Класс Object и его методы

Класс Object

В Java **все классы наследуются от Object**.

Класс Object предоставляет следующие **методы, которые есть у всех объектов**:

1. **getClass()** - возвращает ссылку на класс объекта
2. **hashCode()** - возвращает хэш-код объекта, по умолчанию это рассчитанное специальным образом число на основании генератора случайных чисел;
3. **equals(Object obj)** — возвращает true, если объекты одинаковы, по умолчанию выполняет сравнение по ссылке ==;
4. **clone()** - создает копию объекта;

Класс Object

Класс Object предоставляет следующие **методы, которые есть у всех объектов**:

5. **toString()** - возвращает строковое представление объекта, например для удобства вывода в консоль;
6. **finalize()** - выполняется при уничтожении объекта сборщиком мусора;
7. методы **wait()**, **notify()** и **notifyAll()** используются при многопоточном программировании.

Класс Object

Для определения идентичности объектов используют
методы **equals()** и **hashCode()**.

Они переопределяются оба для соответствия следующим требованиям:
equals:
1. Рефлективность — объект всегда равен самому себе: a.equals(a).
2. Симметричность — если a.equals(b) , то и b.equals(a).
3. Транзитивность — если a.equals(b), b.equals(c), то и a.equals(c).
4. Консистентность — сколько бы раз не вызывался equals без изменения состояния объектов, результат должен оставаться неизменным.

Переопределенный equals проверяет сначала **идентичность с помощью** ==,
потом **на null** и **тип переданного объекта**,
после чего **сравнивает** при помощи equals **все поля**

Класс Object

Для определения идентичности объектов используют методы **equals()** и **hashCode()**.

Они переопределяются оба для соответствия следующим требованиям: **hashCode**:

1. Сколько раз бы не был вызван hashCode на объекте, он должен возвращать то же значение. Значение может быть иным при следующем выполнении программы.

2. Если объекты равны по equals(), то hashCode() должен вернуть одинаковое значение для обоих объектов.

3. Если объекты не равны по equals(), hashCode() может возвращать одинаковые значения.

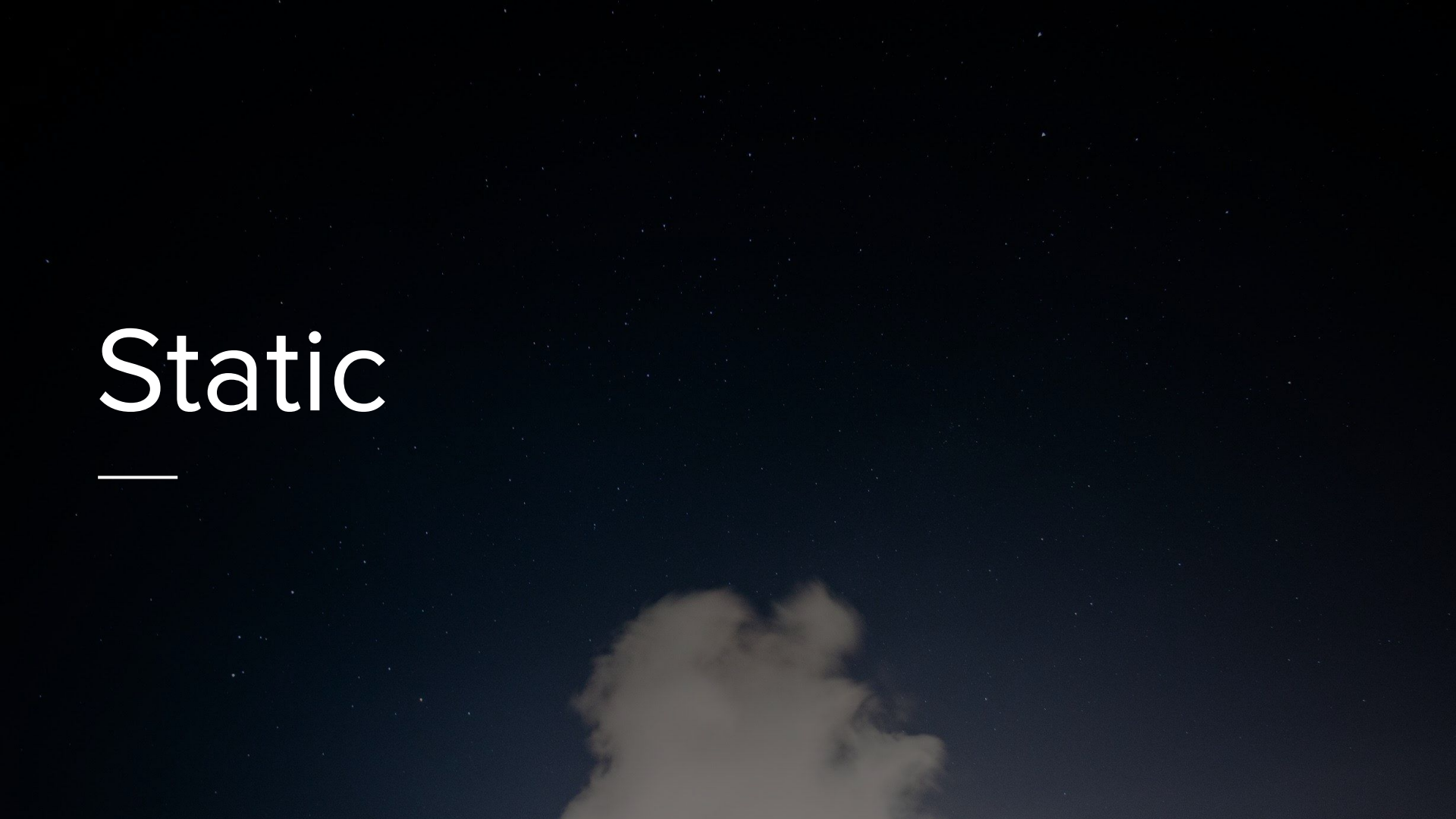
Класс Object

Метод **toString()** вызывается автоматически у объектов, **когда нужно получить его строковое представление**, например, при вызове метода println().

Можно переопределить для удобства вывода.

```
@Override
public String toString() {
        return "Book{" +
                "title='" + title + '\'' +
                ", author='" + author + '\'' +
                ", pages=" + pages +
                '}';
}
```

Static



Оператор Static

1. **Статические переменные** **static int *soldBookCounterStatic*;**

В Java можно использовать ключевое слово static в переменной класса. **Статическая - переменная класса** и **не принадлежит объекту**/экземпляру класса.
Статические переменные **являются общими для всех экземпляров объекта**, они не потоко-безопасные.
Обычно статические переменные используются с ключевым словом final для общих ресурсов или констант, которые могут быть использованы всеми объектами.

Если модификатор доступа позволяет, то получить к ней **доступ** можно следующим способом: **ClassName.staticVarName**

Оператор Static

```
public static void showPriceList(){
   System.out.println("Книги до 500 страниц - 1000р." +
         "Книги более 500 страниц - 2000р.");
}
```

2. **Статические методы**

Статические методы **принадлежат классу**, а не к экземплярам класса.

Статический метод **может получить доступ только к статическим переменным** класса и **вызывать только статические методы класса**.

Обычно статические методы являются утилитными (вспомогательными) методами, которые должны быть использованы другими классами без необходимости создания экземпляра.

Доступ к статическому методу: **ClassName.staticMethodName()**

Начиная с **Java 8**, можно использовать **статические методы в интерфейсах**.

Оператор Static

```
public static int soldBookCounterStatic;
static {
        soldBookCounterStatic = 0;
}
```

3. **Статический блок**

Статический блок с группой операторов используется **для инициализации статических переменных класса**.

В основном он используется для создания статических ресурсов, когда класс загружается.

Нельзя получить доступ к не статическим переменным в статическом блоке.

Статический блок кода **выполняется только один раз, когда класс загружается в память**.

static { ... }

Оператор Static

4. **Статический класс**

Можно использовать **ключевое слово static с вложенными классами**.

Ключевое слово static **не может быть использовано с классами верхнего уровня**.

Final

Оператор Final

final могут быть:

1. переменные
2. методы
3. аргументы методов
4. локальные переменные методов
5. классы

Оператор Final

Переменные final `final String PRINTING_HOUSE;`

Должны быть инициализированы в момент объявления или в конструкторе класса.

Нельзя изменить значение final переменной.

Методы final не могут быть переопределены в дочерних классах.

```
final public void showInfo(){
   System.out.println("soldBookCounter = " + soldBookCounter);
   System.out.println("soldBookCounterStatic = " + soldBookCounterStatic);
}
```

Оператор Final

Аргументы методов final доступны только для чтения.

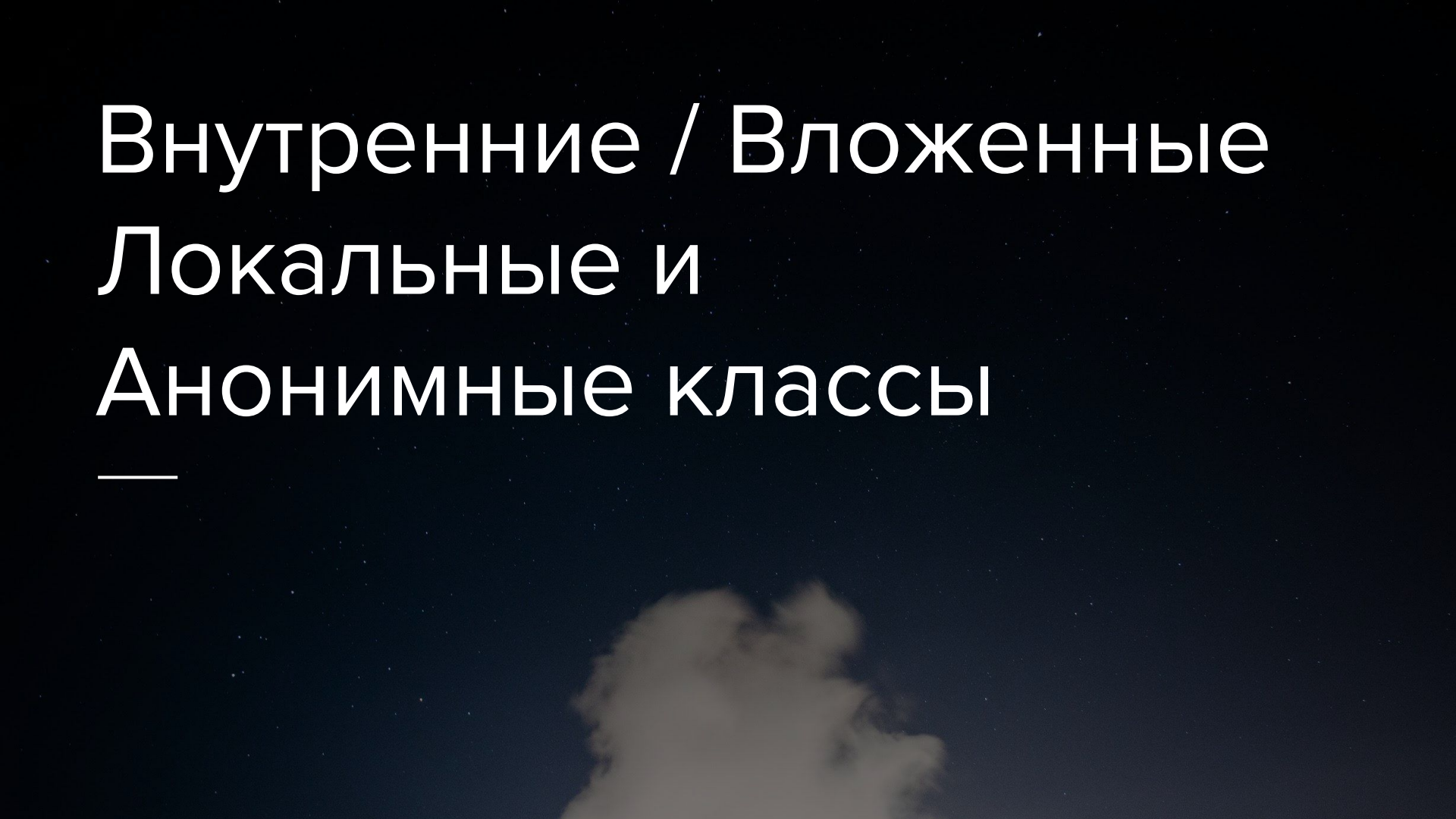
```
public void setDescription(final String description) {
//        description = "Нельзя изменить final аргументы";
        this.description = description; // final аргументы доступны только для чтения
}
```

Классы final не могут иметь наследников.

```
final class Book { тело класса }

class ColoringBook extends Book{ тело класса }
```

Внутренние / Вложенные Локальные и Анонимные классы



Вложенные / Внутренние классы

Часто бывает, что нужен класс в рамках работы другого класса, и больше нигде, а создание нового файла захламляет проект.

Вложенные и внутренние классы позволяют группировать классы, логически принадлежащие друг другу.

Но важно помнить, что **Класс B внутри класса A, не сможет существовать независимо от класса A**.

Для таких случаев часто используют:

- **вложенные** классы **(static nested classes)** и
- **внутренние** классы **(member inner class)**

Особенностью вложенного класса является то, что он не имеет доступа к полям и методам объекта основного класса.

Статический вложенный класс
Nested Inner Class

```
public class OuterClass {
        public static class StaticInnerClass{  // статический вложенный класс
        }
}
```

В StaticInnerClass доступны:

- статические свойства и методы OuterClass (даже private).
- статические свойства и методы родителя OuterClassa public и protected

StaticInnerClass доступен согласно модификатору доступа.

Статический вложенный класс Nested Inner Class

StaticInnerClass может наследовать:

- обычные классы.
- такие же статические внутренние классы в OuterClasse и его предках

StaticInnerClass может имплементировать интерфейс

Может содержать:

- статические свойства и методы.
- не статические свойства и методы.

Экземпляр этого класса создается так:
OuterClass.StaticInnerClass staticInnerClass = new OuterClass.StaticInnerClass();

Внутренний класс
Member Inner Class

```
public class OuterClass {
        public class InnerClass{
        }
}
```

Из InnerClass доступны:

- все (даже private) свойства и методы OuterClassa обычные и статические.
- public и protected свойства и методы родителя OuterClassa обычные и статические.

InnerClass доступен согласно модификатору доступа.

Внутренний класс
Member Inner Class

InnerClass может наследовать:

- обычные классы
- такие же внутренние классы в OuterClasse и его предках

InnerClass может имплементировать интерфейс

InnerClass может содержать только обычные свойства и методы (не статические).

Экзэмпляр этого класса создаётся:
OuterClass outerClass = new OuterClass();
OuterClass.InnerClass innerClass = outerClass.new InnerClass();

Локальный класс
Local Inner Class

```
public class OuterClass {
        public void someMethod(){
                class LocalClass{
                }
         }
}
```

Из LocalClass доступны:

- все (даже private) свойства и методы OuterClassa обычные и статические.
- public и protected свойства и методы родителя OuterClassa обычные и статические.

Локальный класс
Local Inner Class

LocalClass доступен только в том методе где он определён.

LocalClass может наследовать:

- обычные классы
- внутренние классы в OuterClasse и его предках
- такие же локальные классы определенные в том же методе

Может имплементировать интерфейс.

Может содержать только обычные свойства и методы (не статические).

Анонимный класс
Anonymous Class

Класс без имени (ему достаточно только реализации).

Наследует класс или имплементирует какой-то интерфейс.

Из Anonymous класса доступны:

- все (даже private) свойства и методы OuterClassa обычные и статические
- public и protected свойства и методы родителя OuterClassa обычные и статические

Anonymous класс доступен только в том методе где он определён.

Anonymous класс не может быть наследован.

Anonymous класс не может содержать статические свойства и методы.

Anonymous Class

1

```
interface Operation {
        double apply(double a, double b);
}

public class Calculator {
        double a, b, res;
        public Calculator(double a, double b) {
                this.a = a;
                this.b = b;
    }
    public void execute(Operation operation){
        res = operation.apply(a, b);
    }
    public void showRes() {
        System.out.println("res = " + res);
    }
}
```

2

```
public class Main {
    public static void main(String[] args) {
        int a = 1;
        int b = 8;
        Calculator calculator = new Calculator(a,b);
        calculator.execute(new Operation() {
            @Override
            public double apply(double a, double b) {
                return a + b;
            }
        });
        calculator.showRes();

        calculator.execute(new Operation() {
            @Override
            public double apply(double a, double b) {
                return a - b;
            }
        });
        calculator.showRes();
    }
}
```