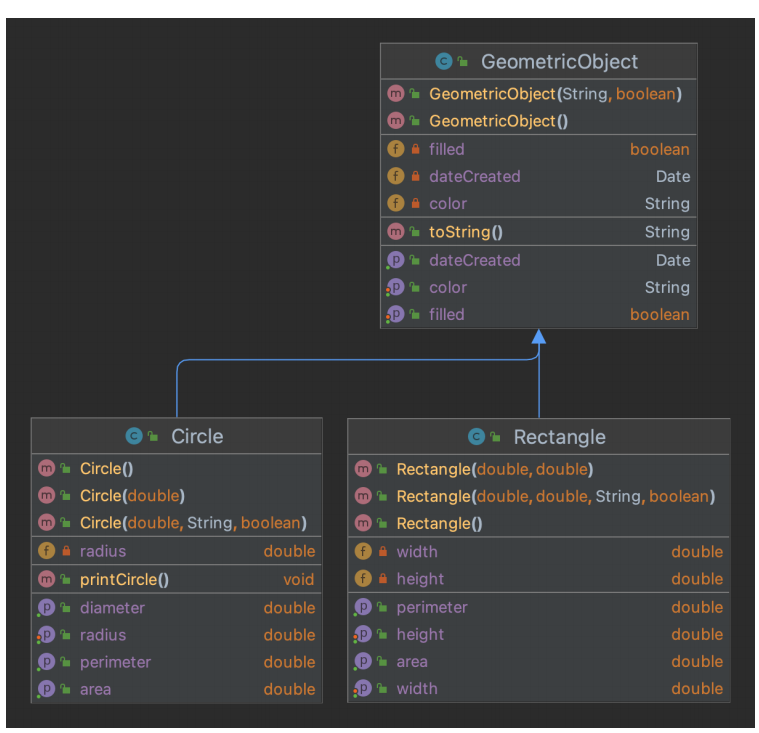
**Практическая работа №4**

**ООП в JAVA. Исключения**

**Теоретическое введение**

Рассмотрим геометрические фигуры. Нам требуется создать классы для моделирования геометрических фигур, таких как круги и прямоугольники. У геометрических фигур есть много общих свойств и вариантов поведения. Они могут быть определенного цвета, закрашенными или незакрашенными. Таким образом, общий класс GeometricObject можно использовать для моделирования всех геометрических фигур. Этот класс содержит свойства цвет color и заливка filled, а также соответствующие им getter- и setter-методы. Этот класс также содержит свойство dateCreated и методы getDateCreated() и toString(). Метод toString() возвращает строковое представление объекта.

Поскольку круг является конкретным типом геометрической фигуры, он имеет общие свойства и методы с другими геометрическими фигурами. Таким образом, имеет смысл определить класс Circle, который будет порожден от класса GeometricObject. Аналогично, класс Rectangle также можно определить как конкретный тип класса GeometricObject.



**UML диаграмма отношения наследования между классами**

Класс Rectangle наследует все поля данных и методы класса GeometricObject. Кроме того, у него есть поля данных width и height и связанные с ними getter- и setter-методы. Он также содержит методы getArea() и getPerimeter() для возврата площади и периметра прямоугольника.

Класс Circle порождается от класса GeometricObject с помощью следующего синтаксиса:

// Circle это подкласс, а GeometricObject - суперкласс public class Circle extends GeometricObject{

...

}

Ключевое слово **extends** сообщает компилятору, что класс Circle порождается от класса GeometricObject, наследуя таким образом методы getColor(), setColor(), isFilled(), setFilled() и toString(). Перегруженный конструктор Circle реализуется путем вызова методов setColor() и setFilled() для присваивания свойств color и filled. Public-методы, определенные в суперклассе GeometricObject, наследуются подклассом Circle, поэтому их можно использовать в Circle.

**Небольшая теория по основам исключений**

Обработка исключений в Java — один из мощных механизмов обработки ошибок времени выполнения, позволяющий поддерживать нормальный поток работы приложения. В Java исключение — это событие, которое нарушает нормальный ход программы. Это объект, который выбрасывается во время выполнения.

Java предоставляет пять ключевых слов, которые используются для обработки исключений.

| **Слово** | **Описание** |
| --- | --- |
| **try** | Ключевое слово "try" используется для указания блока, в который мы должны поместить код исключения. Это означает, что мы не можем использовать блок try в одиночку. За блоком try должен следовать либо catch, либо finally. |
| **catch** | Блок "catch" используется для обработки исключения. Ему должен предшествовать блок try, что означает, что мы не можем использовать блок catch отдельно. За ним может следовать блок finally позже. |
| **finally** | Блок "finally" используется для выполнения необходимого кода программы. Он выполняется независимо от того, обработано исключение или нет. |
| **throw** | Ключевое слово "throw" используется для создания исключения. |
| **throws** | Ключевое слово "throws" используется для объявления исключений. Оно указывает, что в методе может возникнуть исключение. Оно не выбрасывает исключение. Оно всегда используется с сигнатурой метода. |

Рассмотрим пример обработки исключений Java, в котором мы используем оператор try-catch для обработки исключения.

public class JavaExceptionExample{

public static void main(String args[]){

try{

// код, который может вызвать исключение

int data=100/0;

}catch(ArithmeticException e){System.out.println(e);}

// остальной код программы

System.out.println("остальной код...");

}

}

Вывод:

Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

В приведенном выше примере 100/0 вызывает исключение ArithmeticException, которое обрабатывается блоком try-catch.

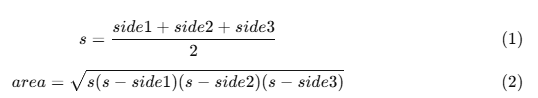
## Задание №1

Создайте класс Triangle для представления треугольников, который порождается от класса GeometricObject. Напишите клиент этих классов — программу, которая запрашивает у пользователя ввести три стороны треугольника, цвет и логическое значение для указания закрашен ли треугольник. Программа должна создавать объект типа Triangle с указанными сторонами и присваивать значения свойствам цвет color и заливка isFilled с помощью этих входных данных. Программа должна отображать площадь area, периметр perimeter, цвет, а также true или false для указания, закрашен треугольник или нет.

Класс Triangle должен содержать:

* Три поля данных типа double с именами side1, side2 и side3 и с заданными по умолчанию значениями, равными 1.0, для обозначения трех сторон треугольника.
* Безаргументный конструктор, который создает треугольник с заданными по умолчанию значениями.
* Конструктор, который создает треугольник с side1, side2 и side3.
* Getter-методы для всех трех полей данных.
* Метод с именем getArea(), который возвращает площадь этого треугольника.
* Метод с именем getPerimeter(), который возвращает периметр этого треугольника.
* Метод с именем toString(), который возвращает строковое описание треугольника.

Формулы для вычисления площади треугольника:



где s — полупериметр, а area — площадь треугольника.

Метод toString() должен быть реализован следующим образом:

java

return "Треугольник: сторона1 = " + side1 + " сторона2 = " +

side2 +

" сторона3 = " + side3;

## Задание №2

В треугольнике сумма длин любых двух сторон больше длины третьей стороны. Класс Triangle должен удовлетворять этому правилу. Создайте класс IllegalTriangleException и измените конструктор класса Triangle, чтобы выбросить объект типа IllegalTriangleException, если треугольник создан со сторонами, нарушающими это правило, следующим образом:

java

*/\*\* Создает треугольник с указанными сторонами \*/*

public Triangle(double side1, double side2, double side3)

throws IllegalTriangleException {

*// Тут надо написать реализацию исключения*

}

## Задание №3

Спроектируйте новый класс Triangle, который наследуется от абстрактного класса GeometricObject.

1. Напишите тестовую программу, которая запрашивает у пользователя ввод трёх сторон треугольника, цвета и логического значения для указания заливки.
2. Программа должна создать объект типа Triangle с этими сторонами и задать свойства color и filled, используя введенные пользователем данные.
3. Программа должна отображать площадь, периметр, цвет и значение true или false для указания заливки.
4. Реализуйте в классе GeometricObject интерфейс Comparable и определите статический метод max() в классе GeometricObject для поиска наибольшего из двух объектов типа GeometricObject.
5. Проверьте тестовой программой работу метода max() для поиска наибольшего из двух кругов и наибольшего из двух прямоугольников.
6. Вернитесь к классу с именем ComparableCircle, который наследуется от Circle и реализует Comparable. Напишите тестовую программу, чтобы найти наибольший из двух экземпляров класса ComparableCircle и наибольший между кругом и прямоугольником, используя метод compareTo().

## Задание №4

1. Создайте интерфейс с именем Colorable с помощью метода howToColor() типа void. Каждый класс раскрашиваемого объекта должен реализовывать интерфейс Colorable.
2. Спроектируйте класс Square, который наследуется от GeometricObject и реализует Colorable. Реализуйте метод howToColor() для отображения сообщения: Раскрасьте все четыре стороны.
3. Класс Square содержит поле данных side с getter- и setter-методами, а также конструктор для создания Square с указанной стороной. У класса Square есть скрытое поле данных типа double с именем side и getter- и setter-методами. У него есть безаргументный конструктор, который создает объект типа Square со стороной, равной 0, и еще один конструктор, который создает объект типа Square с указанной стороной.
4. Напишите тестовую программу, которая создает массив из пяти объектов типа GeometricObjects. Для каждого объекта в массиве отобразите его площадь и вызовите метод howToColor(), если его можно раскрасить.

## Теория

В Java API существует множество предопределенных классов исключений. Исключения – это объекты, а объекты определяются с помощью классов. Корневым классом для исключений является java.lang.Throwable. Классы исключений можно разделить на три основных типа: системные ошибки, исключения и ошибки во время выполнения. В Java можно определить пользовательский класс исключений, породив его от класса java.lang.Exception.

### Описание Throwable

### Описание Exception

### Спецификация Exception в языке Java

### Системные ошибки

Системные ошибки представлены в классе Error, их выбрасывает JVM. Класс Error описывает внутренние ошибки системы, хотя такие ошибки и происходят очень редко. Если все-таки они происходят, то мало что можно сделать, кроме уведомления пользователя и попытки завершить программу.

| **Класс** | **Возникновение исключения** |
| --- | --- |
| LinkageError | У одного класса есть некое отношение с другим классом, но последний класс несовместимо изменился после компиляции первого класса. |
| VirtualMachineError | JVM неисправна или исчерпала ресурсы, необходимые для продолжения работы. |

### Исключения

Исключения представлены в классе Exception, который описывает ошибки, вызванные самой программой и внешними обстоятельствами. Эти ошибки можно обнаружить и обработать с помощью программы.

| **Класс** | **Возникновение исключения** |
| --- | --- |
| ClassNotFoundException | Попытка использовать несуществующий класс. Это исключение может произойти, например, при попытке запуска несуществующего класса с помощью команды java. |
| IOException | Связаны с операциями ввода/вывода, такими как недопустимые входные данные, считывание после конца файла и открытие несуществующего файла. |

### Ошибки во время выполнения

Ошибки во время выполнения представлены в классе RuntimeException, который описывает ошибки программирования, такие как неправильное приведение типов, доступ к массиву вне его границ и числовые ошибки.

| **Класс** | **Возникновение исключения** |
| --- | --- |
| ArithmeticException | Деление целого числа на ноль. Арифметические операции над числами с плавающей точкой не приводят к исключениям. |
| NullPointerException | Попытка доступа к объекту с помощью нулевой переменной ссылочного типа. |
| IndexOutOfBoundsException | Индекс массива находится вне его границ. |
| IllegalArgumentException | Методу передается недопустимый или не соответствующий ему аргумент. |

### Пример использования исключений

В этом примере решения задачи показано объявление, выбрасывание и перехват исключений с помощью изменения метода setRadius() класса Circle. Метод setRadius() будет выбрасывать исключение, если радиус отрицательный.

RuntimeException, Error и их подклассы называются непроверяемыми (unchecked) исключениями. Все остальные исключения называются проверяемыми (checked) исключениями, то есть компилятор заставляет программиста проверять и обрабатывать их в блоке try-catch или объявлять в заголовке метода.

#### Класс TestCircleWithPrivateDataFields

java

public class TestCircleWithPrivateDataFields {

*/\*\* Метод main \*/*

public static void main(String[] args) {

*// Создать круг с радиусом, равным 5*

Circle myCircle = new Circle(5.0);

System.out.println("Площадь круга с радиусом "

+ myCircle.getRadius() + " равна " + myCircle.getArea());

*// Увеличить радиус созданного круга на 10%*

myCircle.setRadius(myCircle.getRadius() \* 1.1);

System.out.println("Площадь круга с радиусом "

+ myCircle.getRadius() + " равна " + myCircle.getArea());

System.out.println("Количество созданных объектов равно "

+ Circle.getNumberOfObjects());

}

}

#### Класс Circle

java

public class Circle {

*/\*\* Радиус круга \*/*

private double radius = 1;

*/\*\* Количество созданных объектов \*/*

private static int numberOfObjects = 0;

*/\*\* Создает круг с радиусом, равным 1 \*/*

public Circle() {

numberOfObjects++;

}

*/\*\* Создает круг с указанным радиусом \*/*

public Circle(double newRadius) {

radius = newRadius;

numberOfObjects++;

}

*/\*\* Возвращает радиус \*/*

public double getRadius() {

return radius;

}

*/\*\* Присваивает новый радиус \*/*

public void setRadius(double newRadius) {

radius = (newRadius >= 0) ? newRadius : 0;

}

*/\*\* Возвращает количество объектов \*/*

public static int getNumberOfObjects() {

return numberOfObjects;

}

*/\*\* Возвращает площадь круга \*/*

public double getArea() {

return radius \* radius \* Math.PI;

}

}

В программе TestCircleWithException определяется новый класс круга с именем CircleWithException, который совпадает с Circle из программы TestCircleWithPrivateDataFields, за исключением того, что метод setRadius(double newRadius) выбрасывает исключение IllegalArgumentException, если аргумент newRadius отрицателен.

#### Класс TestCircleWithException

java

public class TestCircleWithException {

public static void main(String[] args) {

try {

CircleWithException c1 = new CircleWithException(5);

CircleWithException c2 = new CircleWithException(-5);

CircleWithException c3 = new CircleWithException(0);

}

catch (IllegalArgumentException ex) {

System.out.println(ex);

}

System.out.println("Количество созданных объектов: " +

CircleWithException.getNumberOfObjects());

}

}

#### Класс CircleWithException

java

public class CircleWithException {

*/\*\* Радиус круга \*/*

private double radius;

*/\*\* Количество созданных объектов \*/*

private static int numberOfObjects = 0;

*/\*\* Создает круг с радиусом, равным 1 \*/*

public CircleWithException() {

this(1.0);

}

*/\*\* Создает круг с указанным радиусом \*/*

public CircleWithException(double newRadius) {

setRadius(newRadius);

numberOfObjects++;

}

*/\*\* Возвращает радиус \*/*

public double getRadius() {

return radius;

}

*/\*\* Присваивает новый радиус \*/*

public void setRadius(double newRadius)

throws IllegalArgumentException {

if (newRadius >= 0)

radius = newRadius;

else

throw new IllegalArgumentException(

"Радиус не может быть отрицательным");

}

*/\*\* Возвращает numberOfObjects \*/*

public static int getNumberOfObjects() {

return numberOfObjects;

}

*/\*\* Возвращает площадь круга \*/*

public double findArea() {

return radius \* radius \* 3.14159;

}

}

Исходный класс Circle остался нетронутым, за исключением того, что имя класса изменено на CircleWithException, добавлен новый конструктор CircleWithException(newRadius), а метод setRadius() теперь объявляет исключение и выбрасывает его, если радиус отрицательный.

Метод setRadius() объявляет в своем заголовке о выбрасывании исключения IllegalArgumentException (в классе CircleWithException). Класс CircleWithException будет по-прежнему компилироваться, если часть заголовка throws IllegalArgumentException будет удалена из объявления метода, так как этот класс является подклассом RuntimeException, и каждый метод может бросить RuntimeException (непроверяемое исключение) независимо от того, объявлено ли оно в заголовке метода или нет.

В программе-клиенте создается три объекта типа CircleWithException – c1, c2 и c3 – для проверки обработки исключений. При выполнении new CircleWithException(-5) (в классе TestCircleWithException) вызывается метод setRadius() и выбрасывается исключение IllegalArgumentException, так как радиус отрицательный. В блоке catch типом объекта ex является IllegalArgumentException, который соответствует объекту исключения, выброшенному методом setRadius(), поэтому это исключение перехватывается блоком catch. Обработчик исключений отображает в консоли короткое сообщение об исключении с помощью System.out.println(ex). В случае исключения выполнение программы продолжается. Если бы обработчики не перехватили исключение, то программа бы прервалась. Программа-клиент будет по-прежнему компилироваться, если предложение try не будет использоваться, так как метод выбрасывает экземпляр класса IllegalArgumentException, подкласса RuntimeException (непроверяемое исключение).

## Задание 1

С помощью двух массивов напишите программу, которая предложит пользователю ввести целое число от 1 до 12, а затем отобразит месяц и количество дней, соответствующие этому целому числу. Если пользователь вводит недопустимое число, то программа должна отображать Недопустимое число с помощью перехвата ArrayIndexOutOfBoundsException. Программа должна быть реализована таким образом, чтобы предотвратить ввод пользователем любого числа, кроме целого.

java

String[] months = {"январь", "февраль", "март", "апрель", "май",

"июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь", "декабрь"};

int[] dom = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

## Задание 2

Измените в программе TestLoanClass класс Loan таким образом, чтобы выбросить IllegalArgumentException, если годовая процентная ставка, срок или сумма кредита меньше или равны нулю.

## Задание 3

Дополните код 1 задания данной практической работы таким образом, чтобы при выборе февраля была возможность записать год. Добавьте в программу метод для расчета является ли год високосным и измените вывод для количества дней в феврале.