешь?”);
```

```
}
}
//создаем производный класс Swimmer
class Swimmer extends Swim {
….
}
```

Задания на практическую работу № 4.1

1. Необходимо реализовать простейший класс Shape (Фигура). Добавьте метод класса getType() (тип фигуры, возвращает строку тип String название фигуры). С помощью наследования создайте дочерние классы Circle, Rectangle и Square. (из предыдущей практической работы). Также реализуйте во всех классах методы getArea()(возвращает площадь фигуры), getPerimeter() взвращает периметр фигуры). Переопределите в дочерних класс методы класса родителя toString(), getArea(), getPerimeter() и getType(). Создать класс-тестер для вывода информации об объекте и продемонстирировать вызов методов использую родительскую ссылку. Объяснить работу программы.

2. Создайте класс Phone, который содержит переменные number, model и weight.

1)Создайте три экземпляра этого класса. 2) Выведите на консоль значения их переменных. 3) Добавить в класс Phone методы: receiveCall, имеет один параметр – имя звонящего. 4)Выводит на консоль сообщение “Звонит {name}”. 5)Метод getNumber – возвращает номер телефона. 6) Вызвать эти методы для каждого из объектов. 7) Добавить конструктор в класс Phone, который принимает на вход три параметра для инициализации переменных класса - number, model и weight. 8)Добавить конструктор, который принимает на вход два параметра для инициализации переменных класса - number, model. 9)Добавить конструктор без параметров. 10)Вызвать из конструктора с тремя параметрами конструктор с двумя. 11)Добавьте перегруженный метод. receiveCall, который принимает два параметра - имя звонящего и номер телефона звонящего. 12)Вызвать этот метод. 13)Создать метод sendMessage с аргументами переменной длины. Данный метод принимает на вход номера телефонов, которым будет отправлено сообщение. 14)Метод выводит на консоль номера этих телефонов.

3. Создать класс Person, который содержит: а) поля fullName, age. б) методы move() и talk(), в которых просто вывести на консоль сообщение -"Такой-то Person говорит". в) Добавьте два конструктора - Person() и Person(fullName, age). Создайте два объекта этого класса. Один объект инициализируется конструктором Person(), другой - Person(fullName, age).

4. Создать класс Матрица. Класс должен иметь следующие поля: 1) двумерный массив вещественных чисел; 2) количество строк и столбцов в матрице. Класс должен иметь следующие методы: 1) сложение с другой матрицей; 2) умножение на число; 3) вывод на печать; 4) умножение матриц - по желанию.

5. Класс «Читатели библиотеки». Определить класс Reader, хранящий такую информацию о пользователе библиотеки: ФИО, номер читательского билета, факультет, дата рождения, телефон. Методы takeBook(), returnBook(). Разработать программу, в которой создается массив объектов данного класса. Перегрузить методы takeBook(), returnBook(): - takeBook, который будет принимать количество взятых книг. Выводит на консоль сообщение "Петров В. В. взял 3 книги". - takeBook, который будет принимать переменное количество названий книг. Выводит на консоль сообщение "Петров В. В. взял книги: Приключения, Словарь, Энциклопедия". - takeBook, который будет принимать переменное количество объектов класса Book (создать новый класс, содержащий имя и автора книги). Выводит на консоль сообщение “Петров В. В. взял книги: Приключения, Словарь, Энциклопедия”. Аналогичным образом перегрузить метод returnBook(). Выводит на консоль сообщение “Петров В. В. вернул книги: Приключения, Словарь, Энциклопедия”. Или “Петров В. В. вернул 3 книги”.

6. Создайте пример наследования, реализуйте класс Employer и класс Manager. Manager отличается от Employer наличием дополнительных выплат от продаж а) Класс Employer содержит переменные: String firstName, lastName и поле income для заработной платы. Класс Manager также имеет поле double averageSum содержащую среднюю суммы дополнительных выплат за продажи. б) Создать переменную типа Employer, которая ссылается на объект типа Manager. в) Создать метод getIncome() для класса Employer, который возвращает заработную плату. Если средняя количество отработанных дней, то сумма дохода умножается на 12. Переопределить этот метод в классе Manager и добавить к доходу сумму с продаж. г) Создать массив типа Employer содержащий объекты класса Employer и Manager. Вызвать метод getIncome() для каждого элемента массива.

7. Создать суперкласс Учащийся и подклассы Школьник и Студент. Создать массив объектов суперкласса и заполнить этот массив объектами. Показать отдельно студентов и школьников.

Задания на абстрактные классы

8. Перепишите суперкласс Shape из задания 1, сделайте его

абстрактным и наследуйте подклассы, так как это представлено на UML диаграмме на рис. 4.1.1 Circle, Rectangle и Square.

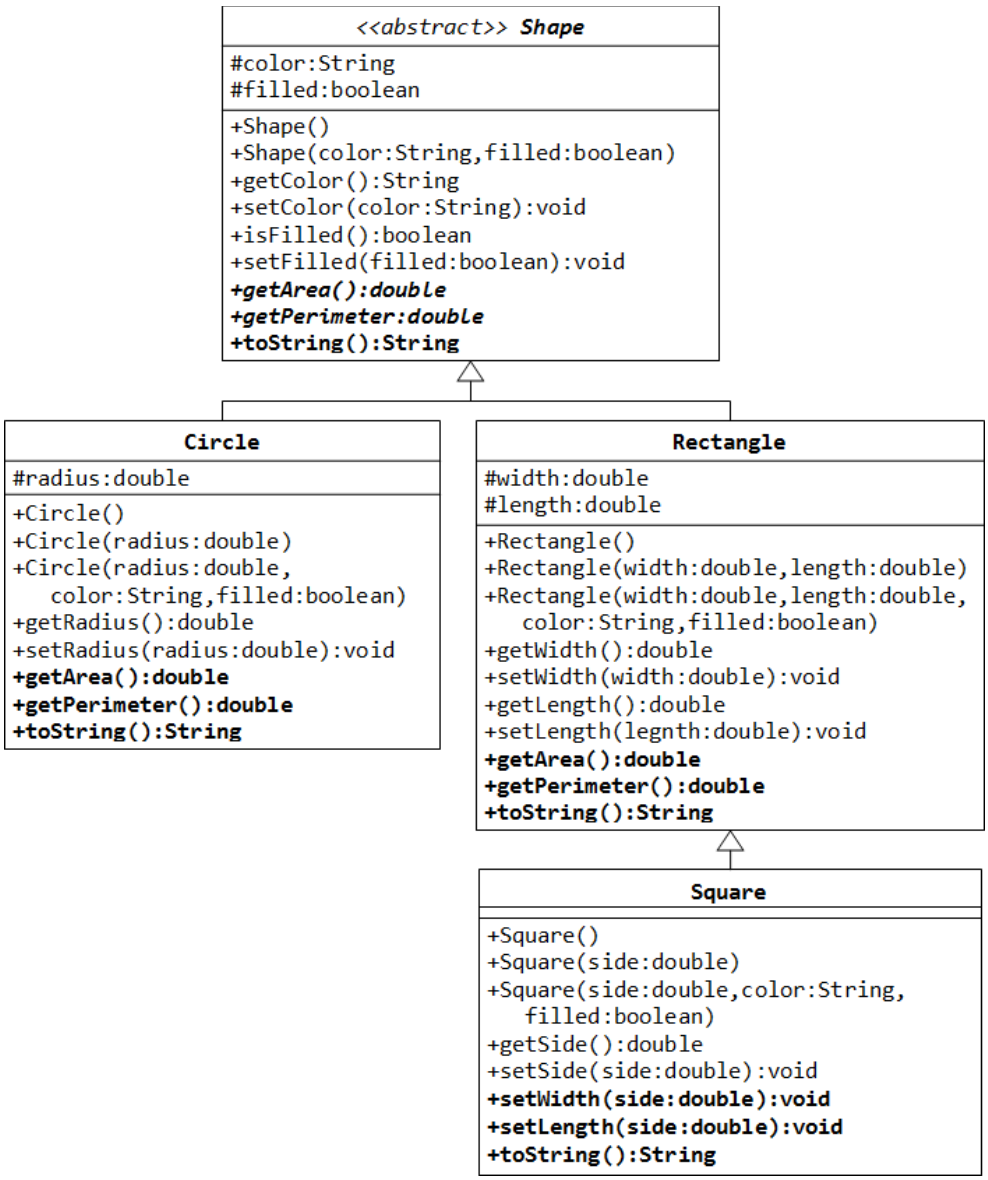

Рисунок 4.1.1. Диаграмма суперкласса Shape.

Замечания. В этом задании, класс Shape определяется как абстрактный класс, которыйсодержит:

- Два поля или переменные класса, объявлены с модификатором ***protected*** color (тип String) и filled (тип boolean). Такие защищенные переменные могут быть доступны в подклассах и классах в одном пакете. Они обозначаются со знаком “#” на диаграмме классов в нотации языка UML.
- Методы геттеры и сеттеры для всех переменных экземпляра класса,и метод toString().
- Два абстрактных метода getArea() и getPerimeter() выделены курсивом в диаграмме класса).

В подклассах Circle (круг) и Rectangle (прямоугольник) должны

переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString() .

9. Создать абстрактный класс, описывающий сущность мебель. С помощью наследования реализовать различные виды мебели. Также создать класс FurnitureShop, моделирующий магазин мебели. Протестировать работу классов.

10. Создать абстрактный класс, описывающий Транспортное средство и подклассы Автомобиль, Самолет, Поезд, Корабль. Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.

Упражнение 1

Внимание. Внимание в заданиях есть код, который потенциально содержит ошибки, вам нужно объяснить ошибки и представить рабочую версию кода.

Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. *Объясните полученные ошибки*, если таковые имеются.

Листинг 4.1.3 – Пример для выполнения

```
Shape s1 = new Circle(5.5, “RED”, false); // Upcast Circle
to ShapeSystem.out.println(s1);//which version?
    System.out.println(s1.getArea());//  which  version?
System.out.println(s1.getPerimeter());//which    version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle  c1  =  (Circle)s1;//  Downcast  back  to  Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());
Shape s2 = new Shape();
Shape s3 = new Rectangle(1.0, 2.0, "RED", false);// Upcast
System.out.println(s3);
```

```
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3; //downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());
Shape s4 = new Square(6.6);//Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());
/*обратите внимание, что выполняем downcast Shape s4 к
Rectangle,      который      является      суперклассом
Square(родителем), вместо Square*//
Rectangle r2 = (Rectangle)s4;
System.out.println(r2); System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());
// Downcast Rectangle r2 к Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```