Programming 1

Christian Grévisse (christian.grevisse@uni.lu)
Gilles Neyens (gilles.neyens@uni.lu)

Lab 8 – Polymorphism, Generics & equals/hashCode

2017–11–28

Exercise 1 – File System #Inheritance #programming:Polymorphism

Consider a file system where you have different files like JPG images or MP3 files, as well as folders. On Moodle, you can find an archive with boilerplate implementations for JPG, BMP and MP3 files.

Extend this code by introducing folders. Those can contain arbitrary files as well as other folders again. Hence, a folder requires a method to add items.

Add a print method to your classes that enables printing of the names of all files contained in a folder and its subfolders (similar to the output of the `tree` utility on Linux).

Finally, write a main program that tests your implementation.

Hint: The *composite pattern* might come handy.

Exercise 2 – Music Player #GenericType #GenericParameter #equals

Develop a small music player program which manages songs and playlists.

1° A song has a title, an artist, a duration and a play count. Each time a song gets played, the play count gets incremented and the metadata of the song are shown.

2° A playlist has a name and holds the contained songs in a `LinkedHashSet`, a data structure which avoids duplicates and enables iteration using the order of element insertion (as opposed to regular `HashSet`s, which do not define any ordering). Make sure that duplicates are properly recognized as such through an appropriate implementation of the `equals` and `hashCode` methods at the level of songs.

Furthermore, it shall be possible to add songs to a playlist, get the total duration of the list and play all songs in the playlist.

3° Finally, write a main program where you create some songs and playlists.

Exercise 3 – Supermarket Loyalty Card #GenericType #GenericParameter #equals

A supermarket gives loyalty cards to its customers. Every card has an ID and a number of accumulated loyalty points. Each time a customer buys at the supermarket, 10% of the total amount are added as points to the card. It is also possible to redeem an amount by paying with the points on the loyalty card.

The supermarket gives several cards to a customer, which can be used by her family members. The ID of the different cards of a same customer account are equal. Two loyalty cards are considered equal if their IDs are the same. For simplicity, you can assume that accumulated points are not shared among the different cards of a customer.

The supermarket holds the information of all cards that have redeemed on a certain date. For that, you might use a data structure such as `Map<LocalDate, List<LoyaltyCard>>`, which maps a date to a list of loyalty cards (the current day can be determined with `LocalDate.now()`).

A redeem action is only successful if a card has not yet been used to redeem on the same day and if the points of the individual card are sufficient for paying the amount due.

Exercise 4 – `hashCode()` Performance Tests #equals

We have seen in the lecture that whenever two objects are considered equal, the contract of the `hashCode` methods states that the returned integer value also needs to be equal. Though it is not required, objects that are not considered equal should have a differing hash code to improve the performance of a certain number of operations in hash tables.

The purpose of this exercise is to show that a proper implementation of `hashCode` can have a significant impact on the performance of operations such as adding and retrieving elements in a `HashSet`.

Define a class with some attributes. Make sure the `equals` method is properly implemented. Provide different implementations of the `hashCode` method, ranging from a constant integer value returned to highly dynamic values.

Investigate the performance impact with respect to the differentiation of the returned hash values for both filling and retrieving elements in a vast hash set.

Variate your experiments for different numbers of elements in the set and for different implementations of `hashCode`. Document your observations.

Exercise 5* – Under The Hood #GenericType #equals

On Moodle, you can find the Java source files of `ArrayList` and `HashSet`. It might be interesting to see how these classes are realized under the hood. In particular:

1° Which data structure does `ArrayList` use to store the elements? How is the dynamic size ensured?

2° How do `HashSet`s ensure that no duplicates are added?