





# Lesson 7

18.05.2023

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



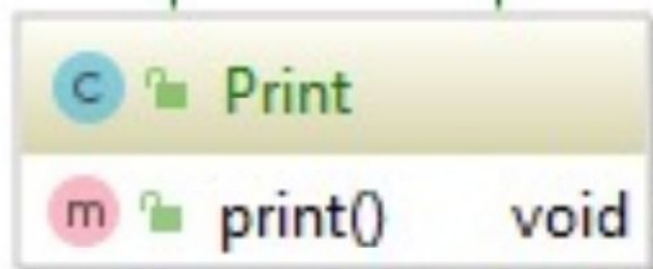
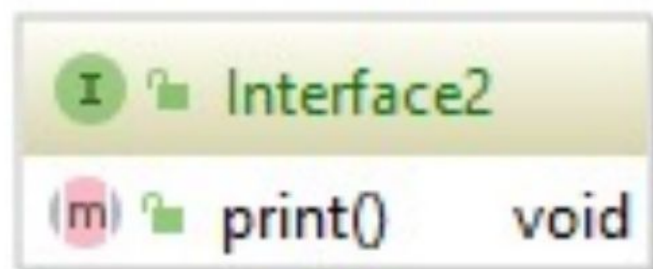
```
public class Ex2 {  
    public static void main(String[] args) {  
        boolean keepGoing = true;  
        int result = 15;  
        int i = 10;  
        do {  
            i--;  
            if (i == 8) keepGoing = false;  
            result -= 2;  
        } while (keepGoing);  
      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



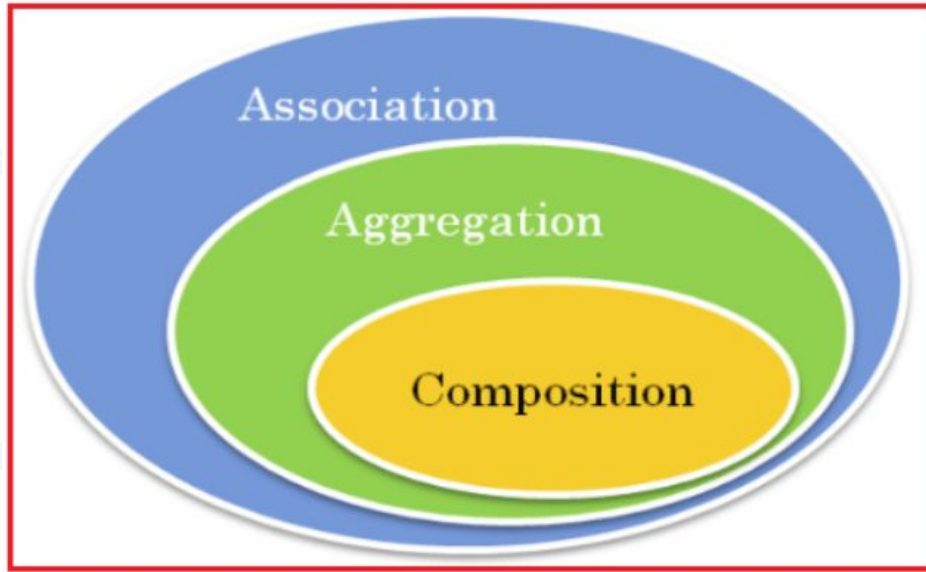
```
public class Ex3 {  
    public static void main(String[] args) { args) {  
        Foo foo = new Foo();  
        System.out.println(foo.a);  
        System.out.println(foo.b);  
        System.out.println(foo.c);  
    }  
}
```

```
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;  
    }  
}
```

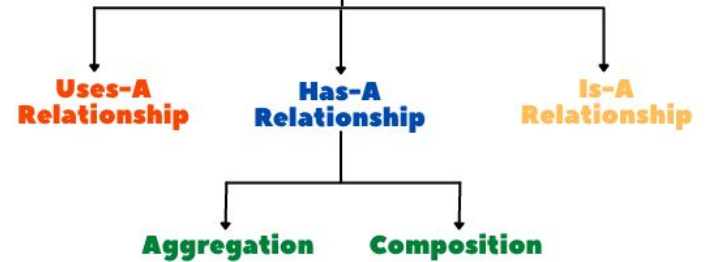


## Java association



### ***Types of***

#### **Class Relationships in Java**



```
class A {  
    .....  
}
```

**Uses-A  
Relationship**

```
class B  
{  
  
    void disp()  
    {  
        A obj=new A();  
        .....  
        .....  
    }  
}
```

```
class A {  
    .....  
}
```

**Has-A Relationship**

```
class B  
{  
    A obj=new A();  
    .....  
    .....  
}
```

```
class A {  
    .....  
}
```

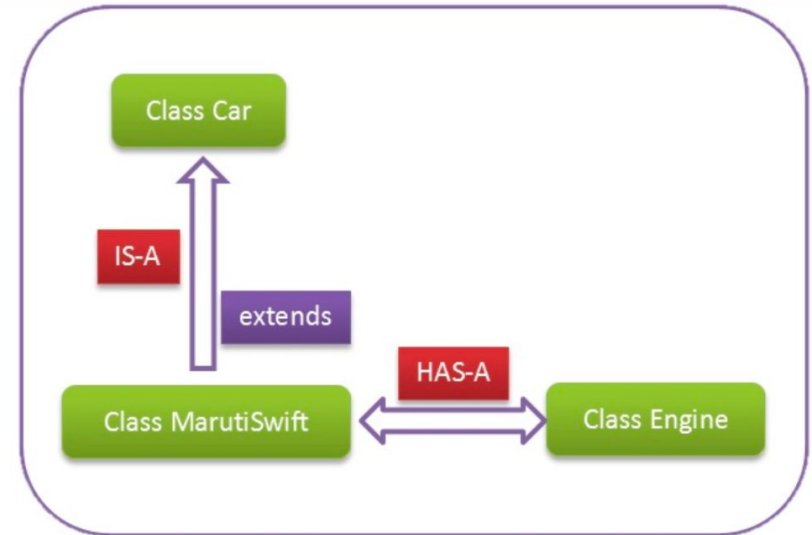
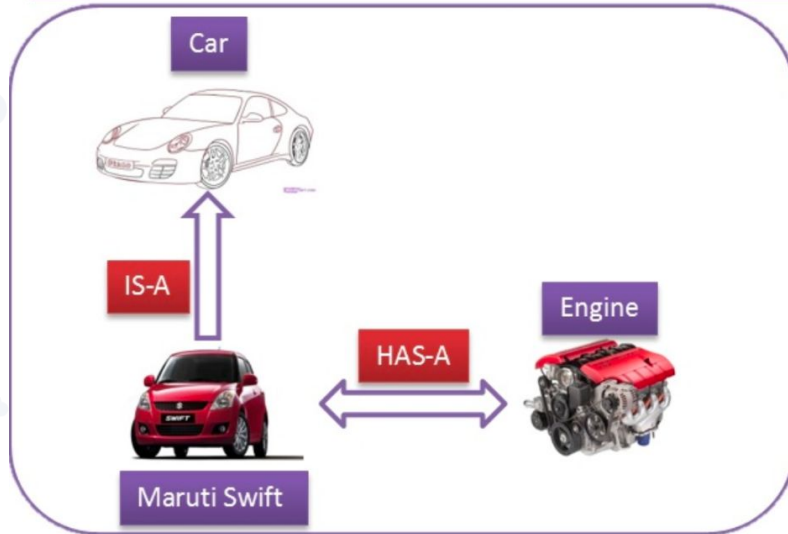
**Is-A Relationship**

```
class B extends A  
{  
    .....  
    .....  
}
```

**Fig: Different forms of relationship between classes in Java**



- ✓ Composition(HAS-A) simply mean use of instance variables that are references to other objects.  
For example: **MarutiSwift** Car has **Engine**, **House** has **Bathroom** , **Person** has **Address**.
- ✓ Has-A means an instance of one class “has a” reference to an instance of another class or another instance of same class.
- ✓ It is also known as “composition” or “aggregation”.
- ✓ There is no specific keyword to implement HAS-A relationship but mostly we are depended upon “**new**” keyword.
- ✓ Has-a relationship is composition relationship which is productive way of code reuse.

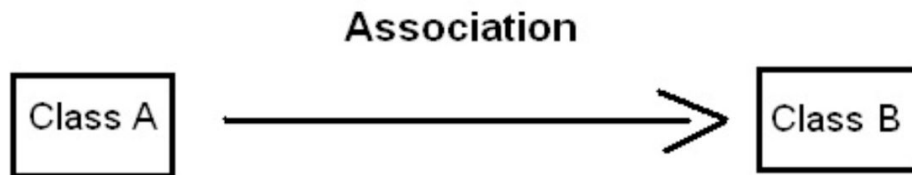


## Ассоциация

**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;  
}
```

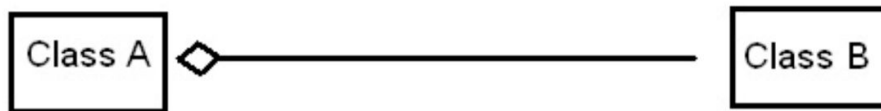


# Агрегация

Агрегация - отношение когда один объект является частью другого.

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

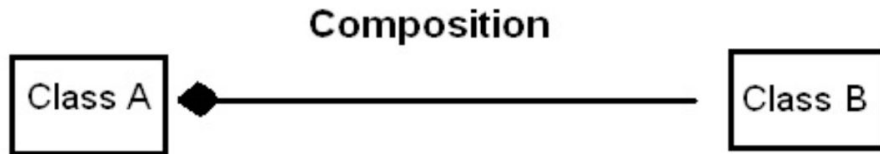
Aggregation



## Композиция

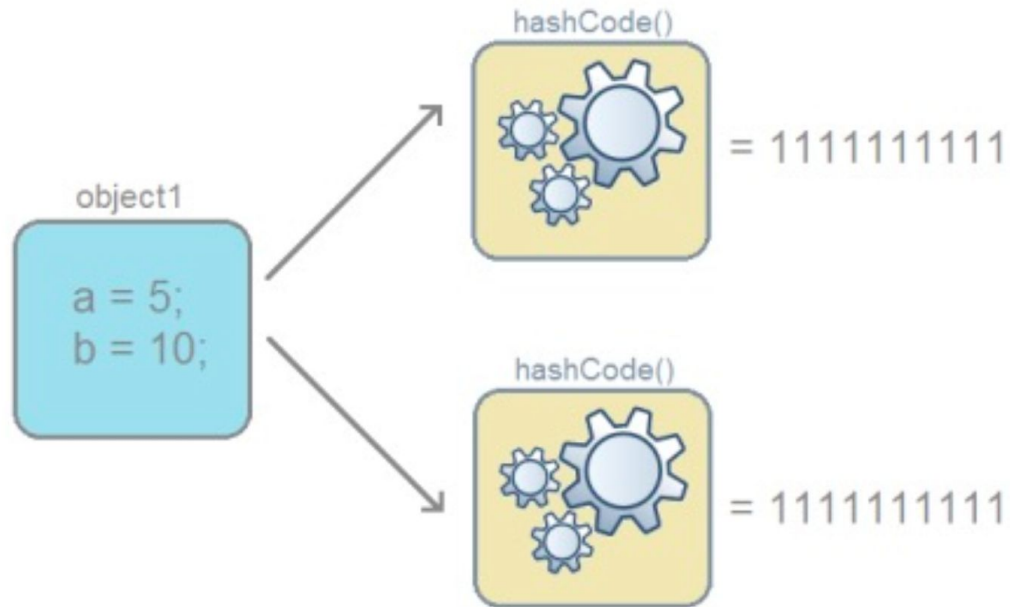
Композиция - еще более тесная связь, когда объект не только является частью другого объекта, но и вообще не может принадлежать другому объекту.

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



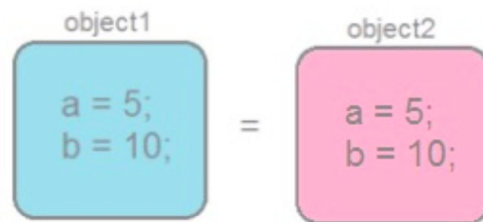
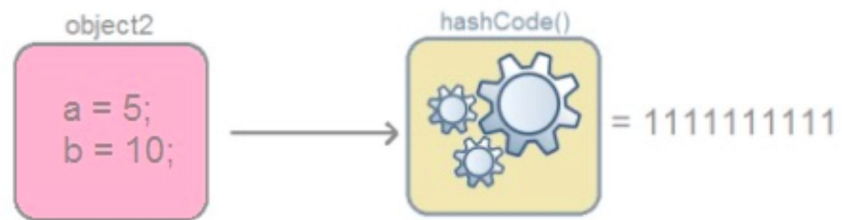
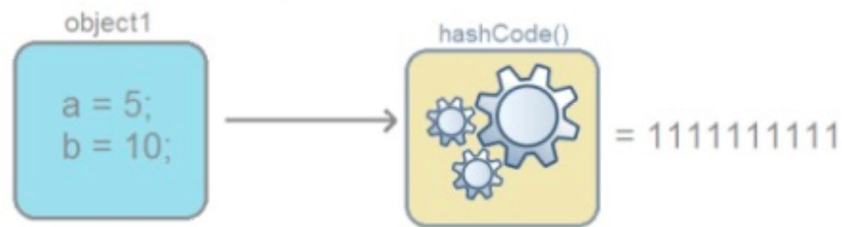
## hashCode() та equals()

Для одного і того ж об'єкта, хеш-код завжди буде однаковим



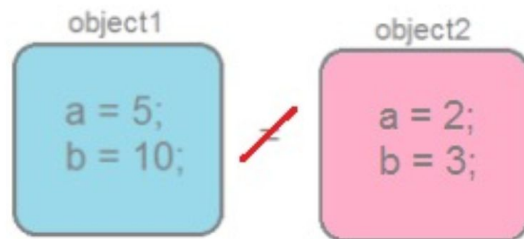
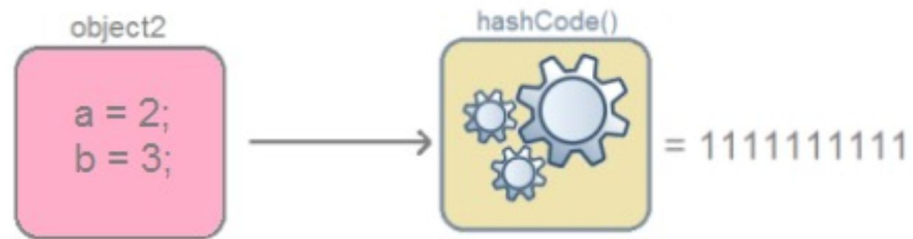
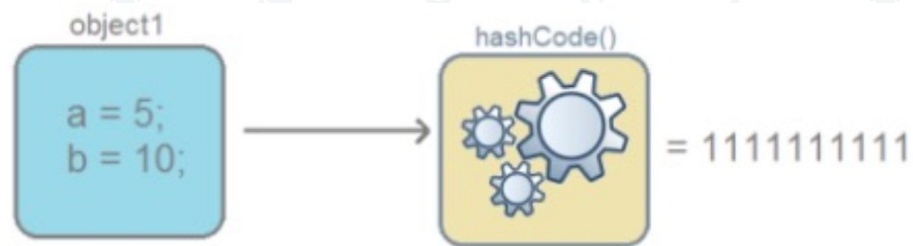
1111111111 = 1111111111

Якщо об'єкти однакові, то і хеш-коди однакові



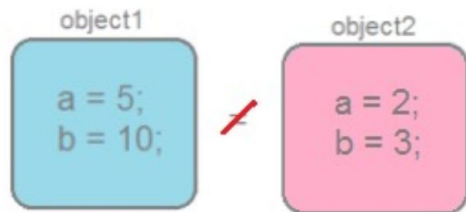
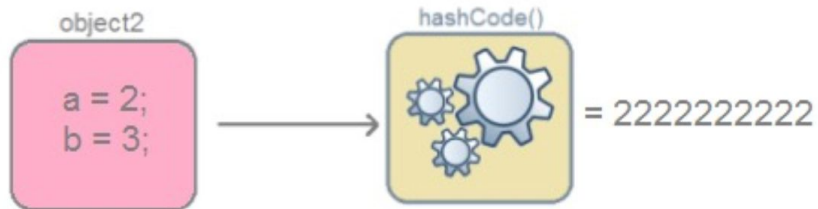
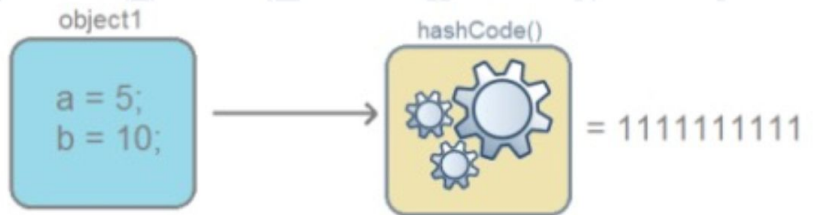
1111111111 = 1111111111

Якщо хеш-коди рівні, то вхідні об'єкти не завжди рівні (колізія)





Якщо хеш-коди різні, то об'єкти гарантовано різні.



1111111111  2222222222





# **equals() - контракт**

## **1 - Рефлексивність**

для будь-якого заданого значення `x` вираз `x.equals(x)` повинен повертати `true`. Заданого - мається на увазі такого, що `x != null`

## **2 - Симетричність**

для будь-яких заданих значень `x` та `y`, `x.equals(y)` має повертати `true` тільки в тому випадку, коли `y.equals(x)` повертає `true`.

## **3 - Транзитивність**

для будь-яких заданих значень `x`, `y` та `z`, якщо `x.equals(y)` повертає `true` і `y.equals(z)` повертає `true`, `x.equals(z)` має повернути значення `true`.

## **4 - Узгодженість**

для будь-яких заданих значень `x` та `y` повторний виклик `x.equals(y)` буде повертати значення попереднього виклику цього методу за умови, що поля, які використовуються порівняння цих двох об'єктів, не змінювалися між викликами.

## **5 - Порівняння null**

для будь-якого заданого значення `x` виклик `x.equals(null)` має повертати `false`



## Коли можна не перевизначати цей метод

1 - Коли кожен екземпляр класу є унікальним. (Enum, Thread)

2 - Коли насправді від класу не потрібно визначати еквівалентність його екземплярів.

Наприклад, для класу `java.util.Random` взагалі немає необхідності порівнювати між собою екземпляри класу, визначаючи, чи можуть вони повернути однакову послідовність випадкових чисел. Просто тому, що природа цього класу навіть не має на увазі таку поведінку.

3 - Коли клас, який ви розширюєте, вже має свою реалізацію методу `equals` та поведінка цієї реалізації вас влаштовує.

4 - Немає необхідності перекривати `equals`, коли область видимості вашого класу є `private` або `package-private` і ви впевнені, що цей метод ніколи не буде викликано



## hashCode() - контракт

1 - виклик методу hashCode() один і більше разів над тим самим об'єктом повинен повертати те саме хеш-значення, за умови що поля об'єкта, значення, що беруть участь у обчисленні, не змінювалися.

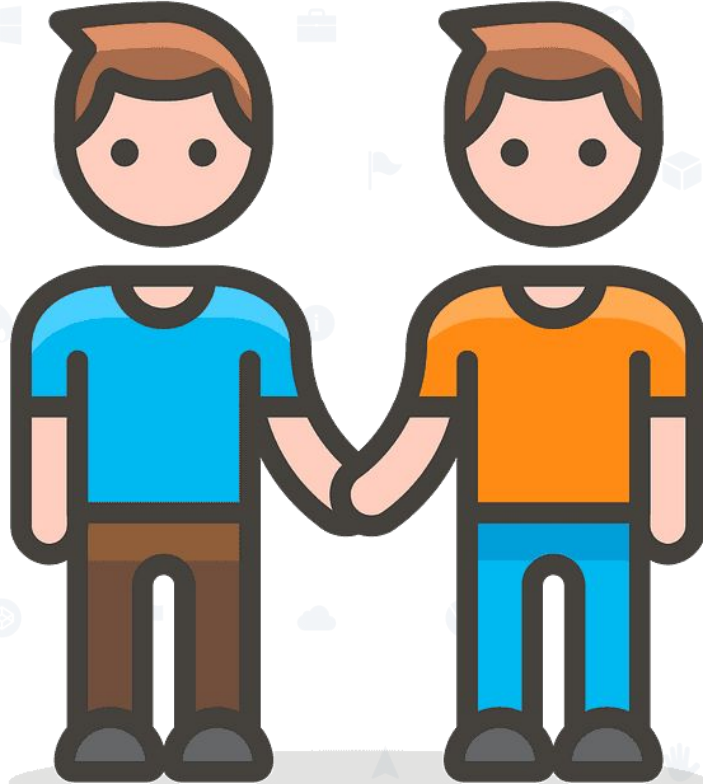
2 - виклик методу hashCode() над двома об'єктами повинен завжди повертати те саме число, якщо ці об'єкти рівні (виклик методу equals() для цих об'єктів повертає true).

3 - виклик методу hashCode() над двома нерівними між собою об'єктами повинен повертати різні хеш-значення. Хоча ця вимога і не є обов'язковим слід враховувати, що його виконання позитивно вплине на продуктивність роботи хеш-таблиць.



equals

hashCode





## Enum

Крім окремих примітивних типів даних та класів у Java є такий тип як enum чи перерахування. Переліки представляють набір логічно пов'язаних констант. Оголошення перерахування відбувається за допомогою оператора enum, після якого йде назва перерахування. Потім іде список елементів перерахування через кому:

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