The Importance of Overriding hashCode() and equals() in Java for Correct Behavior in Collections

**Original Code:**

import java.util.LinkedHashSet;

import java.util.Set;

public class TestNumPOJO {

public static void main(String[] args) {

Set<NumPOJO> listkedHashSet1 = new LinkedHashSet<>();

listkedHashSet1.add(new NumPOJO(100, 0.5));

listkedHashSet1.add(new NumPOJO(100, 0.5));

listkedHashSet1.add(new NumPOJO(500, 0.2));

listkedHashSet1.add(new NumPOJO(1000, 0.1));

System.out.println("NumPOJO :" + listkedHashSet1);

double sum1 = listkedHashSet1.stream()

.mapToDouble((NumPOJO numPOJO) -> numPOJO.getCash() \* numPOJO.getRate())

.sum();

System.out.println("NumPOJO :" + sum1);

Set<NumNonPOJOWithoutHashCode> listkedHashSet2 = new LinkedHashSet<>();

listkedHashSet2.add(new NumNonPOJOWithoutHashCode(100, 0.5));

listkedHashSet2.add(new NumNonPOJOWithoutHashCode(100, 0.5));

listkedHashSet2.add(new NumNonPOJOWithoutHashCode(500, 0.2));

listkedHashSet2.add(new NumNonPOJOWithoutHashCode(1000, 0.1));

System.out.println("NumNonPOJOWithoutHashCode: " + listkedHashSet2);

double sum2 = listkedHashSet2.stream()

.mapToDouble((NumNonPOJOWithoutHashCode numNonPOJOWithoutHashCode) -> numNonPOJOWithoutHashCode.getCash() \* numNonPOJOWithoutHashCode.getRate())

.sum();

System.out.println("NumNonPOJOWithoutHashCode: " + sum2);

}

}

public class NumPOJO {

private int cash;

private double rate;

public NumPOJO() {

super();

}

public NumPOJO(int cash, double rate) {

super();

this.cash = cash;

this.rate = rate;

}

@Override

public int hashCode() {

int hash = 3;

hash = 67 \* hash + this.cash;

hash = 67 \* hash + (int) (Double.doubleToLongBits(this.rate) ^ (Double.doubleToLongBits(this.rate) >>> 32));

return hash;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

NumPOJO other = (NumPOJO) obj;

return cash == other.cash && Double.doubleToLongBits(rate) == Double.doubleToLongBits(other.rate);

}

public int getCash() {

return cash;

}

public void setCash(int cash) {

this.cash = cash;

}

public double getRate() {

return rate;

}

public void setRate(double rate) {

this.rate = rate;

}

@Override

public String toString() {

return "num [cash=" + cash + ", rate=" + rate + "]";

}

}

public class NumNonPOJOWithoutHashCode {

private int cash;

private double rate;

public NumNonPOJOWithoutHashCode() {

super();

}

public NumNonPOJOWithoutHashCode(int cash, double rate) {

super();

this.cash = cash;

this.rate = rate;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

NumNonPOJOWithoutHashCode other = (NumNonPOJOWithoutHashCode) obj;

return cash == other.cash && Double.doubleToLongBits(rate) == Double.doubleToLongBits(other.rate);

}

public int getCash() {

return cash;

}

public void setCash(int cash) {

this.cash = cash;

}

public double getRate() {

return rate;

}

public void setRate(double rate) {

this.rate = rate;

}

@Override

public String toString() {

return "num [cash=" + cash + ", rate=" + rate + "]";

}

}

**Response & Explanation:**

1. **NumPOJO Class (with hashCode overridden)**: Since the NumPOJO class has the hashCode() method overridden, it ensures that objects with the same cash and rate values are treated as duplicates. When these objects are added to a Set like LinkedHashSet, duplicates are not allowed, and only unique objects are stored.

Expected output for NumPOJO:

NumPOJO :[num [cash=100, rate=0.5], num [cash=500, rate=0.2], num [cash=1000, rate=0.1]]  
NumPOJO :250.0

Here, only the unique combinations of cash and rate values are retained in the Set.

1. **NumNonPOJOWithoutHashCode Class (without hashCode)**: The NumNonPOJOWithoutHashCode class does **not** override hashCode(). This results in the default Object.hashCode() behavior, which assigns different hash codes for each instance, even if the cash and rate values are identical. Therefore, when adding these objects to a LinkedHashSet, all objects (even those that should be considered duplicates) are treated as unique.

Expected output for NumNonPOJOWithoutHashCode:

NumNonPOJOWithoutHashCode: [num [cash=100, rate=0.5], num [cash=100, rate=0.5], num [cash=500, rate=0.2], num [cash=1000, rate=0.1]]  
NumNonPOJOWithoutHashCode: 300.0  
  
In this case, even though the first two objects have identical values for cash and rate, they are added separately to the set, violating the Set contract of uniqueness.

**Corrected Code for NumNonPOJOWithoutHashCode:**

To fix the behavior, you can override the hashCode() method in NumNonPOJOWithoutHashCode as well:

@Override

public int hashCode() {

int hash = 3;

hash = 67 \* hash + this.cash;

hash = 67 \* hash + (int) (Double.doubleToLongBits(this.rate) ^ (Double.doubleToLongBits(this.rate) >>> 32));

return hash;

}

After applying this change, the output for NumNonPOJOWithoutHashCode would be similar to NumPOJO, ensuring that duplicate objects are not added to the set:

NumNonPOJOWithoutHashCode: [num [cash=100, rate=0.5], num [cash=500, rate=0.2], num [cash=1000, rate=0.1]]

NumNonPOJOWithoutHashCode: 250.0