"Coffee Break"

Approfondimento di Ingegneria del Software
Singleton e Decorator

Giulia Forasassi

Gennaio, 2018

1 Introduzione

L'approfondimento dell'elaborato riguarda la collaborazione tra due design pattern: il Singleton e il Decorator.

Il **Singleton** è un pattern *creazionale* il cui scopo è assicurare che una classe abbia una sola istanza di se stessa e fornire un punto d'accesso globale a tale istanza. L'implementazione di questo pattern prevede che la classe Singleton abbia un unico costruttore privato in modo da impedire l'instanziazione diretta della classe. La classe fornisce inoltre un metodo getter statico che restituisce l'istanza della classe e ne memorizza il riferimento in un attributo anch'esso statico.

Mentre il **Decorator** è un pattern *strutturale* che permette di aggiungere nuove funzionalità ad un oggetto già esistente. Questo viene realizzato usando un'interfaccia Component che può essere implementata mediante un oggetto concreto o un oggetto decorato la cui classe avvolge l'oggetto originale. Le funzionalità aggiuntive vengono specificate nei ConcreteDecorators e possono essere definite come attributi che identificano nuovi stati oppure come metodi che ne identificano nuovi comportamenti. Il client interagisce solo con l'interfaccia del Component e non si preoccupa se l'oggetto richiesto è un oggetto base o un oggetto decorato.

2 Scenario

Lo scenario proposto prevede la **produzione** di diversi tipi di **caffè** da parte di un barman. E' stato usato il pattern singleton per creare una singola istanza del barman mentre il pattern decorator è stato usato per creare un caffè base Espresso e altri tipi di caffè aggiungendo decorazioni a quest'ultimo.

Le varie **decorazioni** consistono nell'aggiunta di un ingrediente come il latte, la cioccolata, o una bevanda alcolica al caffè Espresso di base. Di conseguenza le varie bevande decorate che vengono prodotte possono essere: Caffè Macchiato, Cappuccino, Caffè Corretto, Caffè con cioccolata e Mocaccino.

Inoltre il cliente può ordinare caffè di diversa altezza: alto o basso, e ci può aggiungere o meno l'ingrediente zucchero.

Si può inoltre ottenere informazioni aggiuntive su ogni bevanda ordinata dal cliente come per esempio il tipo di bevanda, il costo, le calorie e gli ingredienti che contiene.

3 Class Diagram Singoli Design Patterns

Singleton

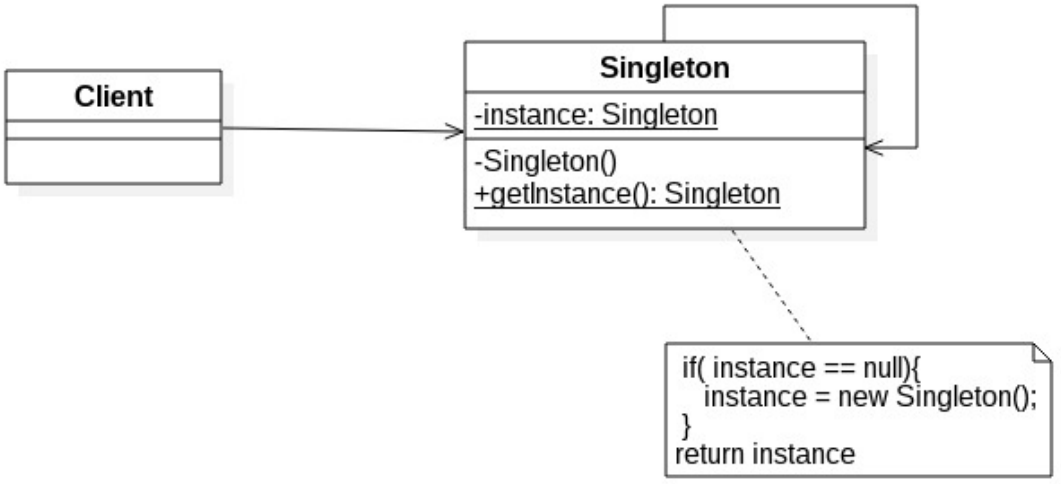

Figure 1: Class Diagram del Singleton

Decorator

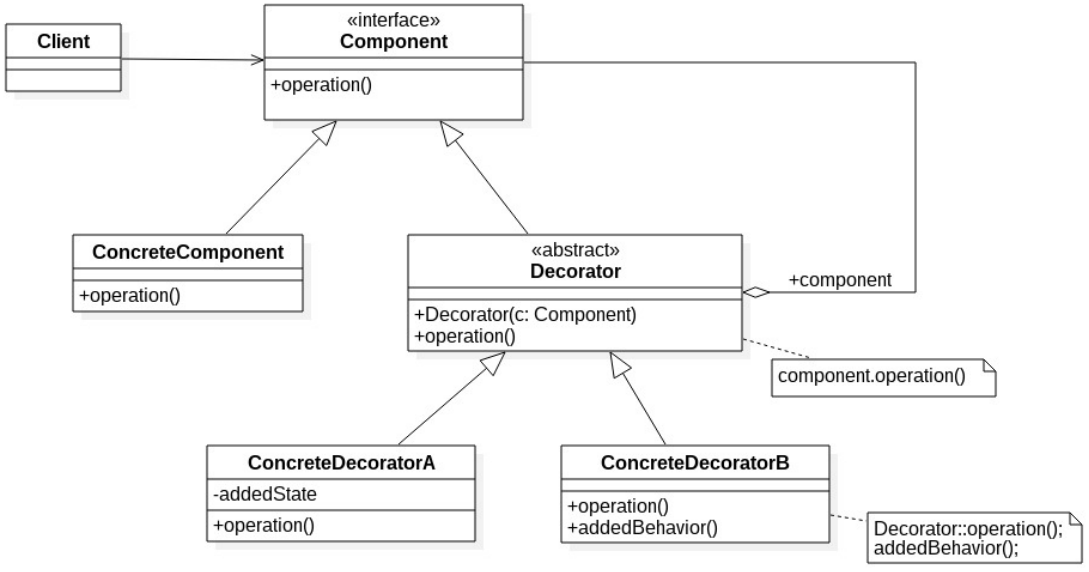

Figure 2: Class Diagram del Decorator

4 Class Diagram dello scenario

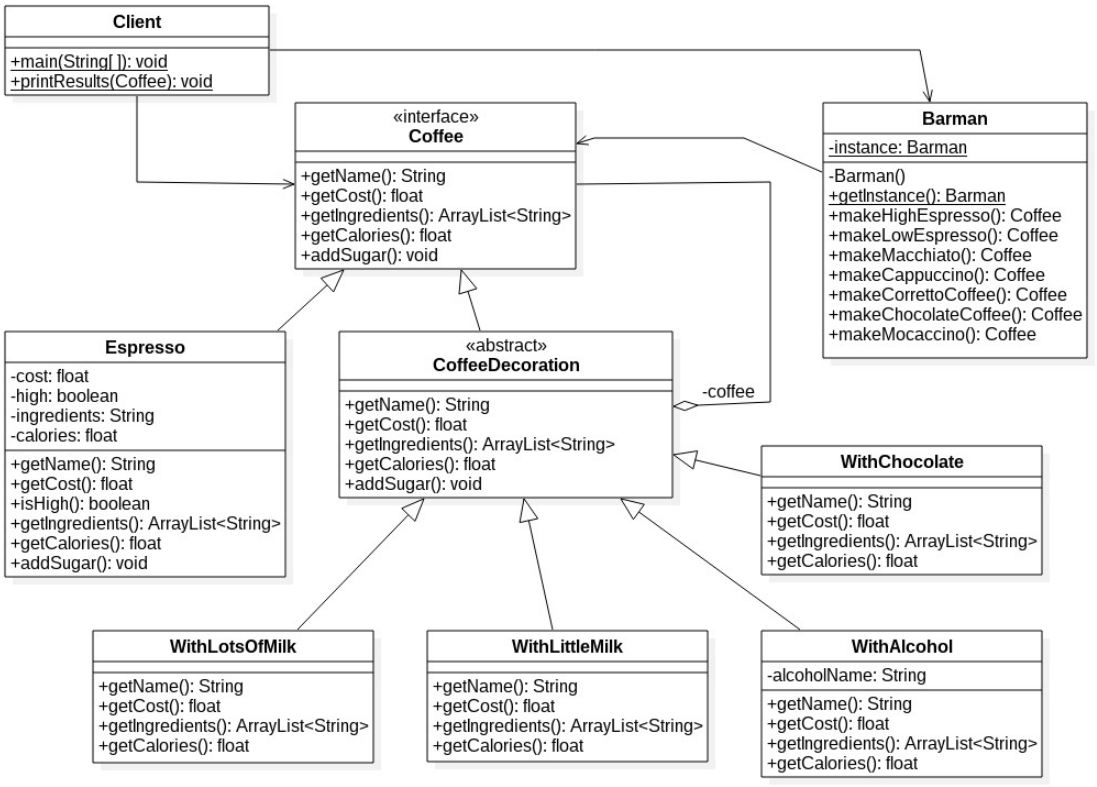

Figure 3: Class Diagram dello scenario proposto

5 Partecipanti

- **Coffee**: corrisponde alla classe Component del pattern Decorator e definisce l'interfaccia comune per gli oggetti ai quali possono essere aggiunte responsabilità dinamicamente. Definisce dei metodi get: *getName()*, *getCost()*, *getCalories()*, *getIngredients()* che ritornano rispettivamente il nome del caffè scelto, il costo che deve pagare il cliente, le calorie e la lista di ingredienti contenuti nel caffè . Inoltre vi è un metodo *addSugar()* che permette di aggiungere lo zucchero al caffè.
- **Espresso**: corrisponde alla classe ConcreteComponent del pattern Decorator e definisce un oggetto concreto Caffè al quale possono essere aggiunte nuove responsabilità, ovvero degli ingredienti aggiuntivi. Contiene i metodi getter della classe padre Coffee, e inoltre definisce un metodo *isHigh()* che ritorna un booleano in base alla scelta di un caffè basso o alto.
- **CoffeeDecoration**: corrisponde alla classe Decorator e definisce una classe astratta che mantiene un riferimento ad un oggetto Coffee e mantiene un'interfaccia conforme all'interfaccia di Coffee. Definisce i metodi getter della classe padre.

- **WithLotsOfMilk**: classe decoratrice che aggiunge una grande quantità di un nuovo ingrediente, latte, all'oggetto ti tipo Coffee. Vengono ridefiniti i metodi getter della classe padre che aggiungono dettagli al nome, aumentano il costo, ridefiniscono le calorie e gli ingredienti rispetto all'ingrediente aggiunto.
- **WithLittleMilk**: classe decoratrice che aggiunge una piccola quantità dell'ingrediente latte all'oggetto di tipo Coffee. Vengono ridefiniti i metodi getter della classe padre che aggiungono dettagli al nome, aumentano il costo, ridefiniscono le calorie e gli ingredienti rispetto all'ingrediente aggiunto.
- **WithAlcohol**: classe decoratrice che aggiunge una quntità di alcohol, scelto a piacere dal cliente, all'oggetto di tipo Coffee. Vengono ridefiniti i metodi getter della classe padre che aggiungono dettagli al nome, aumentano il costo, ridefiniscono le calorie e gli ingredienti rispetto all'ingrediente aggiunto.
- **WithChocolate**: classe decoratrice che aggiunge l'igrediente cioccolata all'oggetto di tipo Coffee. Vengono ridefiniti i metodi getter della classe padre che aggiungono dettagli al nome, aumentano il costo, ridefiniscono le calorie e gli ingredienti rispetto all'ingrediente aggiunto.
- **Barman**: si tratta del Singleton, definisce un operazione getInstance() che consente al client di accedere all'unica istanza esistente della classe. Inoltre mette a disposizione dei metodi per ordinare i vari tipi di caffè.
- **Client**: ordina al barman una tipologia di caffè e riceve ogni volta un oggetto di tipo Coffee visualizzandone i dettagli come il nome, il costo, le calorie e gli ingredienti contenuti.

6 Codice

Coffee.java

```java
package com.giulis13;

import java.util.ArrayList;

// Component
public interface Coffee {

    String getName();
    float getCost();
    ArrayList <String> getIngredients();
    float getCalories();
    void addSugar();
}
```

Espresso.java

```java
package com.giulis13;

import java.util.ArrayList;

// Concrete Component
public class Espresso implements Coffee {

    // attributi
    private float cost;
    private boolean high;
```

```
private ArrayList <String> ingredients;
private float calories;
private int sugar;

// costruttore
public Espresso(float cost, boolean high) {
    this.cost = cost;
    this.high = high;
    this.ingredients = new ArrayList<String>();
    ingredients.add("Coffee");
    this.calories = (float) 2.5;
    this.sugar = 0;
}

// metodo per guardare il nome del caffe'
@Override
public String getName() {
    String name;
    if(high)
        name = "High Coffee";
    else
        name = "Low Coffee";

    // per cambiare il nome in caso ci sia lo zucchero
    if(sugar > 0){
        name = name + " with sugar";

    }
    return name;
}

// metodo per ottenere il costo del caffe'
@Override
public float getCost() {
    return cost;
}


// metodo per ottenere gli ingredienti del caffe'
@Override
public ArrayList<String> getIngredients() {
    return ingredients;

}

// metodo per ottenere le calorie del caffe'
@Override
public float getCalories() {
    return calories;
}

// metodo che ritorna l'altezza del caffe'
public boolean isHigh() {
    return high;
}

// metodo per aggiungere lo zucchero
public void addSugar(){
    if(sugar == 0) {
        ingredients.add("Sugar");
    }
```

```
        sugar ++;
        calories = calories + 20;
    }
}
```

CoffeeDecoration.java

```
package com.giulis13;

// Decorator
public abstract class CoffeeDecoration implements Coffee {

    // attributo
    protected Coffee coffee;

    // costruttore
    public CoffeeDecoration(Coffee coffee) {
        this.coffee = coffee;
    }

    @Override
    public void addSugar() {
        coffee.addSugar();
    }
}
```

WithLotsOfMilk.java

```
package com.giulis13;

import java.util.ArrayList;

// Concrete Decorator
public class WithLotsOfMilk extends CoffeeDecoration {

    // costruttore
    public WithLotsOfMilk(Coffee coffee) {
        super(coffee);
    }

    // viene ridefinito il nome
    @Override
    public String getName() {
        return coffee.getName() + " with lots of milk";
    }

    // viene ridefinito il costo
    @Override
    public float getCost() {
        return (float) (coffee.getCost() + 1.50); // cast del float
    }

    // vengono ridefiniti gli ingredienti
    @Override
    public ArrayList<String> getIngredients() {
        ArrayList <String> list = coffee.getIngredients();
```

```
        list.add("Milk");
        return list;
    }

    // vengono ridefinite le calorie
    @Override
    public float getCalories() {
        return coffee.getCalories() + 70;
    }
}
```

WithLittleMilk.java

```
package com.giulis13;

import java.util.ArrayList;

// Concrete Decorator
public class WithLittleMilk extends CoffeeDecoration {

    // costruttore
    public WithLittleMilk(Coffee coffee) {
        super(coffee);
    }

    // viene ridefinito il nome
    @Override
    public String getName() {
        return coffee.getName() + " with a little milk";
    }

    // viene ridefinito il costo
    @Override
    public float getCost() {
        return (float) (coffee.getCost() + 0.50); // cast del float
    }

    // vengono ridefiniti gli ingredienti
    @Override
    public ArrayList<String> getIngredients() {
        ArrayList <String> list = coffee.getIngredients();
        list.add("Milk");
        return list;
    }

    // vengono ridefinite le calorie
    @Override
    public float getCalories() {
        return (float) (coffee.getCalories() + 7.5); // cast di float
    }
}
```

WithChocolate.java

```java
package com.giulis13;

import java.util.ArrayList;

// Concrete Decorator
public class WithChocolate extends CoffeeDecoration{

    // costruttore
    public WithChocolate(Coffee coffee) {
        super(coffee);
    }

    // viene ridefinito il nome
    @Override
    public String getName() {
        return coffee.getName() + " with chocolate";
    }

    // viene ridefinito il costo
    @Override
    public float getCost() {
        return (float) (coffee.getCost() + 0.80); // cast del float
    }

    // vengono ridefiniti gli ingredienti
    @Override
    public ArrayList<String> getIngredients() {
        ArrayList <String> list = coffee.getIngredients();
        list.add("Chocolate");
        return list;
    }

    // vengono ridefinite le calorie
    @Override
    public float getCalories() {
        return coffee.getCalories() + 13;
    }
}
```

WithAlcohol.java

```java
package com.giulis13;

import java.util.ArrayList;

// Concrete Decorator
public class WithAlcohol extends CoffeeDecoration {

    private String alcoholName;

    // costruttore
    public WithAlcohol(Coffee coffee, String alcoholName) {
        super(coffee);
        this.alcoholName = alcoholName;
    }
```

```
    // viene ridefinito il nome
    @Override
    public String getName() {
        return coffee.getName() + " with " + alcoholName;
    }

    // viene ridefinito il costo
    @Override
    public float getCost() {
        return (float) (coffee.getCost() + 1.00); // cast del float
    }

    // vengono ridefiniti gli ingredienti
    @Override
    public ArrayList<String> getIngredients() {
        ArrayList <String> list = coffee.getIngredients();
        list.add("Alcohol");
        return list;
    }

    // vengono ridefinite le calorie
    @Override
    public float getCalories() {
        return (float) (coffee.getCalories() + 22.5); // cast del float
    }
}
```

Barman.java

```
package com.giulis13;

// Singleton
public class Barman {

    // attributo
    private static Barman instance;

    // costruttore
    private Barman(){

    }

    // metodo statico per istanziare un Barman
    public static Barman getInstance(){
        if(instance == null){
            instance = new Barman();
        }
        return instance;
    }

    // metodo per fare un Espresso corto
    public Coffee makeLowEspresso(){
        Coffee espresso = new Espresso((float) 1.00, false);
        return espresso;
    }

    // metodo per fare un Espresso lungo
    public Coffee makeHighEspresso(){
```

```
        Coffee espresso = new Espresso((float) 1.00, true);
        return espresso;
    }

    // metodo per fare un Caffe' Macchiato
    public Coffee makeMacchiato(){
        return new WithLittleMilk(makeLowEspresso());
    }

    // metodo per fare un Cappuccino
    public Coffee makeCappuccino(){
        return new WithLotsOfMilk(makeLowEspresso());
    }

    // metodo per fare un Caffe' Corretto
    public Coffee makeCorrettoCoffee(String alcoholName){
        return new WithAlcohol(makeLowEspresso(), alcoholName);
    }

    // metodo per fare un Caffe' con la cioccolata
    public Coffee makeChocolateCoffee(){
        return new WithChocolate(makeLowEspresso());
    }

    // metodo per fare un Mocaccino
    public Coffee makeMocaccino(){
        return new WithChocolate(makeCappuccino());
    }
}
```

Client.java

```
package com.giulis13;

public class Client {

    public static void main(String[] args) {

        // creazione del Barman
        Barman barman = Barman.getInstance();
        System.out.println("The barman is ready to make a coffee!");

        System.out.println("\n");

        // creazione dell'Espresso corto
        Coffee coffee = barman.makeLowEspresso();
        printResults(coffee);

        // creazione dell'Espresso lungo
        Coffee coffee1 = barman.makeHighEspresso();
        printResults(coffee1);

        // creazione dell'Espresso corto zuccherato
        coffee = barman.makeLowEspresso();
        coffee.addSugar();
        printResults(coffee);

        // creazione dell'Espresso lungo zuccherato
```

```
        coffee1 = barman.makeHighEspresso();
        coffee1.addSugar();
        coffee1.addSugar();
        printResults(coffee1);

        // creazione del caffe' Macchiato
        Coffee coffee2 = barman.makeMacchiato();
        printResults(coffee2);

        // creazione del Cappuccino
        Coffee coffee3 = barman.makeCappuccino();
        printResults(coffee3);

        // creazione del caffe' Corretto con grappa
        Coffee coffee4 = barman.makeCorrettoCoffee("grappa");
        printResults(coffee4);

        // creazione del caffe' Corretto con sambuca
        Coffee coffee5 = barman.makeCorrettoCoffee("sambuca");
        printResults(coffee5);

        // creazione del caffe' con Cioccolata
        Coffee coffee6 = barman.makeChocolateCoffee();
        printResults(coffee6);

        // creazione del Mocaccino
        Coffee coffee7 = barman.makeMocaccino();
        printResults(coffee7);
    }

    // funzione per stampare i risultati ottenuti
    public static void printResults(Coffee coffee){

        System.out.println("The barman made a " + coffee.getName());
        System.out.println("The client spends " + coffee.getCost());
        System.out.println("The calories of this coffee are: " + coffee.getCalories());
        System.out.println("The coffee choose by the client contains these ingredients: "
            + coffee.getIngredients());

        System.out.println("\n");
    }
}
```

Output

```
The barman is ready to make a coffee!


The barman made a Low Coffee
The client spends 1.0
The calories of this coffee are: 2.5
The coffee choose by the client contains these ingredients: [Coffee]
```

```
The barman made a High Coffee
The client spends 1.0
The calories of this coffee are: 2.5
The coffee choose by the client contains these ingredients: [Coffee]


The barman made a Low Coffee with sugar
The client spends 1.0
The calories of this coffee are: 22.5
The coffee choose by the client contains these ingredients: [Coffee, Sugar]


The barman made a High Coffee with sugar
The client spends 1.0
The calories of this coffee are: 42.5
The coffee choose by the client contains these ingredients: [Coffee, Sugar]


The barman made a Low Coffee with a little milk
The client spends 1.5
The calories of this coffee are: 10.0
The coffee choose by the client contains these ingredients: [Coffee, Milk]


The barman made a Low Coffee with lots of milk
The client spends 2.5
The calories of this coffee are: 72.5
The coffee choose by the client contains these ingredients: [Coffee, Milk]


The barman made a Low Coffee with grappa
The client spends 2.0
The calories of this coffee are: 25.0
The coffee choose by the client contains these ingredients: [Coffee, Alcohol]


The barman made a Low Coffee with sambuca
The client spends 2.0
The calories of this coffee are: 25.0
The coffee choose by the client contains these ingredients: [Coffee, Alcohol]


The barman made a Low Coffee with chocolate
The client spends 1.8
The calories of this coffee are: 15.5
The coffee choose by the client contains these ingredients: [Coffee, Chocolate]


The barman made a Low Coffee with lots of milk with chocolate
The client spends 3.3
The calories of this coffee are: 85.5
The coffee choose by the client contains these ingredients: [Coffee, Milk, Chocolate]

Process finished with exit code 0
```

7 Sequence Diagram

Di sequito viene riportato il sequence diagram relativo alla creazione di un oggetto di tipo Cappuccino.

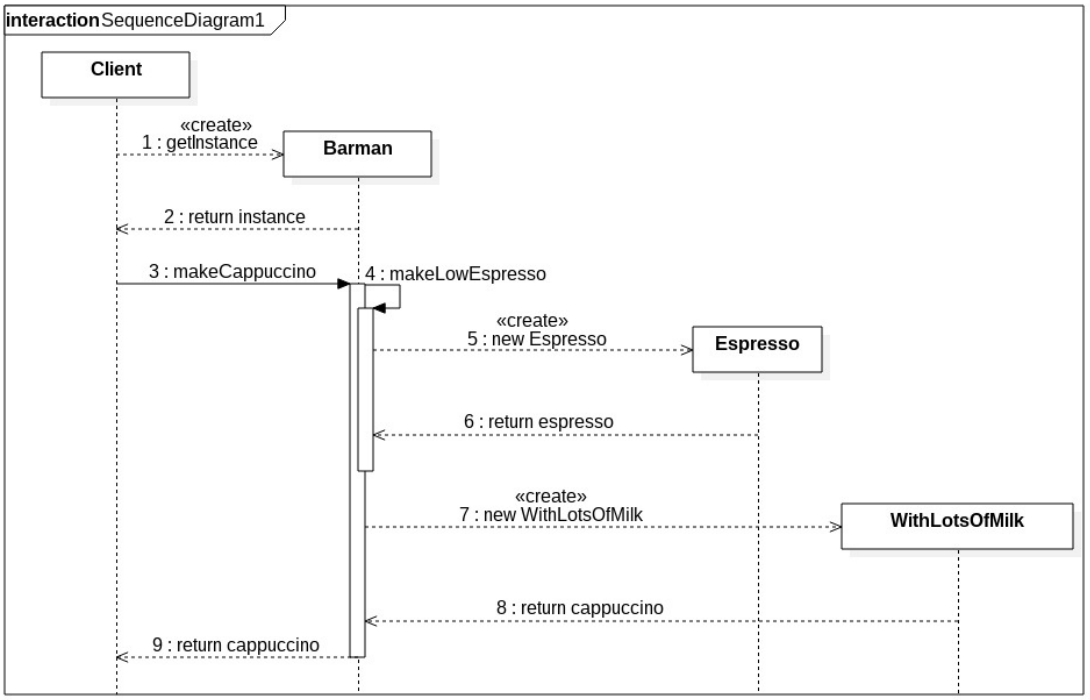

Figure 4: Esempio di Sequence Diagram: creazione di un Cappuccino