



Lesson 7



26.11.2020

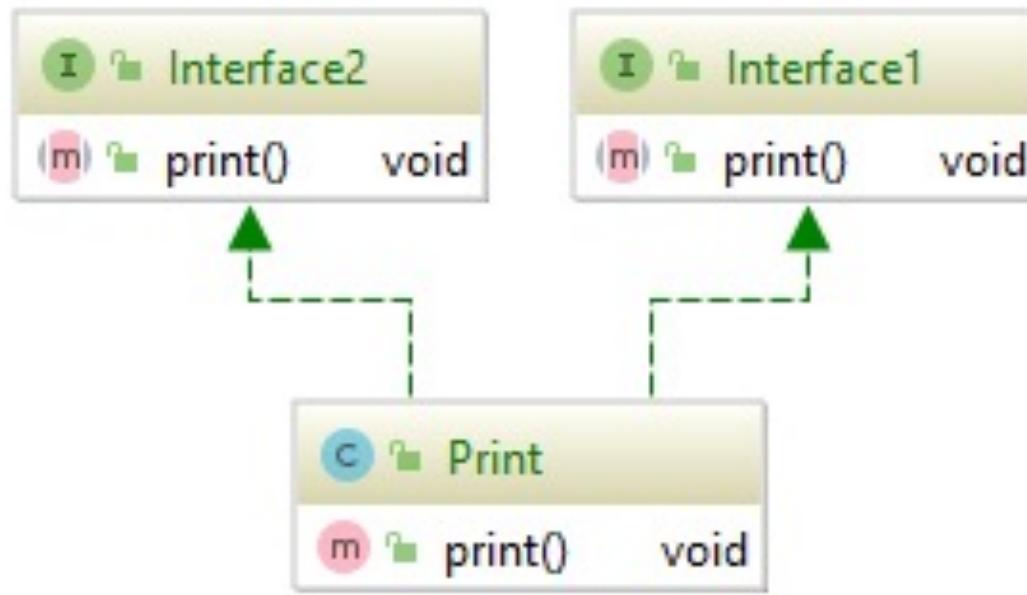
```
public class Ex1 {  
    public static void main(String[] args) {  
        do {  
            int y = 1;  
            System.out.println(y++ + " ");  
        } while (y <= 10);  
    }  
}
```

```
public class Ex2 {  
    public static void main(String[] args) {  
        boolean keepGoing = true;  
        int result = 15;  
        int i = 10;  
        do {  
            i--;  
            if (i == 8) keepGoing = false;  
            result -= 2;  
        } while (keepGoing);  
        System.out.println(result);  
    }  
}
```

```
public class Ex3 {  
    public static void main(String[] args) {  
        Foo foo = new Foo();  
        System.out.println(foo.a);  
        System.out.println(foo.b);  
        System.out.println(foo.c);  
    }  
}  
  
class Foo {  
    int a = 5;  
    protected int b = 6;  
    public int c = 7;  
}
```

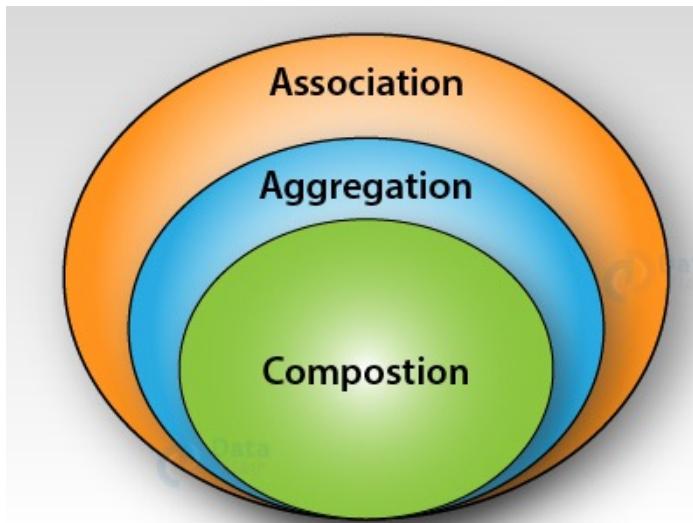
```
public class Ex5 {  
    public static void main(String[] args) {  
        int myGold = 7;  
        System.out.println(countGold, 6);  
    }  
}  
  
class Hobbit {  
    int countGold(int x, int y) {  
        return x + y;  
    }  
}
```

Diamond problem



В ООП выделяет три основных отношения между классами:

- Наследование
- Агрегация и композиция
- Ассоциация



Types of Association

IS-A

HAS-A

Aggregation

Composition



IS-A vs. HAS-A

IS-A

- Отношение “is a” – является.



HAS-A

- Отношение “has a” – содержит.

Машина

Радио



Ассоциация

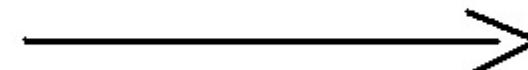
Ассоциация означает, что объекты двух классов могут ссылаться один на другой, иметь некоторую связь между друг другом.

```
public class Halter {}
```

```
public class Horse{  
    private Halter halter;  
}
```

Association

Class A



Class B

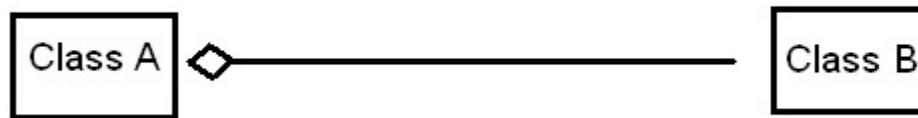


Агрегация

Агрегация - отношение когда один объект является частью другого.

```
public class Horse {  
    private Halter halter;  
  
    public Horse(Halter halter) {  
        this.halter = halter;  
    }  
}
```

Aggregation

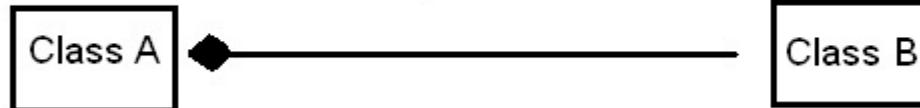


Композиция

Композиция - еще более тесная связь, когда объект не только является частью другого объекта, но и вообще не может принадлежать другому объекту.

```
public class Horse {  
    private Halter halter;  
  
    public Horse() {  
        this.halter = new Halter();  
    }  
}
```

Composition



Статическое и динамическое связывание

Существует два типа связывания методов в языке Java: ранее связывание (его ещё называют статическим) и позднее (соответственно, динамическое) **связывание**. Вызов метода в Java означает, что этот метод привязывается к конкретному коду или в момент компиляции, или во время выполнения, при запуске программы и создании объектов.

Можно понять из названия, статическое связывание носит более статический характер, так как происходит во время компиляции, то есть код «знает», какой метод вызывать после компиляции исходного кода на Java в файлы классов. А поскольку это относится к ранней стадии жизненного цикла программы, то называется также ранним связыванием (early binding). С другой стороны, динамическое связывание происходит во время выполнения, после запуска программы виртуальной машиной Java. В этом случае то, какой метод вызвать, определяется конкретным объектом, так что в момент компиляции информация недоступна, ведь объекты создаются во время выполнения. А поскольку это происходит на поздней стадии жизненного цикла программы, то называется в языке Java поздним связыванием (late binding).

Различия между ранним и поздним связыванием в языке Java

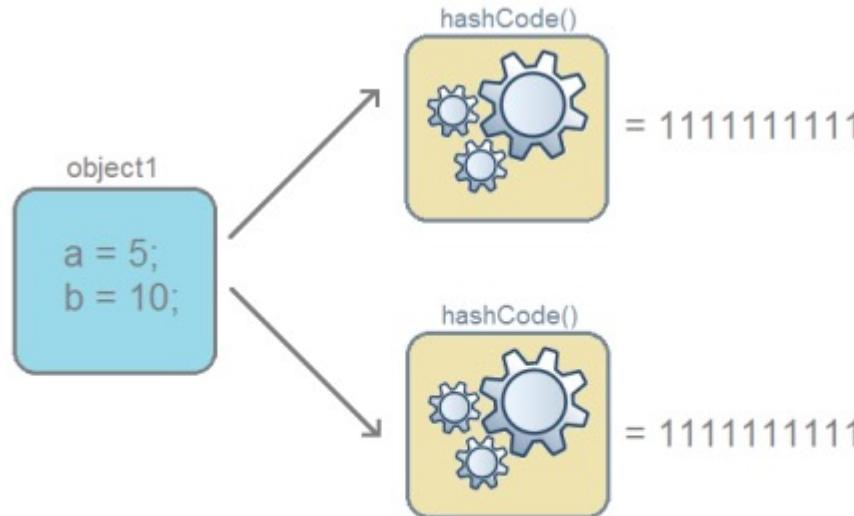
Теперь, когда вы разобрались и понимаете, как в языке Java связываются вызовы методов и как функционирует статическое и динамическое связывание, давайте еще раз перечислим ключевые различия между ранним и поздним связыванием в языке Java:

- Статическое связывание происходит во время компиляции, а динамическое – во время выполнения.
- Поскольку статическое связывание происходит на ранней стадии жизненного цикла программы, его называют ранним связыванием. Аналогично, динамическое связывание называют также поздним связыванием, поскольку оно происходит позже, во время работы программы.
- Статическое связывание используется в языке Java для разрешения перегруженных методов, в то время как динамическое связывание используется в языке Java для разрешения переопределенных методов.

- Аналогично, приватные, статические и терминальные методы разрешаются при помощи статического связывания, поскольку их нельзя переопределять, а все виртуальные методы разрешаются при помощи динамического связывания.
- В случае статического связывания используются не конкретные объекты, а информация о типе, то есть для обнаружения нужного метода используется тип ссылочной переменной. С другой стороны, при динамическом связывании для нахождения нужного метода в Java используется конкретный объект.

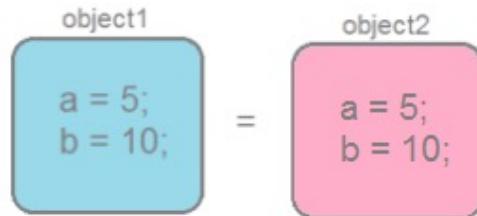
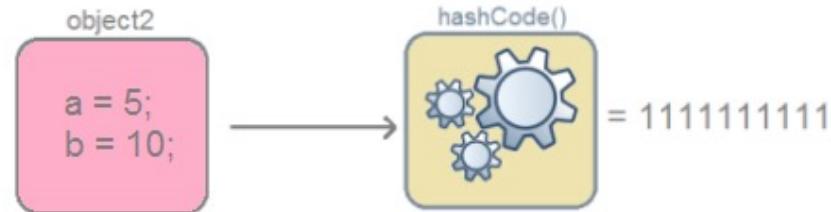
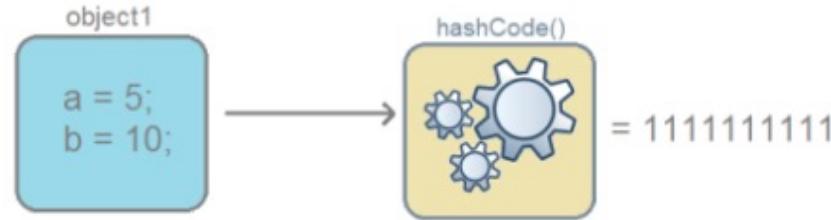
hashCode() и equals()

Для одного и того же объекта, хеш-код всегда будет одинаковым



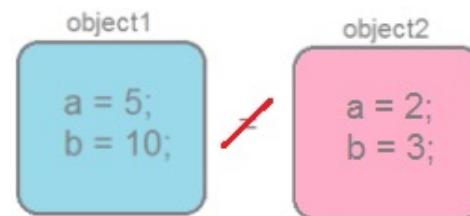
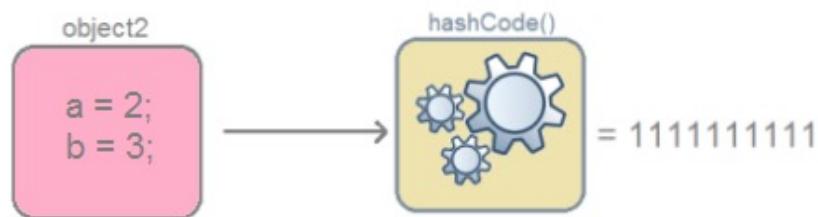
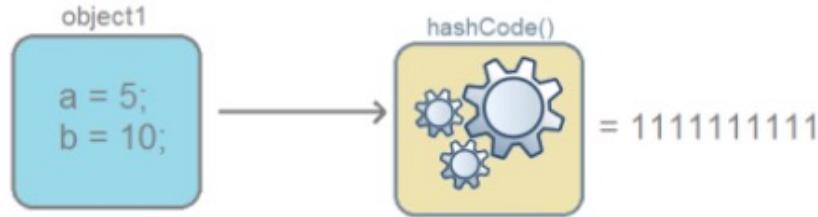
1111111111 = 1111111111

Если объекты одинаковые, то и хеш-коды одинаковые

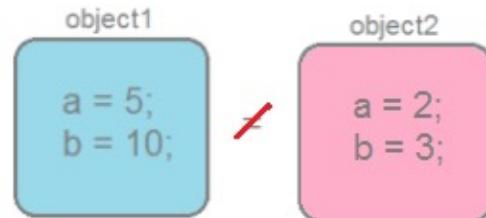
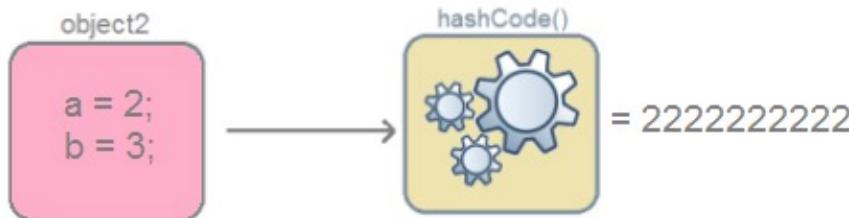
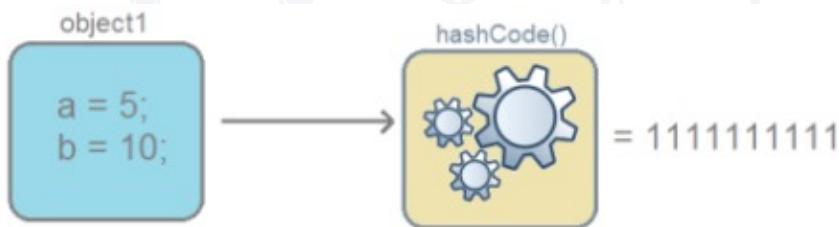


1111111111 = 1111111111

Если хеш-коды равны, то входные объекты не всегда равны (коллизия)



Если хеш-коды разные, то и объекты гарантированно разные



equals() - контракт

▪ Рефлексивность

для любого заданного значения x, выражение `x.equals(x)` должно возвращать `true`.
Заданного — имеется в виду такого, что `x != null`

▪ Симметричность

для любых заданных значений x и y, `x.equals(y)` должно возвращать `true` только в том случае, когда `y.equals(x)` возвращает `true`.

▪ Транзитивность

для любых заданных значений x, y и z, если `x.equals(y)` возвращает `true` и `y.equals(z)` возвращает `true`, `x.equals(z)` должно вернуть значение `true`.

▪ Согласованность

для любых заданных значений x и y повторный вызов `x.equals(y)` будет возвращать значение предыдущего вызова этого метода при условии, что поля, используемые для сравнения этих двух объектов, не изменились между вызовами.

▪ Сравнение null

для любого заданного значения x вызов `x.equals(null)` должен возвращать `false`.

Когда можно не переопределять этот метод

- Когда каждый экземпляр класса является уникальным. (Enum, Thread)
- Когда на самом деле от класса не требуется определять эквивалентность его экземпляров.

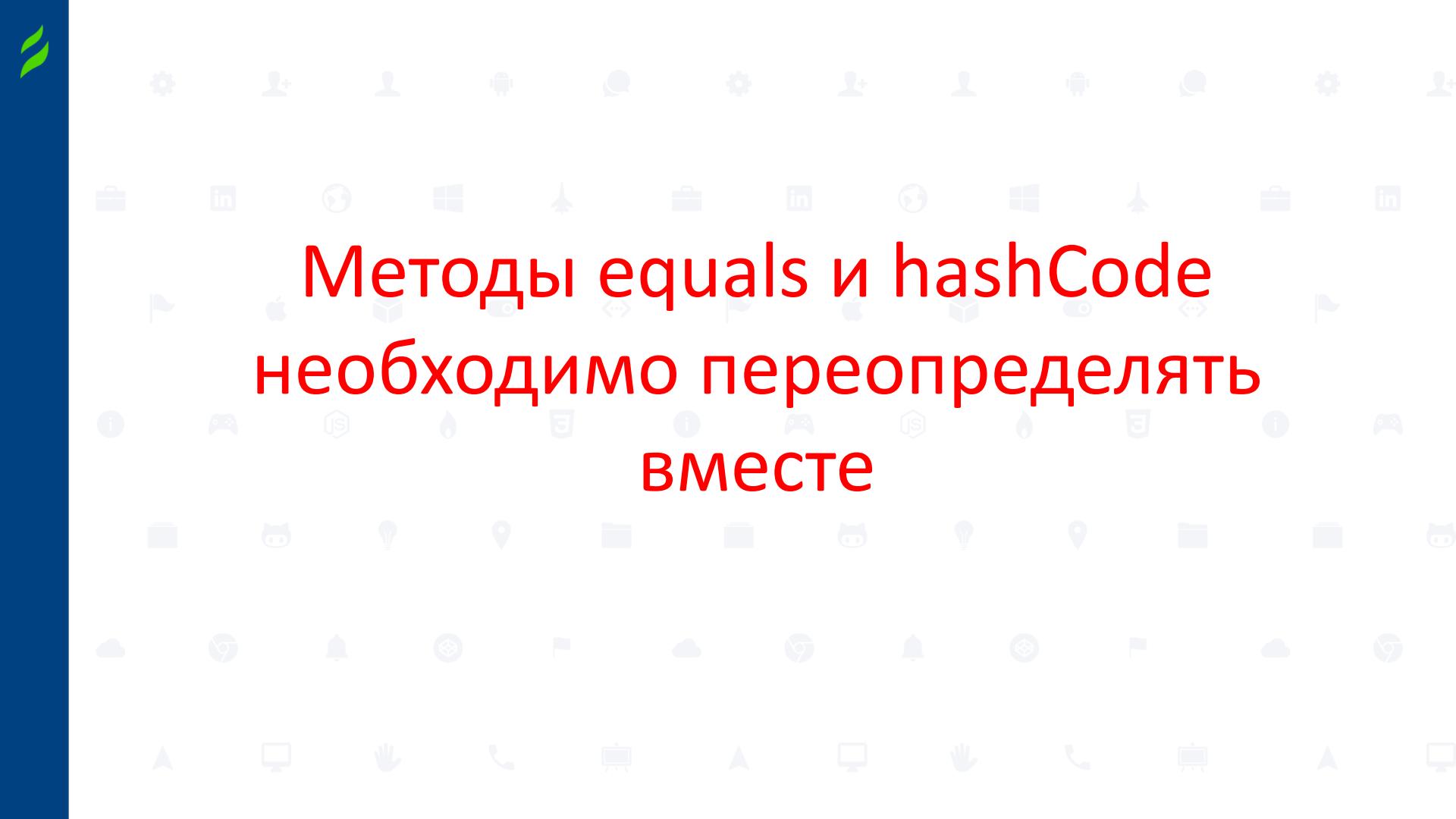
Например для класса `java.util.Random` вообще нет необходимости сравнивать между собой экземпляры класса, определяя, могут ли они вернуть одинаковую последовательность случайных чисел. Просто потому, что природа этого класса даже не подразумевает такое поведение.

- Когда класс, который вы расширяете, уже имеет свою реализацию метода `equals` и поведение этой реализации вас устраивает.
- Нет необходимости перекрывать `equals`, когда область видимости вашего класса является `private` или `package-private` и вы уверены, что этот метод никогда не будет вызван.



hashCode() - контракт

- вызов метода hashCode() один и более раз над одним и тем же объектом должен возвращать одно и то же хэш-значение, при условии что поля объекта, участвующие в вычислении значения, не изменились.
- вызов метода hashCode() над двумя объектами должен всегда возвращать одно и то же число, если эти объекты равны (вызов метода equals() для этих объектов возвращает true).
- вызов метода hashCode() над двумя неравными между собой объектами должен возвращать разные хэш-значения. Хотя это требование и не является обязательным, следует учитывать, что его выполнение положительно повлияет на производительность работы хэш-таблиц.



Методы equals и hashCode
необходимо переопределять
вместе

Enum

Кроме отдельных примитивных типов данных и классов в Java есть такой тип как **enum** или перечисление. Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую:

```
public enum UserStatus {  
    PENDING,  
    ACTIVE,  
    INACTIVE,  
    DELETED;  
}
```