Code error - Demo

Un *code error* rapprenta un frammento di codice (un'istruzione, un metodo, più metodi) che contiene uno o più errori. I *code error* possono appartenere alle seguenti categorie:

- *code error sintattici*: il codice viola le regole del linguaggio.
- *code error di esecuzione*: il codice è sintatticamente corretto, ma produce uno o più errori di esecuzione (il programma crasha).
- *code error logici*: il codice è corretto e non crasha, ma non produce il risultato atteso.

Due puntualizzazioni:

- Nella categoria errori sintattici includo tutti quelli segnalati dal compilatore, anche se in realtà sono semantici (tipo un'assegnazione tra variabili di tipo diverso).
- In alcuni casi l'unica differenza tra errori di esecuzione ed errori logici è il "sintomo" (crash ↔ risultato errato).

Metodo di correzione

I *code error* sintattici dovrebbero essere risolti mediante *code review* (revisione del codice), senza l'ausilio di Visual Studio. Per quanto riguarda i *code error* di esecuzione e logici si dovrebbe comunque tentare una rapida correzione mediante *code review*, e soltanto successivamente utilizzare Visual Studio.

Dimostrazione

Di seguito sono mostrati alcuni *code error*; per ognuno viene proposta la versione corretta. Se necessario:

- Il *code error* viene preceduto da una premessa che spiega lo scopo del codice.
- Vengono indicati l'output prodotto e l'output atteso.

1 Code error sintattici

Code error 1

```
int[] lista = new  [] ( 1, 2, 3);
```

Il codice contiene tre errori.

```
int[] lista = new int[] { 1, 2, 3 };
```

Code error 2

```
int a = int.Parse(Console.ReadLine());
if (a = 0)
    Console.WriteLine("a è zero");
else
    Console.WriteLine("a è diversa da zero");
```

Il codice contiene un errore:

```
int a = int.Parse(Console.ReadLine());
if (a == 0)
    Console.WriteLine("a è zero");
else
    Console.WriteLine("a è diversa da zero");
```

2 Code error di esecuzione

Code error 1

```
static int[] CopiaVettore(int[] dati, int n)
{
    int[] ris = null; //-> la causa è qui!
    for (int i = 0; i < n; i++)
    {
        ris[i] = dati[i]; //-> il sintomo (NullReferenceException) si manifesta qui!
    }
    return ris;
}
```

Il codice contiene un errore.

```
static int[] CopiaVettore(int[] dati, int n)
{
    int[] ris = new int[n];
    for (int i = 0; i < n; i++)
    {
        ris[i] = dati[i];
    }
    return ris;
}
```

3 Code error logici

Code error 1

```
static int Minore(int[] dati)
{
    int min = 0; //-> la causa è qui!
    for (int i = 0; i < dati.Length; i++)
    {
        if (dati[i] < min)
        {
            min = dati[i];
        }
    }
    return min;
}
```

Il codice contiene un errore logico, poiché funziona soltanto se il vettore memorizza almeno un numero negativo; in caso contrario il valore restituito è zero.

```
static int Minore(int[] dati)
{
    int min = dati[0]; // presuppone che "dati" contenga almeno un elemento
    for (int i = 1; i < dati.Length; i++)
    {
        if (dati[i] < min)
        {
            min = dati[i];
        }
    }
    return min;
}
```

Nota bene: la versione corretta del metodo si basa sulla premessa che il vettore non sia vuoto (o nullo); in caso contrario produce un'eccezione di tipo `IndexOutOfRangeException` (oppure di tipo `NullReferenceException`).

La premessa è valida, poiché è privo di significato restituire il valore minimo di un vettore vuoto (o inesistente). Dunque: un'eventuale eccezione non è da imputarsi a un errore nel metodo, ma al fatto che il codice chiamante non rispetta la suddetta premessa.