Explanation of equals() and hashCode() Methods

# 1. Explanation of the line if (getClass() != obj.getClass()) return false;

This line is part of the equals method, which is used to compare two objects for equality.  
  
- Purpose: The if (getClass() != obj.getClass()) statement checks if the two objects being compared are of the same class. If they are not, it returns false, meaning they are not equal.  
  
- Reason: By default, the equals method checks reference equality (i.e., whether the two objects are the exact same object in memory).However, if we want to override it to compare the contents of two different objects, we need to ensure they are of the same type (class).This line prevents comparing objects of different types (classes) and returning true for such comparisons.

For example:  
  
OverrideEqualsHashCode obj1 = new OverrideEqualsHashCode("Red", 100);  
SomeOtherClass obj2 = new SomeOtherClass();  
  
If obj1.equals(obj2) is called, this line will ensure that objects of different types, like OverrideEqualsHashCode and SomeOtherClass, cannot be considered equal.

# 2. Why We Need hashCode in the Class

In Java, if you override the equals method, it is mandatory to override the hashCode method.This is because of how Java's hashCode() and equals() methods are related, especially in collections like HashMap, HashSet, and Hashtable,which use hashing to store and retrieve objects efficiently.

- Why hashCode is important:  
The hashCode method generates an integer (hash value) based on the object's properties (in this case, color and price).When objects are added to hash-based collections, their hashCode is used to find the bucket where they will be stored.When searching for an object in a hash-based collection, Java first compares hash codes. If the hash codes match, it then uses the equals method to check if the objects are truly equal.

- Consistency between equals and hashCode:  
Two objects that are considered equal according to the equals method must have the same hashCode.If you do not override hashCode when overriding equals, you may face issues with object comparison and performance in hash-based collections.

# 3. Overview of hashCode Implementations:

1. First Implementation:  
@Override  
public int hashCode() {  
 int hash = 5;  
 hash = 17 \* hash + Objects.hashCode(this.color);  
 hash = 17 \* hash + this.price;  
 return hash;  
}

2. Second Implementation:  
@Override  
public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + Objects.hash(color, price);  
 return result;  
}

3. Third Implementation:  
@Override  
public int hashCode() {  
 int hash = 7;  
 hash = 31 \* hash + (int) price;  
 hash = 31 \* hash + (color == null ? 0 : color.hashCode());  
 return hash;  
}

# 4. Final Notes on equals and hashCode:

The equals method compares the contents (color and price) of two OverrideEqualsHashCode objects.The hashCode method generates a hash value based on the same properties. This ensures that if two objects are equal according to equals(), they will also have the same hash code,which is critical for collections like HashMap and HashSet.

The recommendation of the book 'Effective Java' by Joshua Bloch is relevant here because it covers best practices for implementing equals and hashCode, including dealing with hash collisions and efficient algorithms.