

Реализация принципа ООП: наследование (продолжение)

РЕАЛИЗАЦИЯ НАСЛЕДОВАНИЯ

```
{<доступ>}0 2 class <имя подкласса> extends <имя суперкласса> {  
    <тело подкласса>  
}
```

Ключевое слово, которое определяет, что описываемый класс расширяет другой класс

- ❑ В зависимости от прав доступа к элементам суперкласса подкласс:
 - наследует открытые (***public***) и защищенные (***protected***) элементы вне зависимости в одном или разных пакетах они определены;
 - наследует элементы ***package-private***, если они определены в одном пакете;
 - закрытые элементы суперкласса (***private***) и конструкторы не наследуются.

Основы наследования

Пример 15:


```
class Person {  
    private String name;  
    public void setName(String name){ this.name = name; }  
    public void show() { System.out.println("Name: " + name); }  
}  
  
class Employee extends Person {  
}  
  
public class Program {  
    public static void main(String[] args) {  
        Person tom = new Person();  
        tom.setName("Tom");  
        tom.show();  
        Employee sam = new Employee();  
        sam.setName("Sam");  
        sam.show();  
    }  
}
```

Вывод консоли:

Tom


Sam

Вызов
унаследованного
метода



Что можно сделать в подклассе?

В отношении полей:

- ❑ Унаследованные поля могут быть использованы непосредственно (напрямую), как и любые другие поля подкласса;
- ❑ Можно объявить в подклассе поле с тем же именем, что и в суперклассе, скрывая его таким образом
 (не рекомендуется);
- ❑ Можно объявить новые поля в подклассе, которых нет в суперклассе.

Что вы можете сделать в подклассе?

В отношении методов:

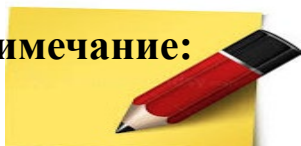
- ❑ Унаследованные методы могут быть использованы непосредственно (напрямую), как есть;
- ❑ Можно в подклассе написать новый метод экземпляра, который будет иметь ту же сигнатуру, что и один из методов суперкласса, таким образом, переопределяя его;
- ❑ Можно в подклассе написать новый метод класса (*статический*), который имеет ту же сигнатуру, что и один из методов суперкласса, таким образом, скрывая его;
- ❑ Можно в подклассе объявить новые методы, которых нет суперклассе.

Что вы можете сделать в подклассе?

В отношении конструкторов:

- ❑ Можно написать конструктор подкласса, который вызывает конструктор суперкласса, неявно или явно (с помощью ключевого слова **super**).

Примечание:



Обычно явный вызов конструктора суперкласса в теле конструктора подкласса используется для инициализации закрытых элементов суперкласса:

super(<список аргументов>);

Основы наследования

Пример 1: инициализация закрытых элементов суперкласса,

```
class Base {  
    private int a, b;  
    Base(int a, int b) {  
        this.a = a;   this.b = b;   }  
}  
class Derived extends Base {  
    private int c;  
    Derived (int a, int b, int c) {  
        super(a, b);  
        this.c = c;  
    }  
}  
  
public class Demo {  
    public static void main(String [] s) {  
        Derived obj = new Derived(1, 2, 3);  
    }  
}
```

Явное обращение к
конструктору суперкласса
должно быть первым
оператором в теле конструктора
подкласса

Создание экземпляра подкласса,
конструктору которого
передаются аргументы для
инициализации собственных
полей и полей суперкласса



Цепочки конструкторов

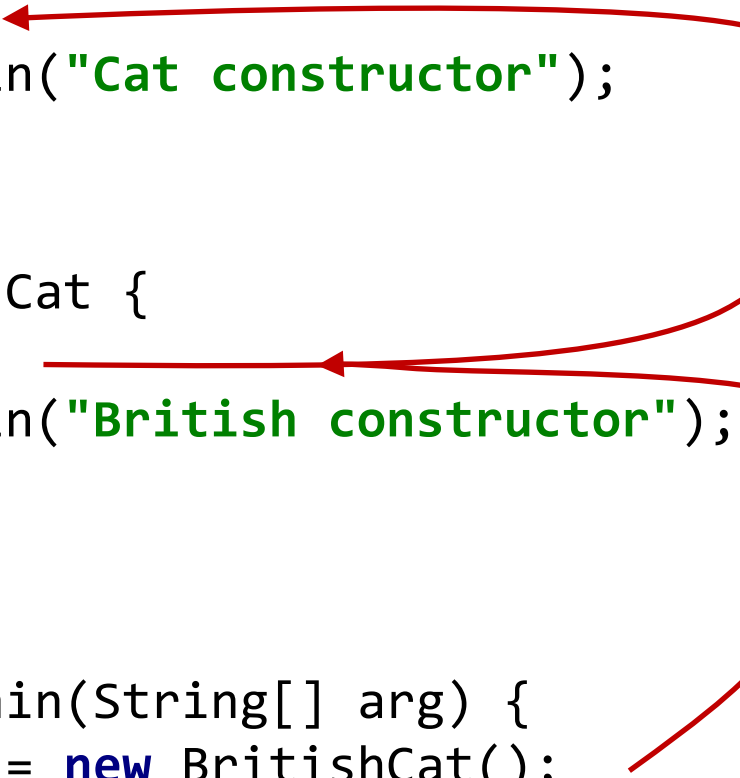
Соглашения

- ❑ При создании объекта подкласса конструкторы всех классов цепочки наследования вызываются в порядке их подчиненности – от суперкласса к подклассу. Такие вызовы называются *цепочками конструкторов*.
- Поэтому явное обращение к конструктору суперкласса должно быть первым оператором конструктора подкласса;
- Если конструктор подкласса не вызывает явно ни один из конструкторов суперкласса, то автоматически вызывается конструктор по умолчанию суперкласса.
- Однако, если в суперклассе нет конструктора по умолчанию, а конструктор подкласса явно не вызывает никакого другого конструктора суперкласса, то компилятор Java выдаст сообщение об ошибке.

Цепочки конструкторов

Пример 2: *вызов конструктора по умолчанию суперкласса*

```
class Cat {  
    public Cat() {  
        System.out.println("Cat constructor");  
    }  
}  
  
class BritishCat extends Cat {  
    public BritishCat(){  
        System.out.println("British constructor");  
    }  
}  
  
public class Main {  
    public static void main(String[] arg) {  
        BritishCat myCat = new BritishCat();  
    }  
}
```



Вывод консоли:

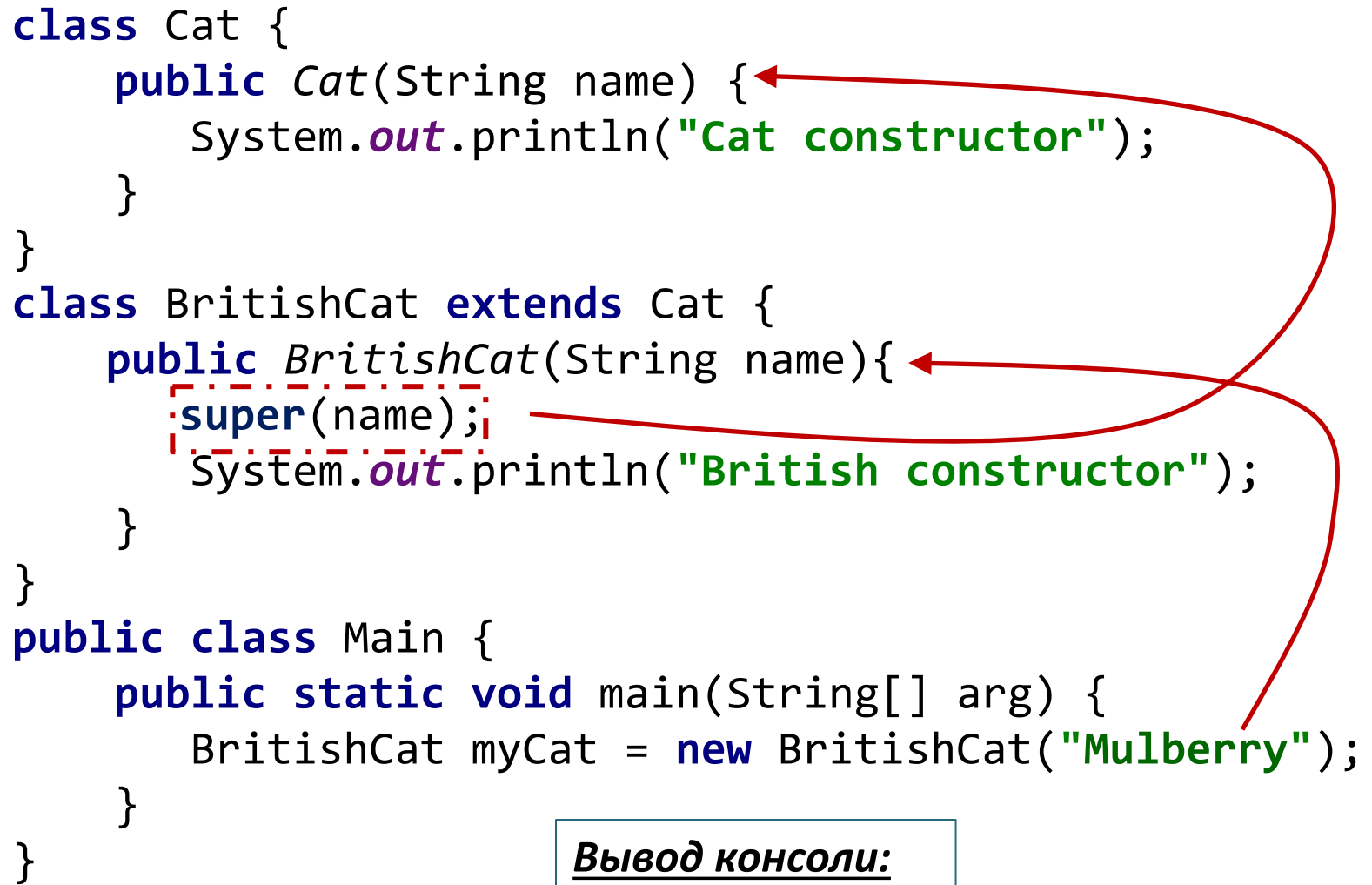
Cat constructor

British constructor

Цепочки конструкторов

Пример 3: вызов конструктора с параметрами суперкласса

```
class Cat {  
    public Cat(String name) {  
        System.out.println("Cat constructor");  
    }  
}  
class BritishCat extends Cat {  
    public BritishCat(String name){  
        super(name);  
        System.out.println("British constructor");  
    }  
}  
public class Main {  
    public static void main(String[] arg) {  
        BritishCat myCat = new BritishCat("Mulberry");  
    }  
}
```



Вывод консоли:

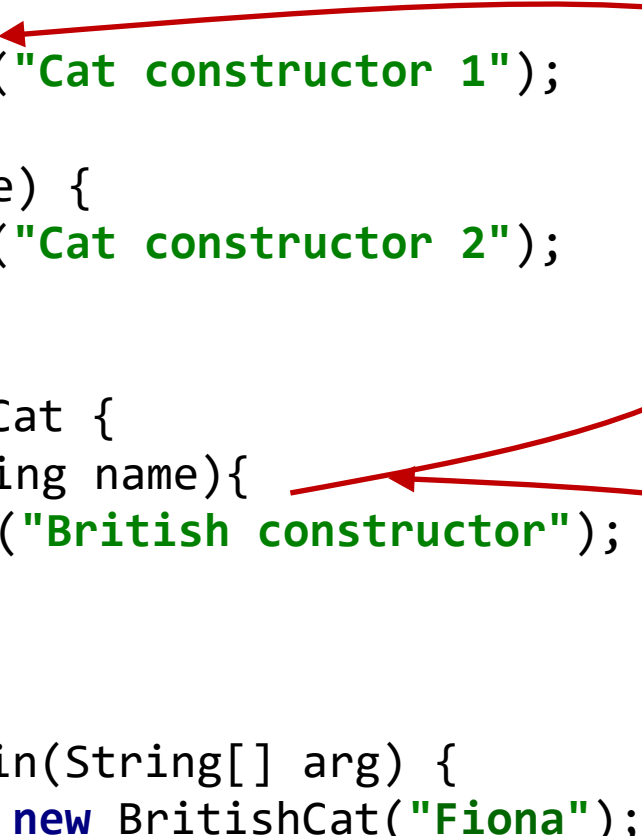
Cat constructor

British constructor

Цепочки конструкторов

Пример 4: вызов конструктора по умолчанию суперкласса

```
class Cat {  
    public Cat() {  
        System.out.println("Cat constructor 1");  
    }  
    public Cat(String name) {  
        System.out.println("Cat constructor 2");  
    }  
}  
class BritishCat extends Cat {  
    public BritishCat(String name){  
        System.out.println("British constructor");  
    }  
}  
public class Main {  
    public static void main(String[] arg) {  
        BritishCat myCat = new BritishCat("Fiona");  
    }  
}
```

A diagram consisting of three red arrows. The first arrow starts from the 'new' keyword in the line 'BritishCat myCat = new BritishCat("Fiona");' and points to the 'public Cat()' constructor of the 'Cat' class. The second arrow starts from the 'BritishCat' part of the same line and points to the 'public BritishCat(String name)' constructor of the 'BritishCat' class. The third arrow starts from the 'Fiona' argument and points to the 'public Cat(String name)' constructor of the 'Cat' class.

Вывод консоли:

Cat constructor 1
British constructor

Цепочки конструкторов

Пример 5: ошибочный вызов конструктора по умолчанию суперкласса

```
class Cat {  
    public Cat(String name) {  
        System.out.println("Cat constructor");  
    }  
}
```

Ошибка компиляции: Java подставила в начало конструктора подкласса вызов конструктора по умолчанию суперкласса, а таковой отсутствует

```
class BritishCat extends Cat {  
    public BritishCat(String name){  
        System.out.println("British constructor");  
    }  
}  
  
public class Main {  
    public static void main(String[] arg) {  
        BritishCat myCat = new BritishCat("Fiona");  
    }  
}
```



Переопределение методов

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ

- ❑ Метод экземпляра подкласса с той же сигнатурой (*имя, количество и типы параметров*) и типом возвращаемого значения, что и метод экземпляра в суперклассе, переопределяет (замещает) этот метод суперкласса.
 - Способность подкласса переопределять методы и позволяет классу наследоваться от некоего суперкласса, чье поведение "достаточно близко" к нему, чтобы изменить поведение по мере необходимости.
- ❑ При переопределении метода можно использовать аннотацию **@Override**, которая указывает компилятору, что это переопределяемый метод суперкласса (*если компилятор обнаруживает, что этот метод не существует ни в одном из суперклассов, то он будет генерировать ошибку*).

Переопределение методов

Пример 6: переопределение методов

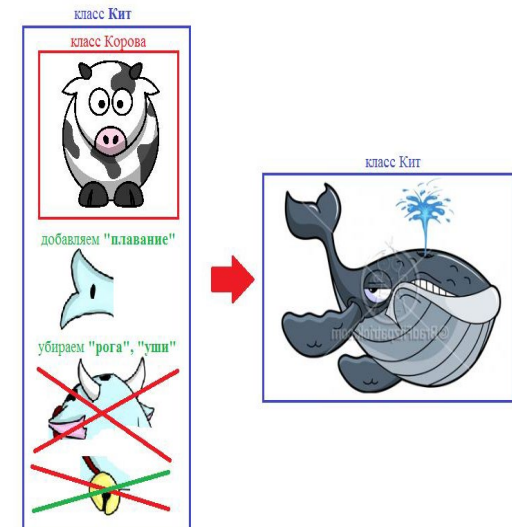
```
class Base {  
    void show() { System.out.println("Base"); }  
}  
class Derived extends Base {  
    @Override  
    void show() { System.out.println("Derived"); }  
}  
public class Demo1 {  
    public static void main(String[] args) {  
        Base base = new Base();  
        Derived obj = new Derived();  
        base.show();  
        obj.show();  
    }  
}
```

Вывод консоли:

Base

Derived

Переопределение
метода

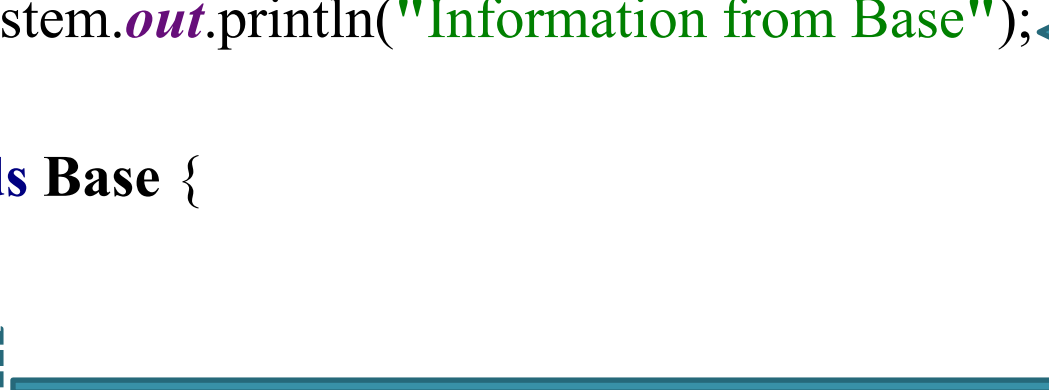


Переопределение методов

- ❑ Если необходимо вызвать метод суперкласса, а не переопределенный метод, то в подклассе используется ключевое слово **super**.

Пример 7:

```
class Base {  
    void show() { System.out.println("Information from Base");  
}  
}  
class Derived extends Base {  
    @Override  
    void show() {  
        super.show();  
        System.out.println("Information from Derived");  
}  
}  
public class Demo2 {  
    public static void main(String[] args) {  
        Derived obj = new Derived();  
        obj.show();  
}  
}
```



Вывод консоли:

```
Information from Base  
Information from Derived
```

При использовании ключевого слова **super** обращение всегда происходит к ближайшему суперклассу.

ЗАПРЕЩЕНИЕ ПЕРЕОПРЕДЕЛЕНИЯ МЕТОДА

- ❑ Для отмены возможности переопределения метода перед его описанием используется модификатор **final**



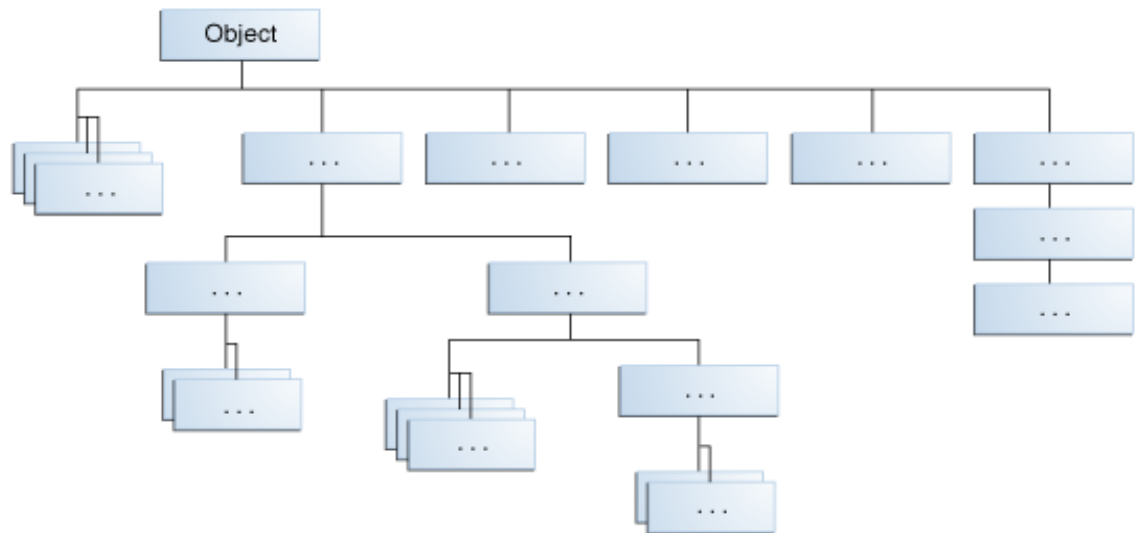
(компилятор допускает переопределение такого метода, однако во время выполнения возникнет «Ошибка исполнения»).



Класс Object и его методы

Класс `Object` и его методы

- ❑ В корне иерархии классов находится класс **`java.lang.Object`**, который является наиболее общим из всех классов.



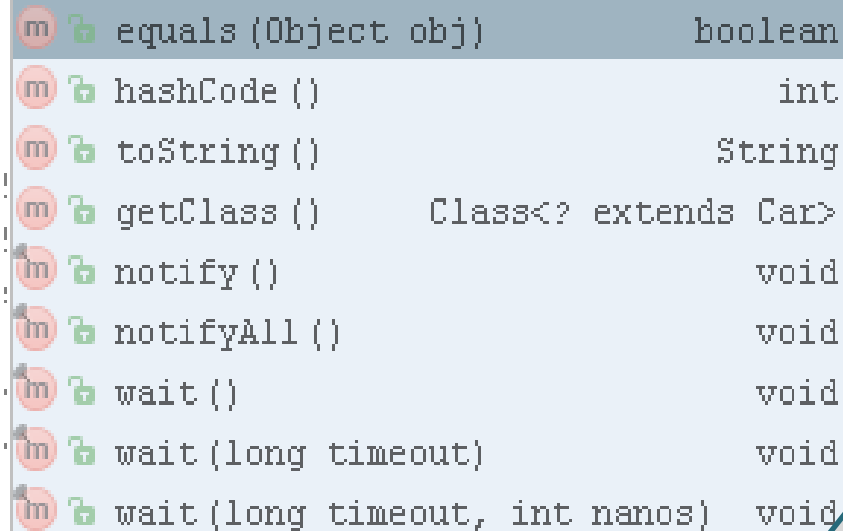
- ✓ Каждый класс имеет класс **`Object`** в качестве суперкласса (неявно).
- ✓ Все объекты, включая массивы, наследуют методы этого класса.

Класс Object и его методы

```
class CustomClass {  
}
```

Класс без методов и полей

```
public class Demo5 {  
    public static void main(String[] arg) {  
        CustomClass myClass = new CustomClass();  
        myClass.  
    }  
}
```



A screenshot of an IDE's method list for the CustomClass class. The list shows various methods, with the first three (equals, hashCode, toString) highlighted in blue. A blue arrow points from the text 'Унаследованные методы класса Object' to the list of methods.

Method	Return Type
equals (Object obj)	boolean
hashCode ()	int
toString ()	String
getClass ()	Class<? extends Car>
notify ()	void
notifyAll ()	void
wait ()	void
wait (long timeout)	void
wait (long timeout, int nanos)	void

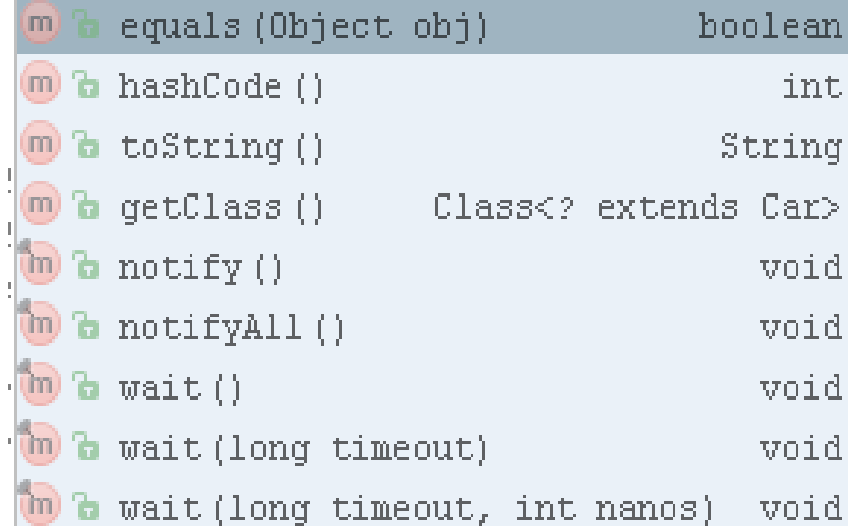
Унаследованные
методы класса
Object

Класс Object и его методы

```
class CustomClass extends Object {  
}
```

Явное наследование Object

```
public class Demo5 {  
    public static void main(String[] arg) {  
        CustomClass myClass = new CustomClass();  
        myClass.  
    }  
}
```



A screenshot of a Java IDE showing a list of methods inherited from the Object class. Each method is preceded by a red circle containing the letter 'm' and a green icon of a document. The methods and their return types are:

Method	Return Type
equals (Object obj)	boolean
hashCode ()	int
toString ()	String
getClass ()	Class<? extends Car>
notify ()	void
notifyAll ()	void
wait ()	void
wait (long timeout)	void
wait (long timeout, int nanos)	void

Унаследованные
методы класса
Object

Методы класса Object

Название	Описание
clone()	Создание клона объекта
equals(Object obj)	Сравнение двух объектов
finalize()	Выполнение завершающей работы перед уничтожением объекта сборщиком мусора
hashCode()	Генерация уникального идентификатора объекта
toString()	Возвращение строки символов с описанием объекта (автоматически вызывается методами <i>print()</i> и <i>println()</i>)
wait()	Перевод поток в состояние ожидания
notify()	Возобновление (уведомление) одного из потоков, вызвавших метод <i>wait()</i> на этом же объекте
notifyAll()	Возобновление (уведомление) всех потоков, вызвавших метод <i>wait()</i> на этом же объекте
getClass()	Возвращает описание класса объекта

1) Метод toString

Определение:

```
public String toString()
```

- ✓ Возвращает строковое представление объекта.
- Результат должен быть кратким, но информативным представлением объекта;
- Рекомендуется, чтобы все подклассы переопределяли этот метод;
- У класса **Object** этот метод возвращает строку, состоящую из имени класса, плюс символ "@" и беззнаковое шестнадцатеричное представление хэш-кода объекта:
`getClass().getName() + '@' + Integer.toHexString(hashCode())`

[illegible]

Класс Object и его методы

Пример 8: использование унаследованного метода

```
package com.knu.testtest;  
  
class CustomClass {  
  
public class Demo6 {  
    public static void main(String[] arg) {  
        CustomClass myClass = new CustomClass();  
        System.out.println(myClass);  
    }  
}
```

Вызов
унаследованного
метода *toString()*
класса **Object**

Вывод консоли :

com.knu.testtest.CustomClass@15e83f9

Пример 9: переопределение метода `toString()`:

```
package com.knu.testtest;  
class CustomClass {
```

```
    @Override
```

```
    public String toString(){  
        return "This is CustomClass";  
    }
```

```
}
```

```
public class Demo6 {
```

```
    public static void main(String[] arg) {  
        CustomClass myClass = new CustomClass();  
        System.out.println(myClass);  
    }
```

```
}
```

Вывод консоли:
This is CustomClass

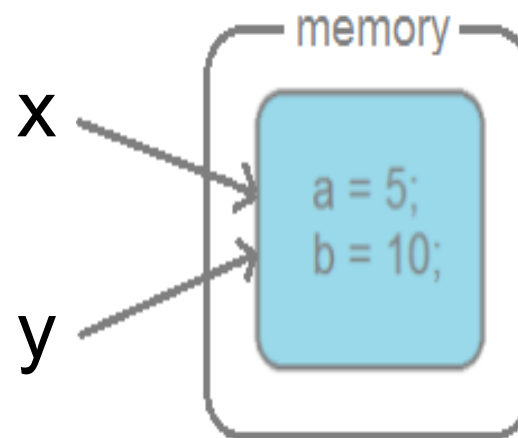
Вызов
переопределенного
метода `toString()`
класса **Object**

2) Метод equals

Определение:

```
public boolean equals(Object obj)
```

- ✓ Реализует отношение эквивалентности на двух объектах (т.е. на не нулевых ссылках).
- У класса **Object** этот метод возвращает **true**, если и только если ссылки **x** и **y** относятся к одному и тому же объекту ($x == y$).





Свойства эквивалентности

- ❑ *рефлексия*: для любого ненулевого значения ссылки **x**, **x.equals(x)** должно возвращать **true**;
- ❑ *симметричность*: для любых ненулевых значений ссылок **x** и **y**, **x.equals(y)** должно возвращать **true**, если и только если **y.equals(x)** возвращает **true**;
- ❑ *переносимость*: для любых ненулевых значений ссылок **x**, **y**, и **z** если **x.equals(y)** возвращает **true** и **y.equals(z)** возвращает **true**, тогда **x.equals(z)** должно вернуться **true**;
- ❑ *согласованность*: для любых ненулевых значений ссылок **x** и **y** многочисленные вызовы **x.equals(y)** последовательно возвращают **true** или **false**, если только нет информации об изменении объектов, используемых в сравнении;
- ❑ для любого ненулевого значения ссылки **x** **x.equals(null)** должно вернуть **false**.

Класс Object и его методы

Пример 10: переопределение метода equals()

```
class Point {  
    protected int x;  
    protected int y;  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    @Override  
    public boolean equals(Object obj) {  
        if (this == obj)  
            return true;  
        if (obj == null ||  
            this.getClass() != obj.getClass())  
            return false;  
        Point other = (Point) obj;  
        if (this.x != other.x)  
            return false;  
        return (this.y == other.y);  
    }  
}
```

Класс Object и его методы

Продолжение примера 10:

```
public class Demo7 {  
    public static void main(String[] arg) {  
        Point point_1 = new Point(5, -5);  
        Point point_2 = point_1;  
        Point point_3 = new Point(5, -5);  
        Point point_4 = new Point(5, 5);  
  
        System.out.println(point_1.equals(point_2));  
        System.out.println(point_1.equals(point_3));  
        System.out.println(point_1.equals(point_4));  
    }  
}
```

Копирование
ссылки

Создание
новых
объектов

Вывод консоли:

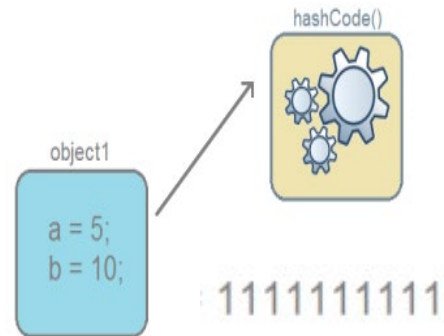
true
true
false

3) Метод hashCode

Определение:

```
public int hashCode()
```

- ✓ Возвращает значение хэш-кода объекта.

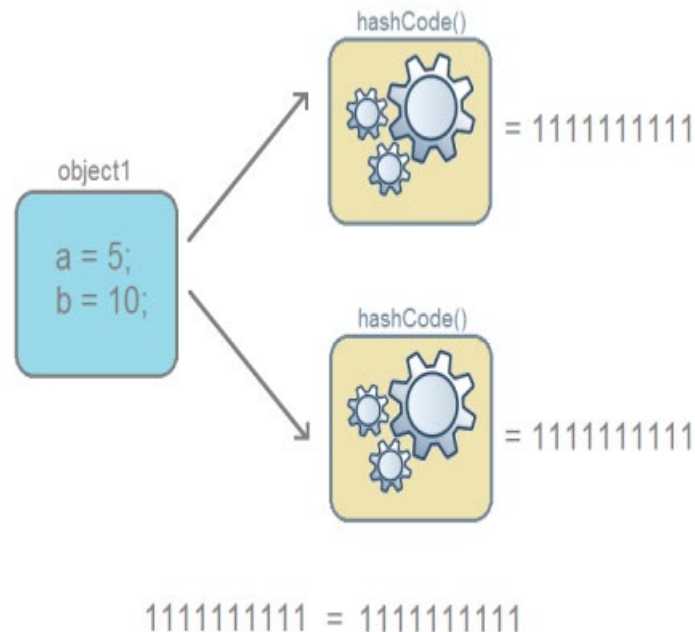


- *Хэш-функция* – это любая функция, которая может быть использована для отображения цифровых данных произвольного размера в цифровые данные фиксированного размера.
- Значения, возвращаемые хэш-функцией, называют: хэш-значения, хэш-коды, хэш-суммы или просто хэши.
- Этот метод поддерживается для работы коллекции, например **HashMap**.



Соглашения по хэш-коду (1/3)

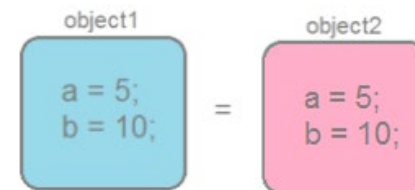
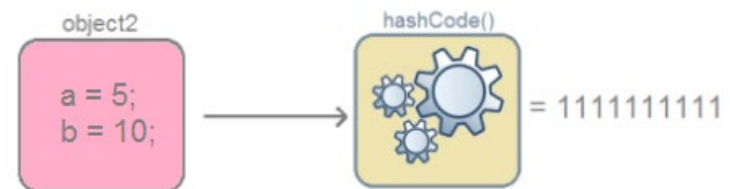
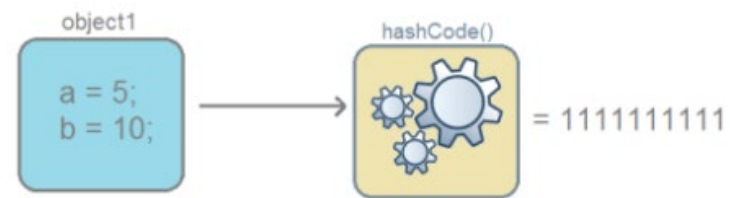
- ❑ Когда метод хэш-кода вызывается на том же объекте несколько раз во время исполнения Java-приложения, он должен возвращать тоже самое значение, если только не представлена информация об изменении объекта;





Соглашения по хэш-коду (2/3)

- ❑ Если два объекта равны в соответствии с методом *equals(Object)*, то при вызове метода хэш-кода на каждом из этих двух объектов, он должен выдавать один и тот же результат;



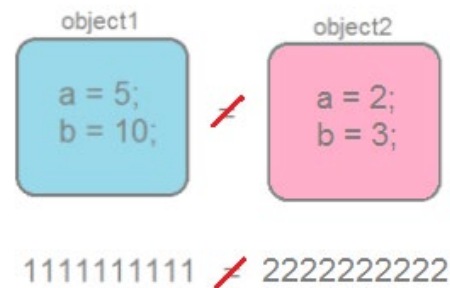
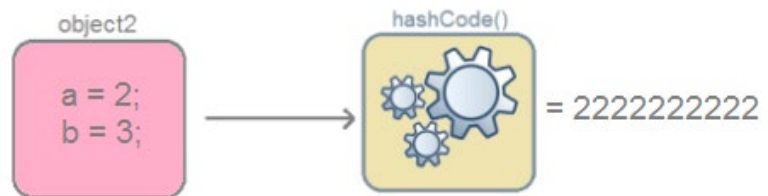
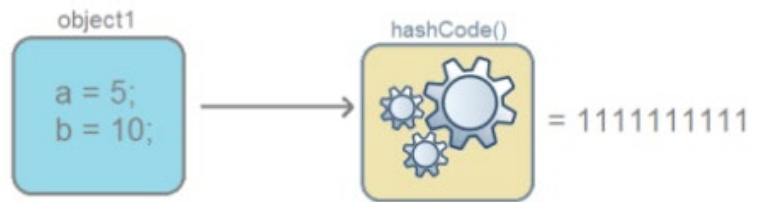
1111111111 = 1111111111

Класс Object и его методы



Соглашения по хэш-коду (3/3)

- ❑ Если два объекта неравны в соответствии с методом *equals(Object)*, то рекомендуется производить различные результаты.



Класс Object и его методы

- ❑ У класса **Object** этот метод вычисляет хэш-код посредством преобразования адреса размещения объекта в памяти в целое число.

Пример 11: использование класса Point из примера 10:

```
public class Demo8 {  
    public static void main(String[] arg) {  
        Point point_1 = new Point(5, -5);  
        Point point_2 = point_1;  
        Point point_3 = new Point(5, -5);  
        Point point_4 = new Point(5, 5);  
        System.out.println(point_1.hashCode());  
        System.out.println(point_2.hashCode());  
        System.out.println(point_3.hashCode());  
        System.out.println(point_4.hashCode());  
    }  
}
```

Вывод консоли:

```
1484678  
1484678  
22052786  
32487478
```



Рекомендации по вычислению хэш-кода (1/2)

- ❑ Сохранить некоторое ненулевое значение литерала, допустим 17, в переменной типа **int**, (*например, result*);
- ❑ Для каждого значимого поля *f* объекта (то поле, которое учитывается в методе сравнения), выполните следующие действия:
 - Вычислить хэш-код *c* типа **int** для каждого поля:
 - Если поле является **boolean**, вычислить **(f ? 1 : 0)**;
 - Если поле является **byte**, **char**, **short** или **int**, вычислить **(int)f**;
 - Если поле **long**, вычислить **(int) (f ^ (f >>> 32))**;
 - Если поле **float**, вычислить **Float.floatToIntBits(f)**;
 - Если поле **double**, вычислить **Double.doubleToLongBits(f)**, а затем как для **long**;



Рекомендаций по вычислению хэш-кода (2/2)

- Если поле является ссылкой на объект и метод *equals()* этого класса сравнивает поле рекурсивно, то рекурсивно вызывайте и метод хэш-кода на этом поле. Если значение поля равно **null**, то традиционно возвращают нулевое значение;
- Если поле является массивом, то рассматривают его так, если бы каждый элемент был отдельным полем (*вычислить хэш-код для каждого элемента с применением этих правил рекурсивно, и объединить эти значения*).

➤ Объедините вычисленные хэш-коды на каждом шаге в **result** следующим образом:

result = 31 * result + c;

❑ Возвращение результата.

Класс Object и его методы

Пример 12: переопределение метода hashCode()

```
public class Student {  
    private String name;  
    private long phone;  
    private int age;  
    // ...
```

@Override

```
public int hashCode() {  
    int result = 17;  
    result = 31 * result + name.hashCode();  
    result = 31 * result +  
        (int) (phone ^ (phone >>> 32));  
    result = 31 * result + age;  
    return result;  
}
```

Для ссылочных полей -
вызов их
переопределённых
методов hashCode()

Класс Object и его методы

Продолжение примера 12,

```
public class Demo9 {  
    public static void main(String[] arg) {  
        Student stud_1 =  
            new Student("Peter", 5558956L, 20);  
        Student stud_2 =  
            new Student("Ivan", 9876543L, 18);  
        Student stud_3 =  
            new Student("Dasha", 5558956L, 20);  
        Student stud_4 =  
            new Student("Ivan", 9876543L, 18);  
        System.out.println(stud_1.hashCode());  
        System.out.println(stud_2.hashCode());  
        System.out.println(stud_3.hashCode());  
        System.out.println(stud_4.hashCode());  
    }  
}
```

Вывод консоли:

```
1160475683  
-1786389060  
-1015000986  
-1786389060
```

Класс Object и его методы

ИНИЦИАЛИЗАЦИЯ С УЧЕТОМ НАСЛЕДОВАНИЯ

```
class SuperClass { }
```

```
class SubClass extends SuperClass { }
```

```
public class Main{  
    public static void main(String[] arg) {  
        SubClass c = new SubClass();  
    }  
}
```

