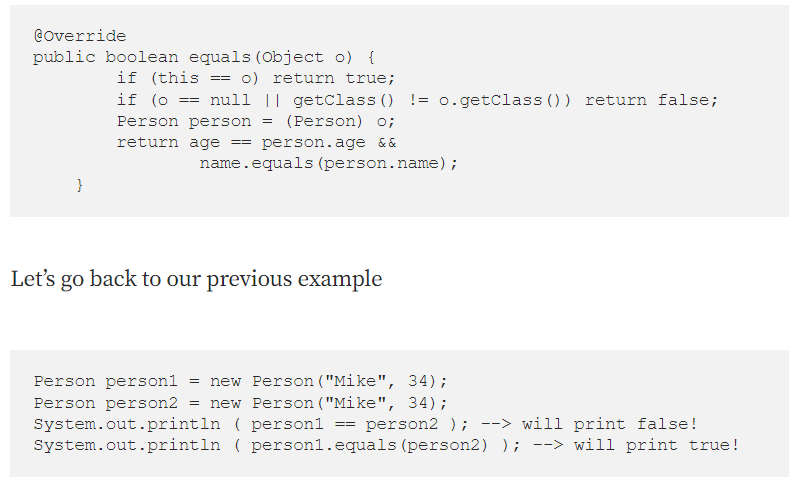
==

* The operator resolves true only if 2 references represent the same instance in memory
* Person person1 = new Person("Mike", 34);  
  Person person2 = new Person("Mike", 34);  
  System.out.println( person1 == person2 ); --> will print false!
* How to make them equal. There comes equals

Equals()

* We override equals() tocheck if some objects have same values for specific fields to be considered equal.
* Select which fields to be compared. 2 Person object will be same if only if they have same name and same age
* 
* **Keep in mind** however, if we don’t provide our custom version of .equals() (a.k.a override) in our class then the predefined .equals() from Object class and == operator will behave exactly the same.
* Default equals() method which is inherited from Object will check whether both compared instances are the same in memory!

Hashcode()

* Default implementation of hashcode() which is inherited from Object considers all objects in memory unique.
* HashSet can not contain duplicate values and HashMap can not contain duplicate keys.
* HashSet is implemented with a HashMap behind the scenes where each value of a HashSet is stored as a key in a HashMap.
* HashMap locates the correct linkedList for each key by applying hashCode() method and after that it iterates through all elements of that linkedList and applies equals() method on each of these elements to check if that element is already contained there. **No duplicate keys are allowed.**
* By default hashCode() method returns a different result for each different instance. If we have the default equals() which behaves like == which considers all instances in memory as different objects we don't have any problem.

But in our previous example we said we want Person instances to be considered equal if their ages and names match.

Person person1 = new Person("Mike", 34);  
 Person person2 = new Person("Mike", 34);  
 System.out.println ( person1.equals(person2) ); --> will print true!

Now let’s create a map to store those instances as keys with some string as pair value

Map<Person, String> map = new HashMap();  
map.put(person1, "1");  
map.put(person2, "2");

In Person class we have not overridden the hashCode method but we have overridden equals method. Since the default hashCode provides different results for different java instances person1.hashCode() and person2.hashCode() have big chances of having different results.

Our map might end with those persons in different linkedLists.

A picture containing diagram

Description automatically generated

This is against the logic of a HashMap.

**A HashMap is not allowed to have multiple equal keys!**

But ours now has and the reason is that the default hashCode() which was inherited from Object Class was not enough. Not after we have overridden the equals() method on Person Class.

That is the reason why we must override hashCode() method after we have overridden equals method.

Now let’s fix that. Let’s override our hashCode() method to consider the same fields that equals() considers, namely age, name

public class Person {  
 private Integer age;  
 private String name;  
   
 ..getters, setters, constructors  
@Override  
public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Person person = (Person) o;  
 return age == person.age &&  
 name.equals(person.name);  
 }  
@Override  
public int hashCode() {  
 int prime = 31;  
 return prime\*Objects.hash(name, age);  
 }

PS: In the hashCode() method, we used a prime value (you can use any other values). However, it is suggested to use prime numbers as this in order to produce **less collisions.**

Now let’s try again to save those keys in our HashMap:

Map<Person, String> map = new HashMap();  
map.put(person1, "1");  
map.put(person2, "2");

person1.hashCode() and person2.hashCode() will definitely be the same. Let's say it is 0.

HashMap will go to bucket 0 and in that LinkedList will save the person1 as key with the value “1”. For the second put HashMap is intelligent enough and when it goes again to bucket 0 to save person2 key with value “2” it will see that another equal key already exists there.

So it will overwrite the previous key. So in the end only person2 key will exist in our HashMap.

A picture containing square

Description automatically generated

Now we are aligned with the rule of Hash Map that says no multiple equal keys are allowed!

Keep in mind however, un-equal instances may have same hashcode and equal instances should return same hashcode.

11

11