
DIASOFT
всё по-настоящему

DIASOFT
всё по-настоящему

VIII. Вложенные классы и перечисления

3. Локальные и анонимные классы

Локальные классы

	Локальный класс – внутренний класс объявленный внутри блока кода, такого как тело метода, конструктор или блок инициализации. Локальный класс не является членом класса, к которому относится блок, а принадлежит самому блоку. Локальные классы недоступны за пределами блока. Единственный модификатор который можно к ним применять это модификатор final запрещающий наследование.

Локальный компаратор

```
public class LocalComparatorDemo {
    public static void main(String args[]) {

        List<String> produce = new ArrayList<String>();

        produce.add("Orange");
        produce.add("Grapes");
        produce.add("Banana");
        produce.add("Apple");
        produce.add("Watermelon");
        produce.add("Kiwi");
        produce.add("Melon");

        System.out.println(produce);

        System.out.println("Sorting in reverse order...");

        class ReverseComparator implements Comparator<String> {
            public int compare(String s1, String s2) {
                return s2.compareTo(s1);
            }
        }

        Collections.sort(produce, new ReverseComparator());
        System.out.println(produce);
    }
}
```

```
[Orange, Grapes, Banana, Apple, Watermelon, Kiwi, Melon]
Sorting in reverse order...
[Watermelon, Orange, Melon, Kiwi, Grapes, Banana, Apple]
```

Локальный итератор

DIASOFT
всё по-настоящему

```
class LocalStringList implements Iterable<String> {

    private String[] strings = new String[100];
    private int size;

    public void add(String string) {
        strings[size++] = string;
    }

    public Iterator<String> iterator() {

        class LocalIterator implements Iterator<String> {

            private int position = 0;

            public String next() {
                return strings[position++];
            }

            public boolean hasNext() {
                return position < size;
            }

            public void remove() {
                throw new UnsupportedOperationException();
            }
        }
        return new LocalIterator();
    }
}
```

Локальный итератор

```
public class LocalIteratorDemo {

    public static void main(String[] args) {

        LocalStringList list = new LocalStringList();
        list.add("Harry Hacker");
        list.add("Tony Tester");
        list.add("Cindy Coder");

        Iterator<String> iter = list.iterator();

        while (iter.hasNext()) {
            System.out.println(iter.next());
        }
    }
}
```

```
Harry Hacker
Tony Tester
Cindy Coder
```

Анонимные классы

DIASOFT
всё по-настоящему

ANONYMOUS

Анонимный класс – локальный класс без имени. Анонимный класс является подклассом существующего класса или реализацией интерфейса. Нельзя создавать ссылочные переменные типа анонимного класса. Для хранения ссылки на объект анонимного класса необходимо использовать ссылку на базовый класс или реализуемый интерфейс. Использование анонимных классов оправдано во многих случаях, в частности когда: тело класса является очень коротким; нужен только один экземпляр класса; класс используется в месте его создания или сразу после него; имя класса не важно и не облегчает понимание кода.

Анонимный компаратор

DIASOFT
всё по-настоящему

```
public class AnonymousComparator {
    public static void main(String args[]) {

        List<String> list = new ArrayList<String>();
        list.add("Harry Hacker");
        list.add("Tony Tester");
        list.add("Alice Coder");
        System.out.println(list);

        Collections.sort(list);
        System.out.println(list);

        Collections.sort(list, new Comparator<String>() {
            public int compare(String s1, String s2) {
                return s2.compareTo(s1);
            }
        });

        System.out.println(list);
    }
}
```

```
[Harry Hacker, Tony Tester, Alice Coder]
[Alice Coder, Harry Hacker, Tony Tester]
[Tony Tester, Harry Hacker, Alice Coder]
```

Анонимный итератор

DIASOFT
всё по-настоящему

```
class AnonymousStringList implements Iterable<String> {

    private String[] strings = new String[100];
    private int size;

    public void add(String string) {
        strings[size++] = string;
    }

    public Iterator<String> iterator() {
        return new Iterator<String>() {

            private int position = 0;

            public String next() {
                return strings[position++];
            }

            public boolean hasNext() {
                return position < size;
            }

            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
    }
}
```

Анонимный итератор

DIASOFT
всё по-настоящему

```
public class AnonymousIteratorDemo {

    public static void main(String[] args) {

        AnonymousStringList list = new AnonymousStringList();
        list.add("Harry Hacker");
        list.add("Tony Tester");
        list.add("Cindy Coder");

        Iterator<String> iter = list.iterator();

        while (iter.hasNext()) {
            System.out.println(iter.next());
        }
    }
}
```

```
Harry Hacker
Tony Tester
Cindy Coder
```

Уникальность объявления анонимных классов

Каждое объявление анонимного класса уникально. Для каждого объвленного анонимного класса создаётся отдельный класс. Когда создаются class файлы для анонимных классов обычно названия файлов Class$1.class, Class$2.class, … где Class название внешнего класса. Но полагаться на такое именование анонимных классов не следует.

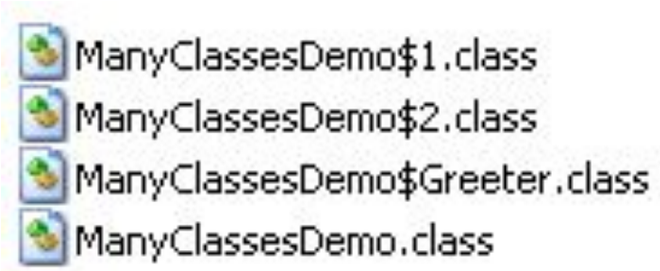
Несколько анонимных классов

DIASOFT
всё по-настоящему

```
import java.util.Comparator;

public class ManyAnonymousDemo {

    interface Greeter {
        public void greet(String name);
    }

    public static void main(String[] args) {

        Greeter firstGreeter = new Greeter() {

            public void greet(String name) {
                System.out.println("Hello " + name);
            }
        };

        Greeter secondGreeter = new Greeter() {

            public void greet(String name) {
                System.out.println("Hello " + name);
            }
        };

        firstGreeter.greet("Harry Hacker");
        secondGreeter.greet("Tony Tester");
    }
}
```

```
Hello Harry Hacker
Hello Tony Tester
```

Class файлы для анонимных классов



Анонимные классы и полиморфизм

При использовании анонимных классов можно вызывать только методы которые объявлены в базовом классе или реализуемом интерфейсе. Таким образом как и для обычных классов методы которые можно вызывать определяются типом ссылки.

Анонимные классы и конструкторы

Вызов конструктора базового класса

Основное ограничение при использовании анонимных классов - это невозможность описания конструктора, так как класс не имеет имени. Аргументы, указанные в скобках, автоматически используются для вызова конструктора базового класса с теми же параметрами.

Вызов конструктора базового класса

DIASOFT
всё по-настоящему

```
public class BaseConstructorDemo {

    static abstract class Greeter {
        private String greeting;

        public Greeter(String greeting) {
            this.greeting = greeting;
        }

        public void greet(String name) {
            System.out.println(greeting + " " + name);
        }
    }

    public static void main(String[] args) {

        Greeter doubleEnglishGreeter = new Greeter("Hello") {

            public void greet(String name) {
                super.greet(name);
                super.greet(name);
            }
        };

        Greeter tripleFrenchGreeter = new Greeter("Salut") {

            public void greet(String name) {
                super.greet(name);
                super.greet(name);
                super.greet(name);
            }
        };

        doubleEnglishGreeter.greet("Harry Hacker");
        tripleFrenchGreeter.greet("Tonny Tester");
    }
}
```

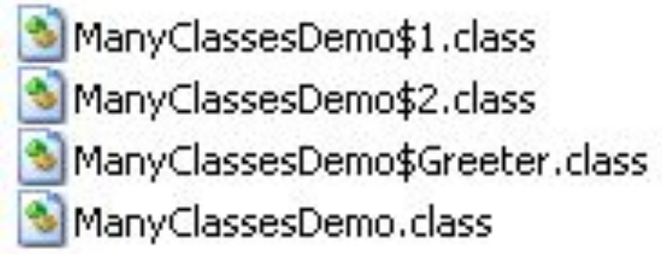
Вызов конструктора базового класса

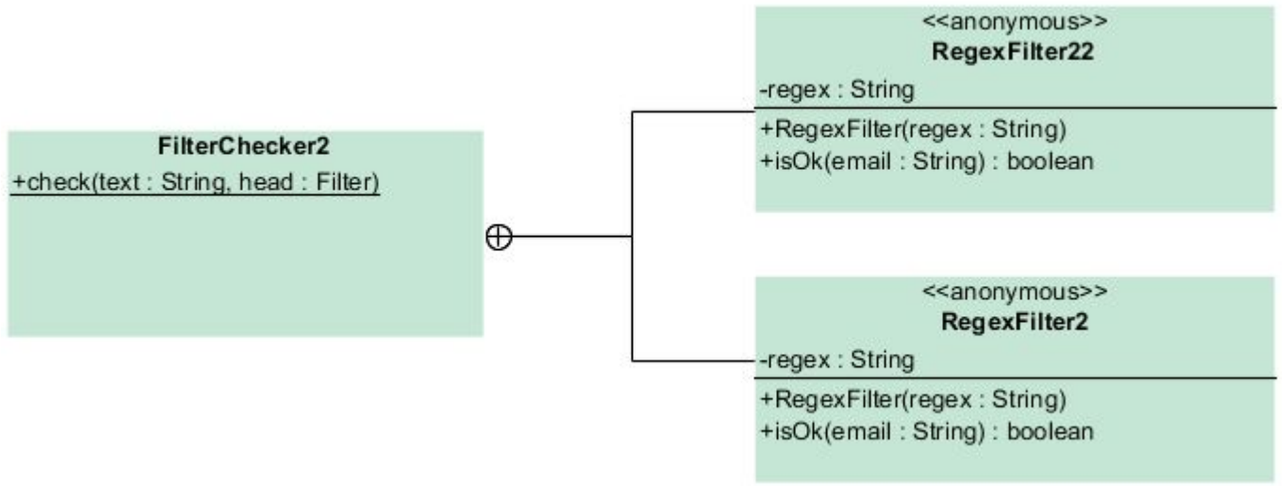
```
Hello Harry Hacker
Hello Harry Hacker
Salut Tonny Tester
Salut Tonny Tester
Salut Tonny Tester
```

Спасибо

Россия, 127018,
Москва, ул. Полковая 3, стр. 14
Тел.: +7(495) 780 7575, 789 9339
Факс: +7(495) 780 7576, 789 9338
info@diasoft.ru, www.diasoft.ru

Class файлы для анонимных классов





Анонимный компаратор

DIASOFT
всё по-настоящему

```
public class AnonymousThreadDemo {

    public static void main(String[] args) {

        Thread t = new Thread() {
            public void run() {
                for (int i = 0; i < 10; i++) {
                    Date now = new Date();
                    System.out.println("Hello World! " + now);
                    try {
                        sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        t.start();
    }
}
```

```
Hello World! Thu Jan 03 15:35:52 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:53 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:54 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:55 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:56 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:57 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:58 GMT+07:00 2013
Hello World! Thu Jan 03 15:35:59 GMT+07:00 2013
Hello World! Thu Jan 03 15:36:00 GMT+07:00 2013
Hello World! Thu Jan 03 15:36:01 GMT+07:00 2013
```

Анонимные классы сделать наблюдатель

Here, we have gone over to the terse side. We've allocated and started a new Thread, using an anonymous inner class that extends the Thread class and invokes our performBehavior( ) method in its run( ) method. The effect is similar to using a method pointer in some other language. However, the inner class allows the compiler to check type consistency, which would be more difficult (or impossible) with a true method pointer. At the same time, our anonymous adapter class with its three lines of code is much more efficient and readable than creating a new, top-level adapter class named AnimalBehaviorThreadAdapter.

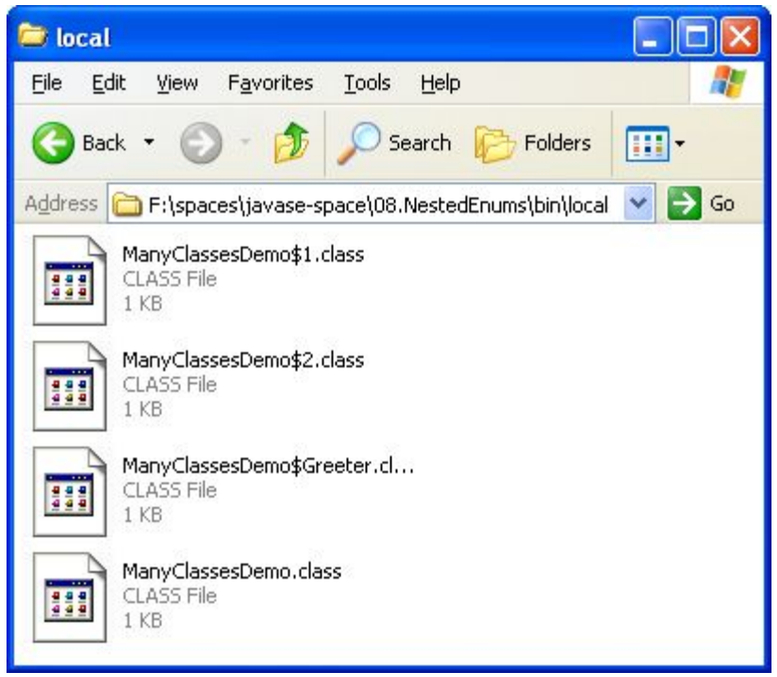
While we're getting a bit ahead of the story, anonymous adapter classes are a perfect fit for event handling (which we cover fully in Chapter 16). Skipping a lot of explanation, let's say you want the method handleClicks( ) to be called whenever the user clicks the mouse. You would write code such as:

```
addMouseListener( new MouseInputAdapter( ) { public void mouseClicked(MouseEvent e) {
handleClicks(e); } } );
```

In this case, the anonymous class extends the MouseInputAdapter class by overriding its mouseClicked( ) method to call our method. A lot is going on in a very small space, but the result is clean, readable code. You assign method names that are meaningful to you while allowing Java to do its job of type checking.

he previous example is a bit extreme and certainly does not improve readability. Inner classes are best used when you want to implement a few lines of code, but the verbiage and conspicuousness of declaring a separate class detracts from the task at hand. Here's a better example. Suppose that we want to start a new thread to execute the performBehavior( ) method of our Animal:

```
new Thread( ) { public void run( ) { performBehavior( ); } }.start( );
```


local
File Edit View Favorites Tools Help
Back Search Folders
Address F:\spaces\javase-space\08.NestedEnums\bin\local
Go
ManyClassesDemo$1.class
CLASS File
1 KB
ManyClassesDemo$2.class
CLASS File
1 KB
ManyClassesDemo$Greeter.cl...
CLASS File
1 KB
ManyClassesDemo.class
CLASS File
1 KB

Итератор как анонимный класс

```
public class MyList implements Iterable {
    private Object [ ] a;
    private int size;
    public Iterator iterator( ) {
        return new Iterator ( ) {
          private int pos = 0;
          public boolean hasNext( ) { return pos < size; }
          public Object next( )     { return a[pos++]; }
        } ;
    }
}
```

Anonymous List

```
public class MyList implements Iterable {
    private Object [ ] a;
    private int size;
    public Iterator iterator( ) {
        return new Iterator ( ) {
          private int pos = 0;
          public boolean hasNext( ) { return pos < size; }
          public Object next( )       { return a[pos++]; }
        } ;
    }
}
```

DIASOFT
всё по-настоящему

MyList With *Explicit* Inner Class

- **Code**

```
public class MyList implements Iterable {
    private Object [ ] a;
    private int size;
    public Iterator iterator( ) {
        return new MyIterator( );
    }
    public class MyIterator implements Iterator {
        private int pos = 0;
        public boolean hasNext( ) { return pos < size; }
        public Object next( )     { return a[pos++]; }
    }
}
```

Using Inner Classes in GUIs

```
javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndDisplayGUI();
        }
} );

button.addActionListener (new ActionListener( ) {
        public void actionPerformed (ActionEvent evt) {
            System.out.println("Button pushed");
        }
} );
```


Nested Class

Спа

Anonymous Inner Class

- **Description**
  - Inner class without name
  - Defined where you create an instance of it
    - In the middle of a method
    - Returns an instance of anonymous inner class
  - Useful if the only thing you want to do with an inner class is create instances of it in one location
- **Syntax**

```
new ReturnType( ) {      // unnamed inner class
        body of class…  // implementing ReturnType
} ;
```

Россия
Москв
Тел.: +
Факс:
info@c