Программирование на Java

Лекция 3. Объектно-ориентированное программирование.

Основные понятия ООП

- **Инкапсуляция** – ограничение доступа к данным и их объединение с подпрограммами, обрабатывающими эти данные. Данные называют полями, а подпрограммы – методами класса. Поля и методы называют членами класса.
- **Наследование** – возможность строить на основе первоначального класса другие классы, добавляя новые поля и методы. Первоначальный класс называется прародителем или суперклассом, а новые классы – его потомками или подклассами.
- **Полиморфизм** – явление, при котором методу с одним и тем же именем соответствует разный программный код. Метод исполняется по-разному в зависимости от типа объекта.

Объявление класса

```
спецификатор_доступа class Имя_класса extends Суперкласс
implements Базовые_интерфейсы {
    спецификатор_доступа тип имя_поля1;
    спецификатор_доступа тип имя_поля2;
    /...
    спецификатор_доступа Конструктор1(аргументы) {
        // тело конструктора1
    }
    спецификатор_доступа Конструктор2(аргументы) {
        // тело конструктора2
    }
    /...
    спецификатор_доступа возвращаемый_тип метод1(аргументы) {
        // тело метода1
    }
    спецификатор_доступа возвращаемый_тип метод2(аргументы) {
        // тело метода2
    }
    /...
}
```

Спецификаторы доступа

- **public** - член класса доступен из любого кода.
- **protected** - член класса доступен только из данного класса и его потомков.
- **private** - член класса доступен только из данного класса.

Конструкторы

Имя конструктора класса должно совпадать с именем класса. В классе может быть несколько конструкторов, различающихся принимаемыми аргументами. Конструктор вызывается при создании объекта класса и предназначен для его инициализации. Конструктор не возвращает никаких значений.

Пример класса

```
public class Circle {
    public double centerX, centerY;
    private double radius;
    public Circle() {
        centerX = centerY = 0.0;
        radius = 1.0;
    }
    public Circle(double cX, double cY, double r) {
        centerX = cX;
        centerY = cY;
        if(r > 0)
            radius = r;
        else
            radius = 1.0;
    }
    public void setRadius(double r) {
        if(r > 0) radius = r;
    }
    public double getRadius() {
        return radius;
    }
}
```

Создание экземпляра класса

```
// объявление
Имя_класса имя_переменной;

// инициализация
имя_переменной = new Имя_конструктора
(аргументы_конструктора);
```

Пример:

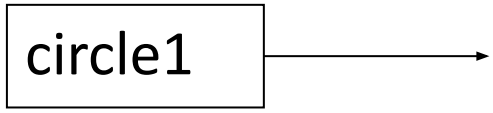
```
Circle circle1, circle2;
circle1 = new Circle();
circle2 = new Circle(4.0, 5.5, 2.0);
```

или

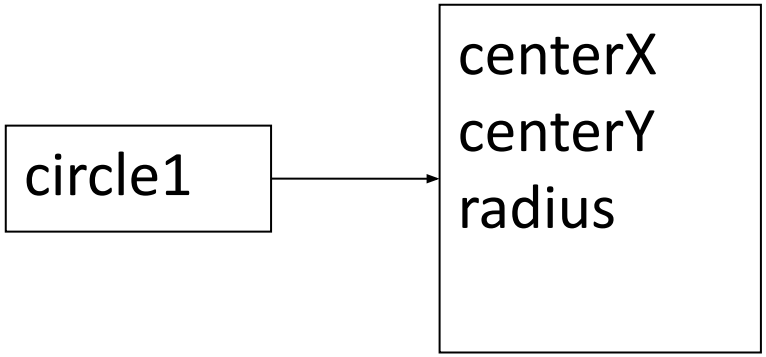
```
Circle circle1 = new Circle();
Circle circle2 = new Circle(4.0, 5.5, 2.0);
```

Создание экземпляра класса

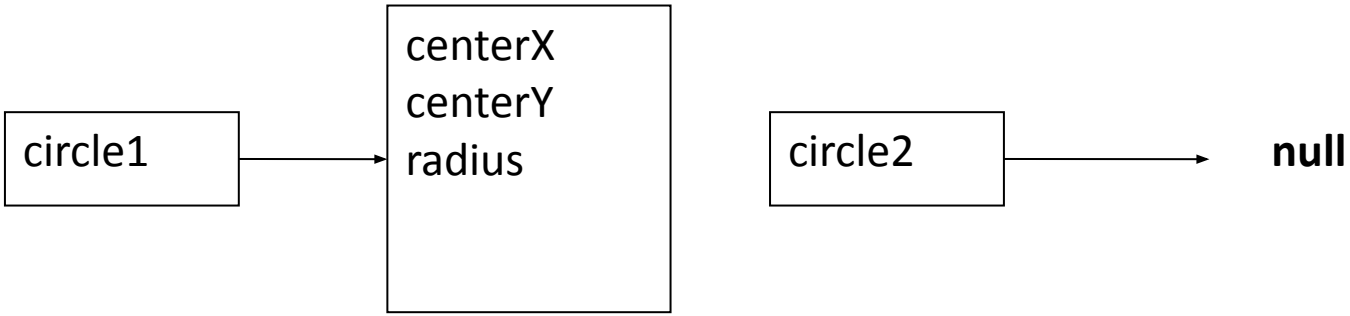
```
Circle circle1;
```



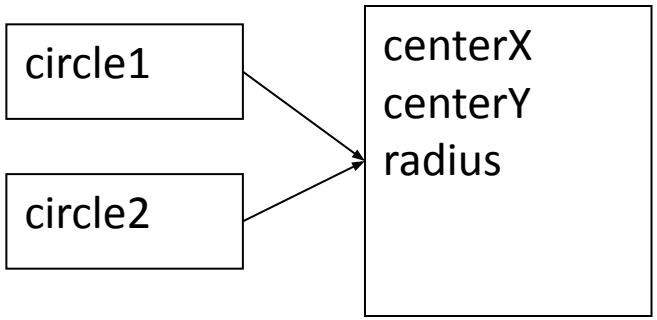

```
circle1 = new Circle();
```



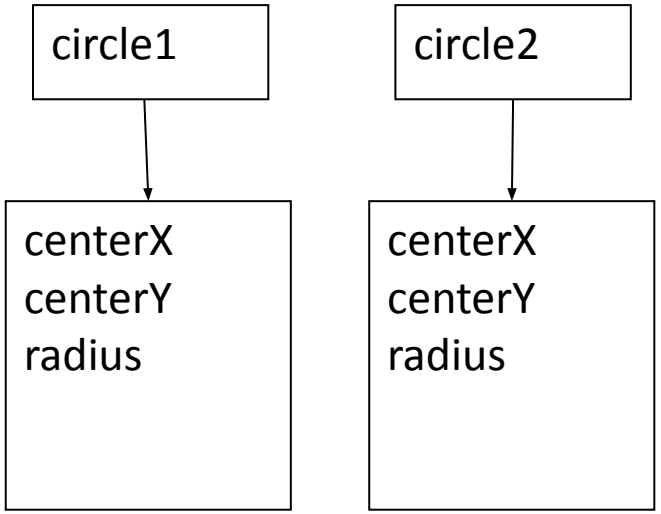
Копирование




```
// Копирование ссылки
circle2 = circle1;
```




```
// Копирование объекта
circle2 = circle1.clone();
```



Сравнение

- Сравнение ссылок (оператор ==) - истинно, если переменные ссылаются на один и тот же экземпляр класса.
  Пример:
```
if(circle2 == circle1)
    System.out.println(“circle1 и circle2 ссылаются на один и тот же объект”);
```
- Сравнение объектов (метод boolean Object.equals(Object obj)) - истинно, если переменные ссылаются на равные экземпляры класса.
  Пример:
```
if(circle2.equals(circle1))
    System.out.println(“circle1 и circle2 ссылаются на равные объекты”);
```

Перегрузка методов

Синтаксис Java позволяет создавать в одном классе методы с одинаковыми именами, различающиеся только принимаемыми аргументами.

Пример:

```
public class Rectangle {
    private double width, height;
    public void setSize(int a) {
        width = a;
        height = a;
    }
    public void setSize(double a) {
        width = a;
        height = a;
    }
    public void setSize(double w, double h) {
        width = w;
        height = h;
    }
}
```

Зарезервированное слово this

Зарезервированное слово **this** предоставляет доступ к полям, методам и конструкторам данного класса. Его удобно использовать, если члены класса оказываются недоступны, из-за наличия в области видимости других переменных или функций с такими же именами, или при вызове одного конструктора из другого.

Пример:

```
public class Circle {
    public double cX, cY;
    private double r = 1.0;
    public Circle(double cX, double cY, double r) {
        this.cX = cX;
        this.cY = cY;
        this.setRadius(r);
    }
    public Circle() {
        this(0.0, 0.0, 1.0);
    }
    public void setRadius(double r) {
        if(r > 0) radius = r;
    }
    public double getRadius() {
        return radius;
    }
}
```

Наследование

У любого класса в Java может быть только один класс-прародитель. Он указывается с помощью зарезервированного слова **extends** после имени класса. Если класс-прародитель не указан, прародителем считается класс Object.

Пример:

```
class Point {
    // Тело класса
}
class Circle extends Point {
    // Тело класса
}
class Rectangle extends Point {
    // Тело класса
}
```

Переопределение методов

Класс-потомок может переопределять методы класса прародителя.
Пример:

```
class Point {
    public double x, y;
    public double getSquare() {
        return 0;
    }
}
class Circle extends Point {
    public double r;
    public double getSquare() {
        return Math.PI * r * r;
    }
}
class Rectangle extends Point {
    public double width, height;
    public double getSquare() {
        return width * height;
    }
}
```

Зарезервированное слово super

Зарезервированное слово **super** предоставляет доступ к полям, методам и конструкторам класса-прародителя.

Пример:

```
class Point {
    public double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
class Circle extends Point {
    public double r;
    public Circle(double x, double y, double r) {
        super(x, y);
        this.r = r;
    }
    public boolean inCircle(double x, double y) {
        return ( (super.x - x)*(super.x - x) +
            (super.y - y)*(super.y - y) < r*r);
}
```

Использование подклассов и суперклассов

Пример:

```
Point point1, point2;
Circle circle1 = new Circle(0.0, 5.0, 2.5);
point1 = (Point)circle1;
point2 = new Rectangle(0.0, 0.0, 5.0, 3.0);
circle1.getSquare();
point1.getSquare();
point2.getSquare();
```

Оператор instanceof

Оператор **instanceof** проверяет принадлежность объекта к какому-либо классу.

Пример:

```
Point point1;
Circle circle1, circles[];
// ...
if(point1 instanceof Point)
    System.out.println(“point1 – объект класса
Point”);
point1 instanceof Circle; // вернёт false
circle1 instanceof Point; // вернёт true
circle1 instanceof Circle; // вернёт true
circles instanceof Circle; // вернёт false
circles[0] instanceof Circle; // вернёт true
```

Интерфейсы

Интерфейсы в Java предназначены для поддержки возможности множественного наследования.

Объявление интерфейса:

```
спецификатор_доступа interface Имя_интерфейса
extends Базовые_интерфейсы {
    спецификатор_доступа тип константа1 = значение1;
    спецификатор_доступа тип константа2 = значение2;
    /...
    спецификатор_доступа возвращаемый_тип
заголовок_метода1(аргументы);
    спецификатор_доступа возвращаемый_тип
заголовок_метода2(аргументы);
    /...
}
```

Использование интерфейсов

```
class Point {
    public double x, y;
}
interface Squareable {
    public double getSquare();
}
class Circle extends Point implements Squareable {
    public double r;
    public double getSquare() {
        return Math.PI * r * r;
    }
}
class Rectangle extends Point implements Squareable {
    public double width, height;
    public double getSquare() {
        return width * height;
    }
}
```

```
Squareable figures[] = new Squareable[3];
figures[0] = new Circle(0.0, 0.0, 3.0);
figures[1] = new Rectangle(0.0, 0.0, 3.0, 4.5);
figures[2] = new Rectangle(1.5, -0.5, 3.0, 4.5);
for(int i = 0; i < figures.length; i++)
    System.out.println("Площадь фигуры " + (i+1) +
" равна " + figures.getSquare());
```

Исключения

В Java предусмотрен механизм обработки исключений. Исключением называется ошибка времени выполнения программы. Исключения в Java реализованы в виде объектов, описывающих исключительную ситуацию. В случае возникновения ошибки времени выполнения, создаются объект-исключение и управление передаётся соответствующему этому объекту обработчику исключений.

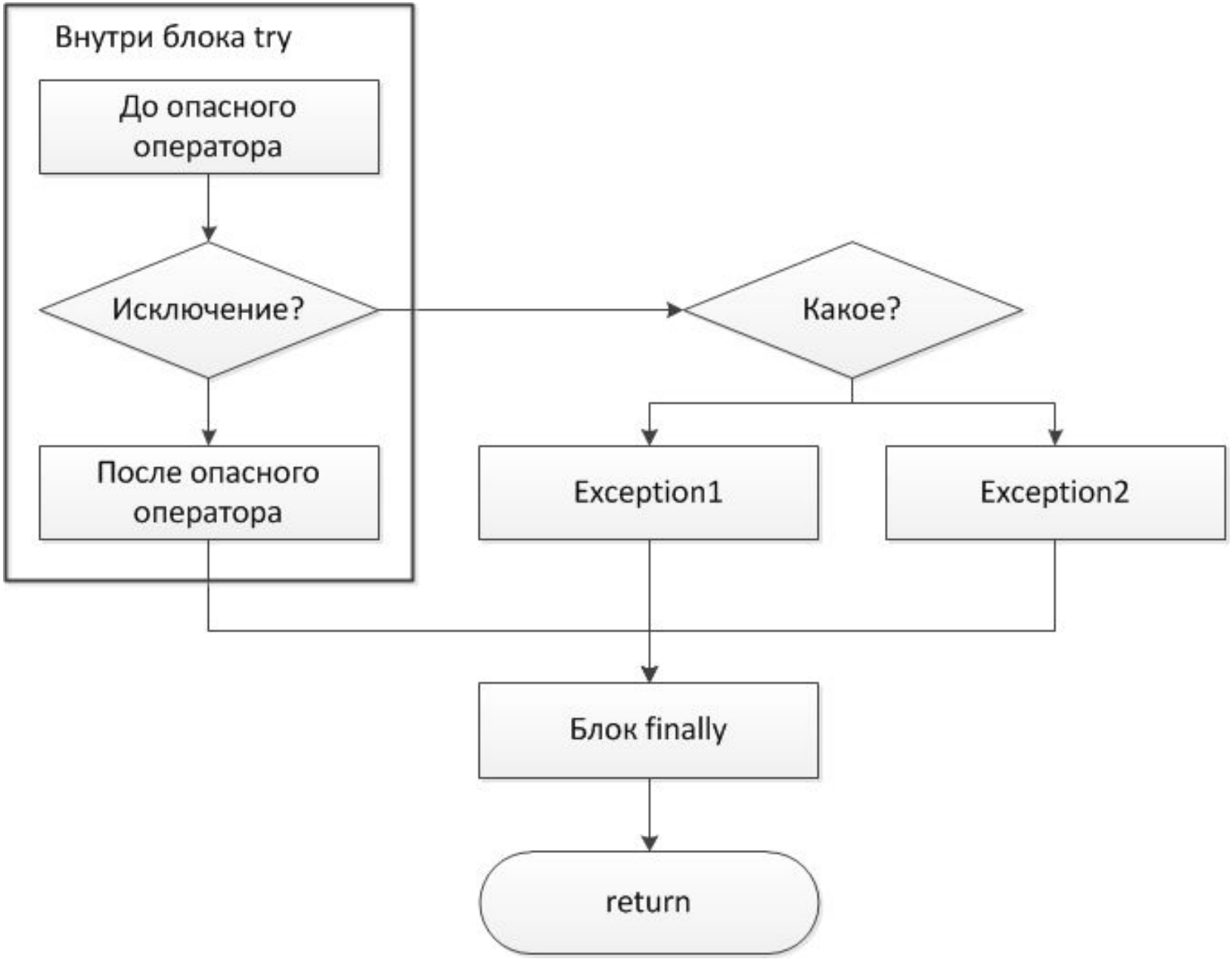
Конструкция try

В общем случае для обработки исключений используется следующая конструкция.

```
try {
   // здесь возможно возникновение исключения
} catch(тип_исключения1 переменная1) {
   // обработчик исключения типа тип_исключения1
} catch(тип_исключения2 переменная2) {
   // обработчик исключения типа тип_исключения2
}
// ...
finally {
   // код, который выполняется в любом случае после
выполнения блока try или завершения обработки
исключения в блоке catch
}
```

Использование конструкции try

```
try {
    // здесь возможно возникновение исключения
    return;
} catch(Exception1 e) {
    // обработчик исключения Exception1
} catch(Exception2 e) {
    // обработчик исключения Exception2
} finally {
    // блок finally
}
```


Внутри блока try
До опасного оператора
Исключение?
После опасного оператора
Какое?
Exception1
Exception2
Блок finally
return

Использование конструкции try

```
try {
    int a = 4 / 0; // деление на 0
} catch(ArithmeticException e) {
    System.out.println(“Исключение: “ + e);
}
```

Приоритет обработчиков исключений

```
try {
    int a = 4 / 0; // деление на 0
} catch(Exception e) {
    System.out.println(“Исключение: “ + e);
} catch(ArithmeticException e) {
    // этот блок не выполнится, потому что класс
ArithmeticException является подклассом класса
Exception
    System.out.println(“Исключение: “ + e);
} finally {
    System.out.println(“Этот блок выполнится в любом
случае после завершения блока try или обработки
исключения”);
}
```

Генерация исключения

Для генерации исключений в Java предназначен оператор **throw**, которому передаётся объект исключения. Обычно этот объект создаётся непосредственно при вызове оператора **throw**.

Пример:

```
ArithmeticException e = new ArithmeticException();
throw e;
```

или

```
throw new Exception();
```

Оператор throws

Если внутри функции может быть сгенерировано исключение, необработанное с помощью конструкции **try**, после объявления этой функции должно стоять зарезервированное слово **throws** и тип генерируемого исключения.

Пример:

```
public void someFunction()
throws SomeException {
    // ...
    throw new SomeException();
    // ...
}
```

Эти и другие материалы можно найти по адресу
http://drop.io/ibts_java

Официальная документация JDK6
http://download.oracle.com/javase/6/docs/api/