Advanced Programming

Lab Sheet 12 – Syntactic Sugar

Introduction

Syntactic sugar is the name given to techniques that allow you to do more work with less code, where the Java compiler automatically translates the shorter or simpler syntax into its longer-form counterpart during compilation. The language features are entirely optional: its possible to use the (longer) ways that you already know to complete any task that can be achieved using the techniques discussed. However, you may well be called upon in the future to maintain existing code that uses many of these techniques, so it is beneficial to have at least a passing degree of familiarity with each of them. These exercises give you a chance to practice these techniques, which may also come in handy on the assignment.

Task 1 – Boolean Expressions & Ternary Ifs

In your first year programming unit last year, you used the codingbat website to write some simple Java methods that were subjected to a set of automated tests, in a similar manner to the automated marker on the first assignment. Your solutions to these methods would typically have been 3-6 lines of Java code, with a few if-then-else statements and some returns. The majority of these methods could be written in a single line with appropriate use of boolean expressions, and ternary if statements, as depicted in Fig.1, below.

```
//complex expression
//true if x > y and z
boolean xbiggest = (x > y) && (x > z);

//ternary ifs
//if x > y then x, if y > z then y else z
int picked = (x>y ? x : (y > z ? y : z));
```

Figure 1: Expression & Ternary if Examples

Visit the codingbat website again (link here), and attempt to solve the first three problems using just a single line of code within each of the methods. If you're up for an extra challenge, some of the later exercises are also possible to complete in a similar manner, try your hand at the answerCell, old35 and teaParty exercises. That last one is getting a little silly, though.

Task 2 – Sorting Lists with an Anonymous Inner Class

An online game tracks various pieces of data about its users, to allow it to present various statistics and league tables. You are going to create a Java class capable of storing this data in various properties, and a Java program that creates a number of instances of this class, and is able to sort ArrayLists of these objects to present the data.

Table I, below, depicts the data held by the game on its players, with a brief description of what the data is normally used for.

Table 1: Player Data Held by Online Game

Property	**Function**
Name	Used to label the player's data with their name
Level	The player's level in game
High-Score	The player's best ever score
Account Creation Date	The date on which the account was created
Total Play Time	Total number of minutes spent in-game
Last Login Date/Time	Used to track which players have played recently

You will need to create a PlayerData class with properties for each of the pieces of data in Table I, selecting an appropriate data type for each of them. Your class will also need getters and setters for the properties, and a constructor method.

You also need to create a Main class, complete with a main() method, which will create at least 10 players with appropriate data. Store these PlayerData objects in an ArrayList.

Write a program capable of sorting and displaying the data in each of the following formats. You will need to use the Collections.sort() method, an anonymous inner class that implements the Comparator interface, and the Collections.shuffle() method to complete the tasks.

1. Displayed in their original order (the order in which you created them)

2. Displayed in a random order

3. Sorted by their high score (highest first, like a high score table)

4. Sorted by their level (lowest first)

5. Sorted by last login date/time (most recent first)

Extension Task: Lambdas and Streams

If you fancy a bit of a challenge, now that the assignment is over, then you could take a look at the official Java documentation on lambdas (here) and streams (here). Write a method that uses streams and lambdas to *filter* your ArrayList of PlayerData objects to display only players with a level lower than 10. Write another method, again using streams and lambdas, that uses the *map* and *reduce* operations to calculate the average daily play time of all players, in minutes.