Exercise: GoF Template Method and GoF Strategy

In this exercise, you will see an example of the use of GoF Template Method. You will analyse this implementation and then use it as an offset to implement a similar solution based on a GoF Strategy pattern.

Through this, you will gain an insight and appreciation of the differences and similarities between the two patterns. Oh, and you will gain an understanding (hopefully) of the power of black-box test suites[1]!

The *Heap* is a vital part of any system that allows dynamic memory allocation, e.g. C++ and C#. Whenever you wave the magic memory allocation wand and write "`var x = new Something()`", you invoke the heap management system and ask it to allocate enough memory to store a new `Something` object. This allocation takes its memory from a dedicated section of memory, namely the heap.

We can consider the heap as a long, contiguous section of memory – in essence, it is an array. Memory in the heap can be either *allocated* or *free*. Clients of the heap can allocate memory from the `Heap` by calling `Allocate()` and de-allocate it after use by calling `Deallocate()` (see below).

There are two vital operations on the `Heap`:

- `int Allocate(int x)` allocates `x` bytes from the heap and returns the address of the allocated block. When memory is allocated, others cannot use it until it is freed again. `Allocate()` will fail if there is not enough free memory to satisfy the request.

- `void Deallocate(int addr, int x)` deallocates the memory from address `addr` and `x` bytes forwards, so that it may be allocated by other clients in the future. `Deallocate()` will fail if the entire block is not allocated.

The `Heap` can allocate memory in the heap in different ways. The four prominent algorithms for this are *first-fit*, *next-fit*, *best-fit* and *worst-fit*. A brief description of the four algorithms is given at the end of this exercise text.

Your friendly teacher has provided you with an implementation of the `Heap`, which uses GoF Template Method to implement the four different algorithms for memory allocation. He has also provided an extensive test suite because he really likes to test stuff.

Exercise 1:
Identify exactly where the GoF Template Method pattern is used in the provided code. Which methods is/are the template method(s), which methods are the abstract variation points and how does the pattern work to implement the four different algorithms?

Exercise 2:
Extend the existing implementation by implementing the two algorithms *best-fit* and *worst-fit*. Hint: If you understand where the GoF Template Method pattern is used for the provided *first-fit* and *next-fit* algorithm, you have also understood where the variation point(s) is/are. This is where you need to "plug in" your implementations. Don't worry, the two algorithms are way simpler to implement than next-fit.

Exercise 3:
Change the implementation to use GoF Strategy instead of GoF Template Method

[1] For all you test aficionados out there: Note how the test fixtures for the individual algorithms all inherit from a base test suite, which tests scenarios that <u>all</u> algorithms must pass, regardless of implementation. Note also how the base test fixture uses the GoF Factory Method pattern to obtain it's concrete unit-under-test object. Nifty, isn't it?

Exercise 4:
Ideally, you should be able to use the same test cases for the GoF Strategy-based solution as you did for the GoF Template Method-solution, although the initiation of objects may be different. This is because your teacher has written a black box test suite and he likes black box tests.

Answer the rhetorical question: Do you see how a black box test suite is preferable to a white box test suite in this case, where you refactor an existing solution?

A brief description/example of the memory allocation algorithms

With reference to Figure 1: Assume that a heap of size 100 has three blocks allocated:

- Block '1' from address 10-40
- Block '2' from address 80-85
- Block '3' from address 45-70.

Assume also that the blocks were allocated in that order. When the heap receives an allocation request for 5 bytes the four algorithms will return different addresses:

- *first-fit* will return address ‘0’ because this is the address of the *first* block in the heap that will satisfy the request.
- *next-fit* will return address ‘70’, because this is the address of the first block in the heap *after* the most recent allocation that will satisfy the request.
- *best-fit* will return address ‘40’ because this is the start address of the *smallest* block in the heap that will satisfy the request
- *worst-fit* will return address ‘85’ because this is the address of the *largest* block in the heap that will satisfy the request.

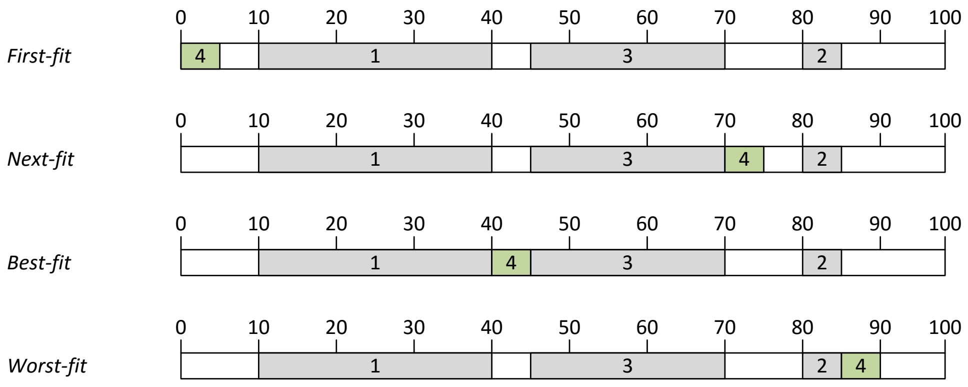

Figure 1: The four prominent memory allocation algorithms and their satisfaction of a memory allocation request.