# Esercitazione di Fine Settimana – Week 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Graziella |
|  |  | Cognome | Caputo |
|  |  | Data | 7/5/2021 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.

1. Descrivere la differenza tra Value Type e Reference Type

Un dato di tipo value è un dato in cui la variabile immagazzina direttamente il dato. Quando si passa una variabile value type da un metodo ad un altro metodo, il sistema crea una copia separata della variabile nel secondo metodo: nel caso in cui il suo valore fosse cambiato in un metodo non ci sarebbero ripercussioni sulla variabile contenuta nell’altro metodo. Questo tipo di variabili viene solitamente salvato nell’area stack. Un dato di tipo reference invece è un dato in cui la variabile rappresenta un puntatore al dato contiene dunque l’indirizzo alla locazione di memoria in cui è posizionato il dato. I reference type vengono allocati all’interno dell’area heap.

1. Cos’è un Delegate? Spiegarlo anche con un esempio.  
   Un delegato è un puntatore a una funzione una sorta di oggetto che sa come chiamare un metodo. In altre parole un delegato definisce il tipo di ritorno e i tipi dei parametri di metodi.  
   Es delegate int Operazione(int primoOp, int secondoOp);

compatibile con qualsiasi metodo accetti due parametri di tipo intero che restituisca un intero (int Soma(int num1, int num2) ; int Divisione(int num1, int num2 ) ecc)

1. Descrivere il Factory pattern e perché utilizzarlo  
   È un pattern di [tipo creazionale](http://www.datrevo.com/tag/pattern-creazionale/) che si occupa dunque della costruzione degli oggetti e delle problematiche che si possono presentare. Questo tipo di pattern nasconde i dettagli della creazione e rende i sistemi indipendenti da come gli oggetti sono creati e composti.
2. A cosa serve il metodo Assert della libreria Xunit?  
   Si tratta di una classe, fornita dal framework xUnit, che propone dei metodi statici per eseguire il confronto tra il risultato atteso e quello ottenuto mettendoli a confronto.

**Esercitazione Pratica**

Realizzare una Console app (C#) che:

* Effettui il monitoraggio di una cartella in attesa di un file di testo con l'elenco delle spese (*spese.txt*)
* Apra e legga il file. Ogni riga è nel formato:



* Per ogni riga, determini se la spesa è approvata. Esistono diversi livelli di approvazione, a seconda dell'import della spesa
  + **Manager**: spese fino a 400€
  + **Operational Manager**: da 401€ fino a 1000€
  + **CEO**: sopra i 1000€
  + Nessuna spesa sopra i 2500€ è approvata

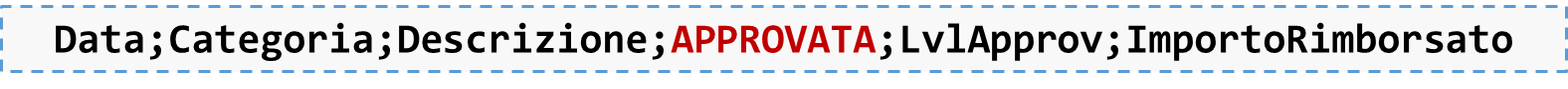
**Usare il Chain of Responsibility pattern**

**(restituire il livello di approvazione)**

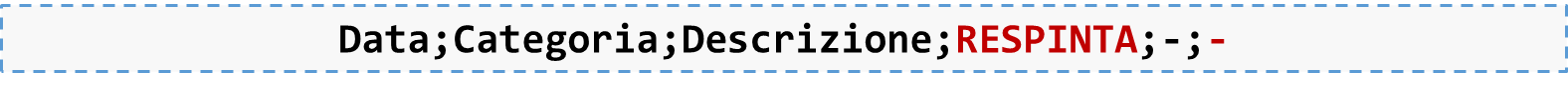
* Per ogni spesa approvata, determini l'importo rimborsato sulla base della Categoria
  + **Viaggio**: 100% dell'importo + 50€ fisse
  + **Alloggio**: 100% dell'importo
  + **Vitto**: 70% dell'importo
  + **Altro**: 10% dell'importo

**Usare il Factory pattern**

* Salvi poi le informazioni sulle spese rimborsate e non rimborsate in un file di testo (*spese\_elaborate.txt*)
* Per ogni spesa rimborsata salvare una riga nel formato



* Per ogni spesa non rimborsata salvare una riga nel formato



**OPZIONALE**

* Realizzare una batteria di test (*xUnit*) che verifichi la corretta gestione dei seguenti casi da parte della catena di approvazione:

