OU2 - Design Patterns

HT2022

2022-12-12

Theodor Jonsson
ens18trn@cs.umu.se

Applikationsutveckling (Java)
5DV222

1. Introduction

This assignment is composed of four smaller assignments in which the task is to create programs or implement functions using different types of design patterns. All of these programs are created and compiled in Java 17.

2. Builder

The builder design pattern is useful when an object has multiple attributes that aren't always needed and don't want to create several different constructors with different parameters. This task is to create a dog with the attributes name, type, age and a list of toys and we want to be able to instantiate it with any combination of attributes.

We create an inner class in the Dog class that is a DogBuilder and that implements the given Builder interface. This inner class has methods to assign values to the attributes of the Dog class and a method called build which calls the constructor of the dog to create the object. The attributes that haven't been assigned from the DogBuilder class will be null. The constructor of the Dog class is private because it is not supposed to be used outside of the DogBuilder class. See code example of building a dog using this design pattern

```
Builder building = new Dog.DogBuilder();
building.name("Nova");
building.type("Chihuahua");
Dog hund = (Dog) building.build();
```

Figure 1. Example of building a dog using DogBuilder

Theodor Jonsson
ens18trn@cs.umu.se

3. Decorator

This task is to extend the bakery program by being able to create more advanced cakes in the form of different extras and a different kind of cake.

This is accomplished by adding functionalities to the abstract class Cake as in adding methods for adding sprinkles, adding text and making the cake extra large. Then making the different types of cake classes inherit from the Cake class and only calling methods from the superclass but with different parameters to account for the different prices of the cakes. All this is done to make it easier to add on more features such as different cakes or different extras. Figure 2 shows how the StrawberryCake class only uses methods from the superclass Cake to construct itself.

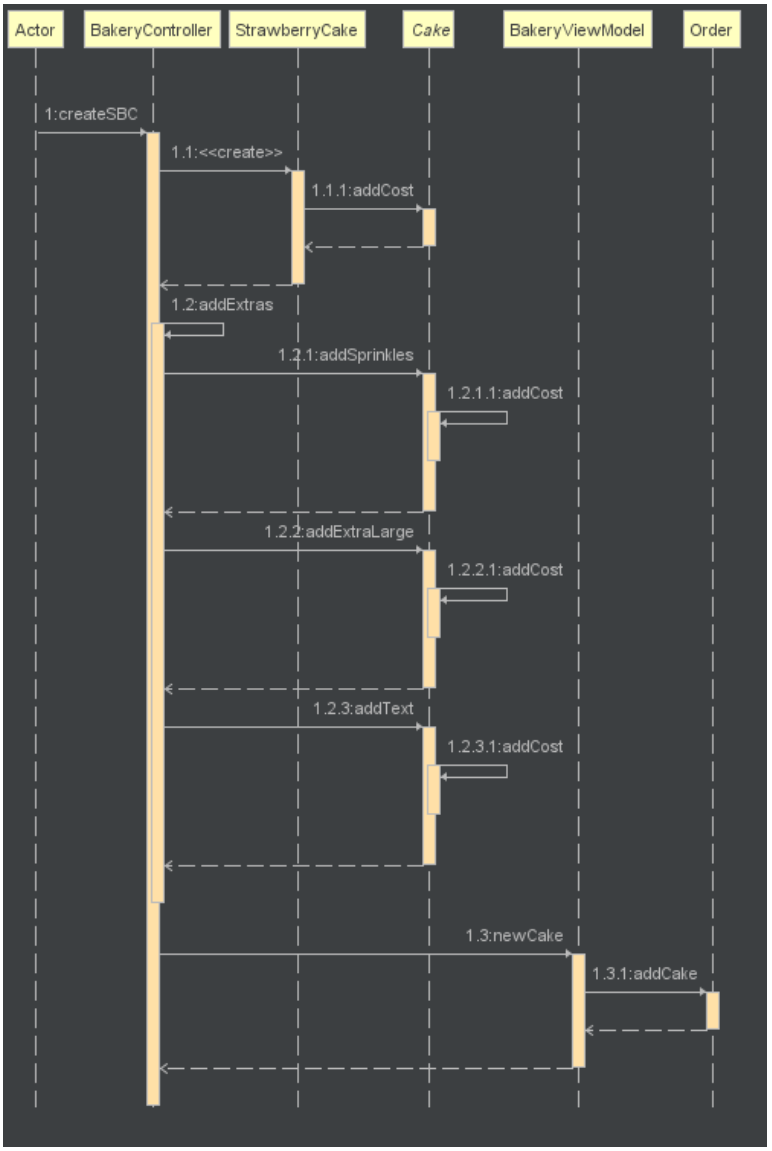

Figure 2. Sequence diagram when the user purchases a Strawberry cake

Theodor Jonsson
ens18trn@cs.umu.se

4. Memento and Observer

This task was to create a text editor that is able to save states of the text editor and later restore these states. States meaning text and placement of the cursor. The text editor uses the Memento and Observer to accomplish this. When the user types in the text editor the JETextArea will notify all listeners of updates in this case it is the EditorText class. The EditorText class will store the text in the JETextArea as a String and the cursor positions as integers.

When the user clicks the save button a memento of the EditorText will be saved and stored in a stack. When the user wants to undo and go back to a stored state it will pop the stack and assign the mementos attributes to the EditorText this will notify the JETextArea which will update the GUI.

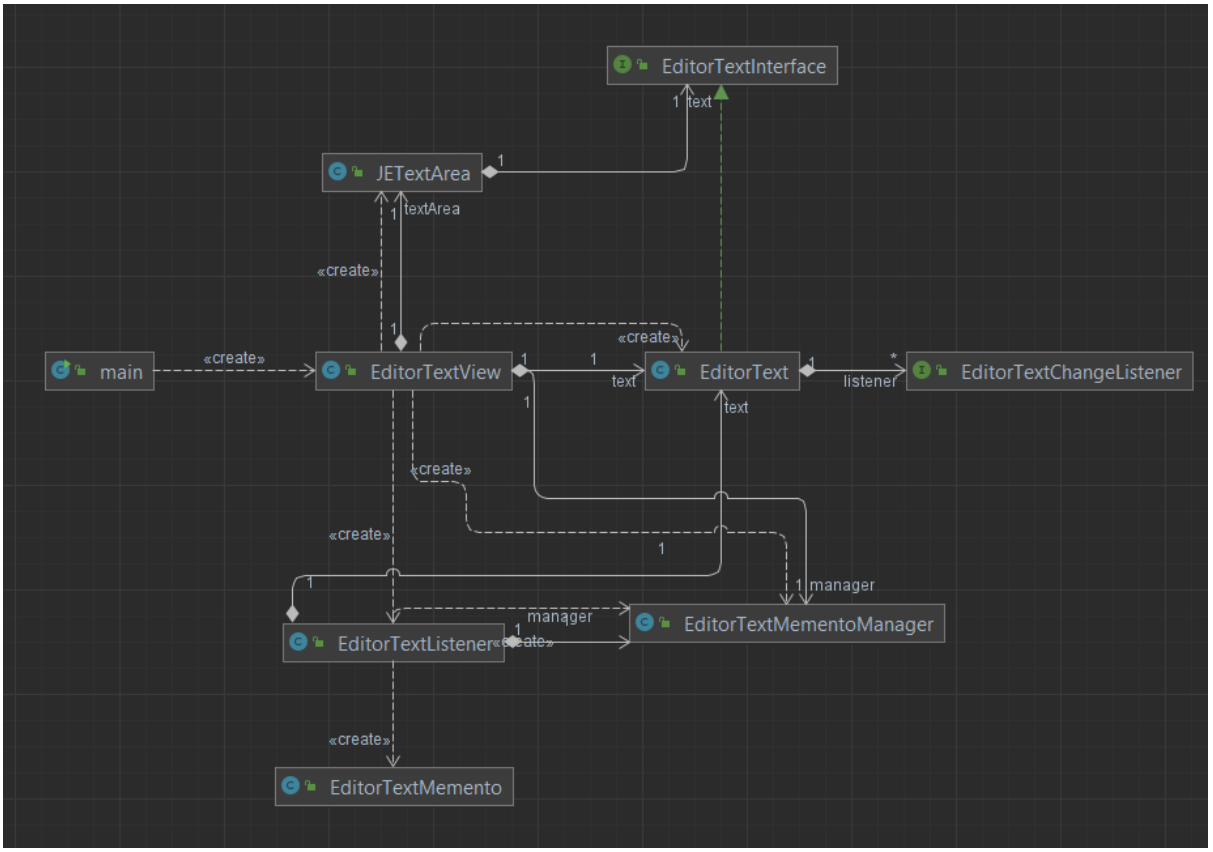

Figure . UML - Diagram over the EditorText program

Theodor Jonsson
ens18trn@cs.umu.se

5. Iterator

The iterator design pattern is a design pattern which lets you traverse a collection of elements without exposing the underlying representation. In this task the underlying representation is a doubly linked list.

Here we are tasked to implement the Iterable interface in the doubly linked list to accomplish this we have to create a method that returns an iterator. This iterator in this task is a private inner class that implements the iterator interface and overrides the default methods called next which returns the next element and hasNext which returns true if there is another element false if there isn't. These methods are used in the default implementation of the foreach loop

```
public class testDoublelinkedlist {
    public static void main(String[] args) {
        DoubleLinkedList<Integer> list = new DoubleLinkedList();
        DoubleLinkedList.Position pos = list.first();
        for(int i = 0; i < 100; i++){
            list.insert(i, pos);
            list.next(pos);
        }


        for(Integer i : list){
            System.out.println(i);
        }

    }
}
```

Figure . Example use of the iterator