

Université Abdelmalek Essaadi
Ecole Nationale des Sciences Appliquées
Al Hoceima



Travaux Pratiques

Les bases de la programmation C# .Net

Deuxième Année Génie Informatique

Module : Technologies Distribuées

Année Universitaire : 2021/2022

Enseignant : Tarik BOUDAA

Email : t.boudaa@uae.ac.ma

Travaux Pratiques N°1 : Les bases de du langage C#

1- Implémentation de la méthode dichotomie en C#

a et b deux nombres réels et f une fonction réelle continue et strictement monotone sur l'intervalle $[a, b]$. L'équation $f(x) = 0$ a un zéro unique dans l'intervalle $[a, b]$ (*résultat du théorème des valeurs intermédiaires + f bijective*). Une des méthodes de résolution de $f(x) = 0$ est la dichotomie, son algorithme peut se présenter par :

```
Tant que (b - a) > ε
    Calcul m = (a + b) / 2
    Calcul f(m)
    Si ((f(a)*f(m)) > 0) alors
        m → a
    sinon
        m → b
    Fin
Fin
```

Figure 1

Avec ϵ la précision souhaitée.

- Ecrire une interface **Function** qui sert à présenter une fonction mathématique avec une méthode abstraite *computeFunctionValue*.
- Ecrire une classe **Dichotomie** disposant d'une méthode *solveDichotomie(double a, double b, double p, Function f)* avec a et b les bornes de l'intervalle dans lequel on cherche la solution et p la précision souhaitée.
- Ecrire un programme de test qui calcule la solution de l'équation :
 $x^5 + 3x^4 + 2x^3 + x^2 + 3x + 1 = 0$ dans l'intervalle $[-1000, 1000]$ avec la précision $1.0e-15$.

2- Implémentation d'une pile et son utilisation

On veut écrire une classe de gestion d'une pile générique offrant les opérations classiques suivantes : Initialiser, Afficher, Empiler, Dépiler et Donner le sommet de la pile.

1. Définir l'exception ***PileVideException***
2. Donner une définition de la classe ***Pile*** qui définit une pile générique (pile que l'on représentera à l'aide d'une liste d'objets).
Cette classe comportera les méthodes suivantes :
 - a- Un constructeur sans argument construit une pile vide.
 - b- Une méthode ***estVide*** permettant de tester si la pile est vide.
 - c- Une méthode ***empiler*** ajoute un élément au sommet de la pile.
 - d- Une méthode ***depiler*** retourne le sommet et le retire de la pile, si la pile est vide la méthode génère une exception de type ***PileVideException***.
 - e- Une méthode ***getSommet*** retourne le sommet de la pile sans le retirer, si la pile est vide la méthode génère une exception de type ***PileVideException***.
3. On veut définir une méthode qui vérifie si une chaîne (ou expression) est bien " parenthésée " : Une parenthèse fermante est en correspondance logique avec une parenthèse ouverte.

Exemple :

- (((a+51) - (c))) Expression correcte
- ((a+b) Expression incorrecte, ainsi que (a+b))

Ecrire une classe ***ArithmeticGrammarChecker*** disposant d'une méthode ***checkParenthesis(String expression)*** qui permet de vérifier la validité d'une expression passée en paramètre.

Indications :

Utiliser la pile définie dans la question précédente, en suivant la méthode suivante

- Parcourir l'expression et à la rencontre d'un caractère '(' on l'empile sur la pile p et à la rencontre d'un caractère ')' on dépile la pile p

- On ignore tout autre caractère.
- A la fin de l'expression, si la pile est vide l'expression est bien "parenthésée". Sinon, il y a plus de parenthèses ouvrantes que de parenthèses fermantes. Si la pile est vide prématurément, lors d'un dépilement, alors il y a plus de parenthèses fermantes que de parenthèses ouvrantes.

Travaux Pratiques N°2 : Programmation Orientée Objet en C# et accès aux données

1- POO en C# et notion d'événement

On désire réaliser une application de gestion de comptes bancaires en C#. Pour cela on a proposé d'organiser les objets de l'application en deux catégories (**CompteSurCarnet** et **CompteSurCheque**). Les classes **CompteSurCarnet** et **CompteSurCheque** ont des propriétés communes telles que **Numéro**, **Titulaire** et **Solde** qui donnent naissance à une troisième classe abstraite **Compte** considérée comme classe de base pour éviter les redondances (cf. figure 2). La classe **CompteSurCarnet** possède, en plus des propriétés communes, les propriétés **NuméroCanret** et **Plafond** (représente le débit maximal possible). La classe **CompteSurCheque** est caractérisée par le **NuméroChèque** et le **NuméroCarte** ainsi que **DateFinValiditéCarte** (de type **DateTime**).

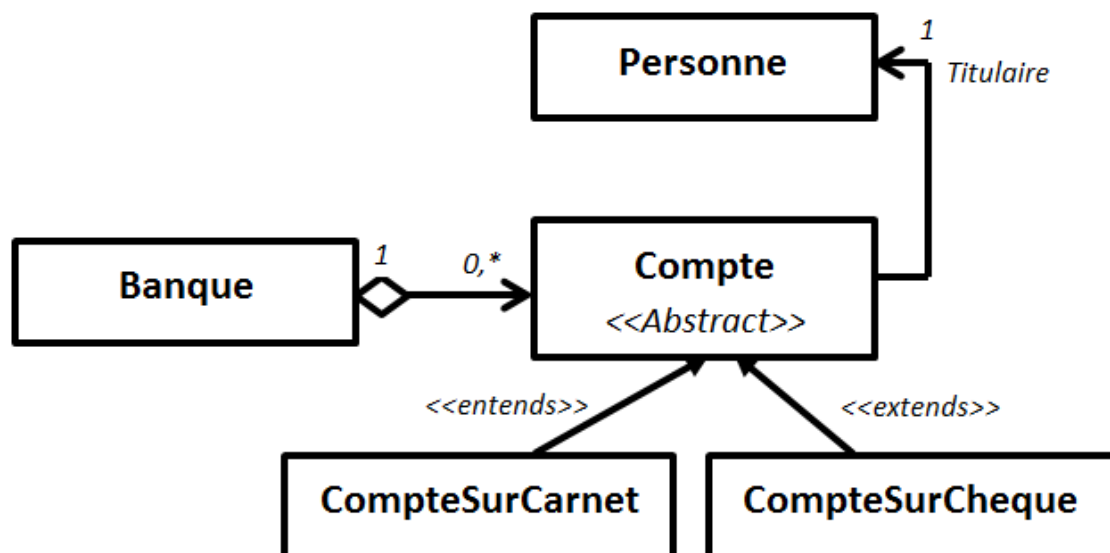


Figure 2

1. **Définir la classe abstraite Compte ayant les méthodes suivantes :**

- Un constructeur permettant d'initialiser ses propriétés
- Une redéfinition appropriée de la méthode *ToString*
- Deux méthodes abstraites *Crediter* et *Debiter* qui prennent en paramètre la somme à ajouter au compte ou à retirer du compte.

2. **Définir la classe CompteSurCheque ayant les méthodes suivantes :**

- Un constructeur permettant d'initialiser toutes ses propriétés
- Une implémentation de la méthode *Crediter* qui prend en paramètre la somme à ajouter au compte.
- Une implémentation de la méthode *Debiter* qui prend en paramètre la somme à débiter du compte qui ne doit pas dépasser le solde sinon une exception de type *Exception* est générée avec le message «Solde insuffisant».

3. **Définir la classe CompteSurCarnet ayant les méthodes suivantes :**

- Un constructeur permettant d'initialiser toutes ses propriétés
- Une implémentation de la méthode *Crediter* qui prend en paramètre la somme à ajouter au compte.
- Une implémentation de la méthode *dDebiter* qui prend en paramètre la somme à retirer du compte qui ne doit pas dépasser le solde sinon une exception est générée avec le message : « Solde insuffisant », en plus la somme à retirer ne doit pas dépasser le plafond sinon une exception de type *Exception* est générée avec le message : « Plafond dépassé ».

4. **Définir la classe Banque contenant une collection de comptes (CompteSurCheque et CompteSurCarnet) et ayant une méthode Add permettant d'ajouter un compte à la collection. (vérifier que le numéro n'existe pas déjà avant d'ajouter le compte)**

5. Nous voulons que l'objet de type **Banque** reçoit une notification automatiquement une fois qu'une opération est effectuée (débiter ou créditer) sur un de ses comptes en utilisant la notion d'événement en C#. A la réception de cette notification une méthode nommée *NotificationOperation* de la classe Banque s'exécute automatiquement et affiche les informations du compte qui a subi une opération. Pour ce faire il faut effectuer des changements dans plusieurs classes, ci-dessous un

extrait des changements effectués sur la classe Compte et un exemple de programme de test.

Donner les changements nécessaires dans toutes les classes pour avoir le fonctionnement demandé.

Extrait de la classe Compte :

```
public abstract class Compte
{
    // .... code ...
    public event SoldeChangeDelegate sldchangeEvnt;
    // .... code...
    public void SoldeChange()
    {
        if (sldchangeEvnt != null)
        {
            sldchangeEvnt(this);
        }
    }
    // .... code...
}
```

Programme de test :

```
Personne p1 = new Personne("R12111", "BOUDAA");
Personne p2 = new Personne("R12112", "FENTARSI");
Compte c1 = new CompteSurCarnet(1, p1, 100, 21);
Compte c2 = new CompteSurCarnet(2, p2, 100, 22);
Banque b = new Banque();
b.Add(c1); b.Add(c2); c1.Debiter(20); c2.Crediter(21);
```

Résultat d'exécution :

```
file:///C:/Users/boudaa/AppData/Local/Temporary Projects/DS/bin/Debug/DS.EXE
Une opération vient d'être effectuée sur le compte
:nom : BOUDAA cin :R12111 numéro : 1 solde :80
Une opération vient d'être effectuée sur le compte
:nom : FENTARSI cin :R12112 numéro : 2 solde :121
```

Figure 3

2- Application de dessin des formes géométriques avec C#

En géométrie euclidienne, un polygone est une figure géométrique plane, formée d'une suite cyclique de segments consécutifs et délimitant une portion du plan. Le polygone le plus élémentaire est le triangle : un polygone possède au moins trois sommets et trois côtés (cf. Figure 4).

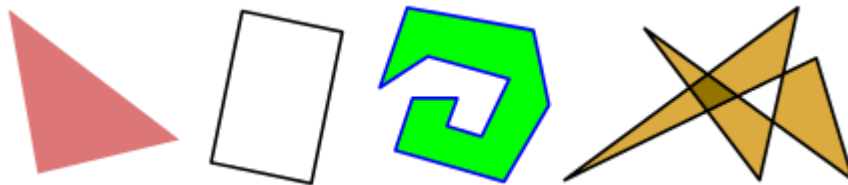


Figure 4 : Exemples de polygones

On propose dans cet exercice d'écrire une nouvelle classe Polygone où un polygone sera représenté par un tableau d'objets de type Point.

N.B. : Dans cet exercice on donnera une grande importance à la qualité de la conception des classes et à la réutilisation et maintenabilité du code grâce à l'exploitation de l'héritage et polymorphisme.

1. Écrire une classe **Point** permettant de décrire les coordonnées d'un point dans le plan. Cette classe a :
 - Deux propriétés **x** (l'abscisse) et **y** (l'ordonnée) de type double.
 - Un attribut nom de type chaîne de caractère représente le nom d'un point.
 - Un constructeur qui permet d'initialiser le point lors de sa création.
 - Ecrire une méthode **getXmlPresentation()** qui retourne une chaîne de caractères contenant la représentation XML d'un point, en respectant la

syntaxe suivante : Un point ayant le nom='A', x= X0 et y= Y0 on le représente en XML avec la balise **<point nom="A" X="X0" Y="Y0"/>**.

2. Ecrire une classe **Polygone** représentant un polygone par un tableau d'objet de type Point. Cette classe comportera les méthodes suivantes :

- Un constructeur permet de construire un polygone à partir du tableau de ses sommets, si le nombre de sommets est inférieur ou égale à 2 il faut lever une exception **InvalideNombreSommetException** (Cette exception de type **Exception** est à définir).
- **aire()** : calcul de la surface du polygone. Sachant que l'aire S d'un polygone ayant les sommets $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots, (x_{n-1}, y_{n-1})$ est :

$$S = \frac{1}{2} \times |(x_0 + x_1) \times (y_0 - y_1) + (x_1 + x_2) \times (y_1 - y_2) + \dots + (x_{n-2} + x_{n-1}) \times (y_{n-2} - y_{n-1}) + (x_{n-1} + x_0) \times (y_{n-1} - y_0)|$$

- **getSommetMaxX()** : permet d'obtenir le point d'abscisse maximale du polygone. Au cas où il y en aurait plusieurs, on retournera celui ayant la plus petite ordonnée.
 - Une méthode abstraite **getXmlPresentation()** destinée à être implémentée dans les classes filles pour donner la représentation XML de chaque forme géométrique héritant d'un polygone.
3. Définir une classe **Triangle** qui représente un triangle, sous-classe de **Polygone**, avec un constructeur

Triangle(Point a, Point b, Point c) qui permet de construire un triangle ayant les trois sommets indiqués.

Donner également une implémentation de la méthode **getXmlPresentation**, sachant qu'un triangle ayant les sommets A(X0,Y0), B(X1,Y1) et C(X2,Y2) se présente par le code XML suivant :

```
<triangle>
  <point nom="A" X="X0" Y="Y0"/>
  <point nom="B" X="X1" Y="Y1"/>
  <point nom="C" X="X2" Y="Y2"/>
</triangle>
```

4. Ecrire une classe ***RectangleHorizontal*** qui représente un rectangle horizontal, sachant qu'un rectangle est un polygone, La classe ***RectangleHorizontal*** doit hériter de la classe ***Polygone***. Le constructeur de la classe ***RectangleHorizontal*** doit prendre trois paramètres, le premier est un objet de type Point et représente le coin inférieur gauche du rectangle, le second est de type double et représente la longueur du rectangle, le troisième est de type double et représente la largeur du rectangle.

La classe ***RectangleHorizontal*** dispose également d'une implémentation de la méthode ***getXmlPresentation*** mais il n'est pas demandé de l'écrire dans cet exercice. Cette méthode pourra être utilisée dans la suite de l'exercice.

5. On suppose à présent qu'on veut réaliser des dessins de constructions (immeubles, bâtiments, maisons, mosquées,...) à l'aide de polygones (cf. figure 5), ainsi on définit la classe ***Construction*** qui se compose d'une collection de polygones.

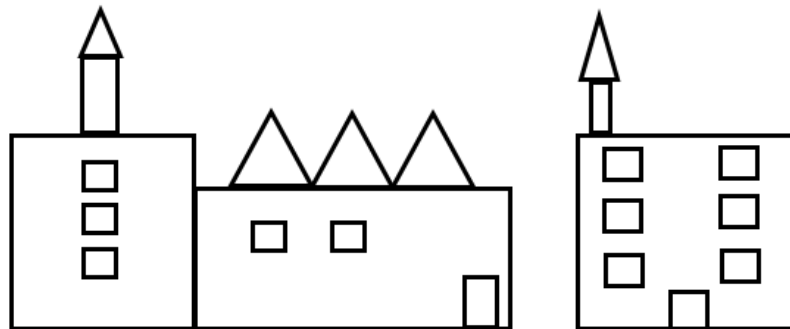


Figure 5 : Exemples des dessins de constructions réalisées à l'aide des polygones

Ecrire la classe ***Construction*** qui dispose des méthodes suivantes :

- Un constructeur permettant d'initialiser la collection des polygones constituant la construction
- ***addPolygone*** : Ajoute un polygone dans la collection des polygones constituant la construction

- **getXmlPresentation** permettant de donner une représentation XML d'une construction, en respectant la syntaxe des polygones et en la complétant si nécessaire.

```
<desin>
...
<triangle>
  <point nom="A" X="X0" Y="Y0"/>
  <point nom="B" X="X1" Y="Y1"/>
  <point nom="C" X="X2" Y="Y2"/>
</triangle>
...
<rectangleHorizontal>
  <point nom="D" X="X3" Y="Y3"/>
  <point nom="E" X="X4" Y="Y4"/>
  <point nom="F" X="X5" Y="Y5"/>
</rectangleHorizontal>
...
</desin>
```

6. Faire les modifications nécessaires dans le code pour ajouter les fonctionnalités suivantes :
 - Sauvegarder la représentation XML d'une forme dans un fichier
 - Lire et initialiser une forme à partir de son fichier XML

3- Accès aux données avec Entity Framework

Dans cet exercice on propose d'écrire un programme de gestion des QCM. Ci-dessous la liste des fonctionnalités demandées :

- Génération d'un QCM de *n* questions. Les questions sont tirées à partir de la base de données aléatoirement.
- Enregistrement des réponses de l'utilisateur : pour chaque question le programme stocke la réponse de l'utilisateur
- Affichage de la moyenne sur 10 à la fin de la réponse au QCM
- Enregistrement des scores d'un utilisateur
- Authentification des utilisateurs

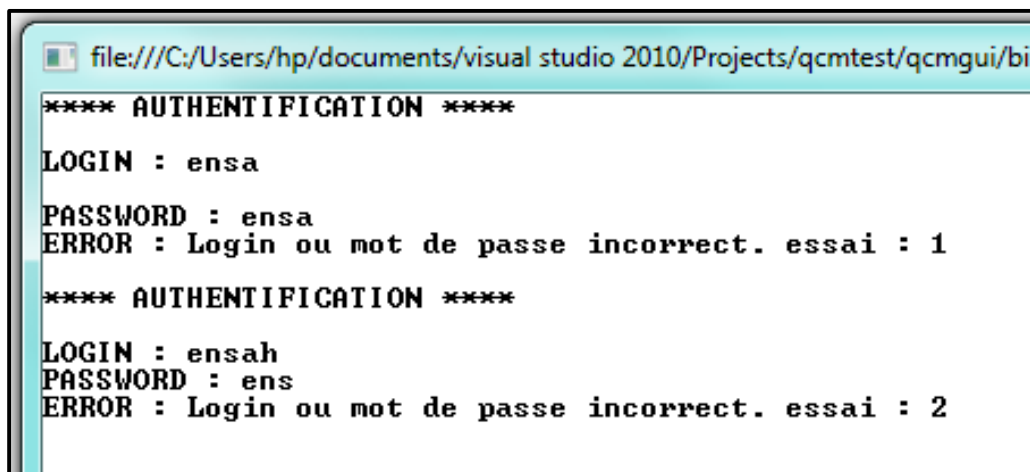
Le programme fonctionne en mode console, mais pour garantir l'évolutivité de ce programme, vous devez décomposer votre programme aux moins en deux couches :

- Couche IHM présentée par le programme console
- Le noyau du programme qui regroupe les dao et les objets métiers.

Pour la gestion des accès aux données on propose d'utiliser Entity Framework et LINQ. (Cf. quelques rappels à la fin de l'exercice)

Ci-dessous une description par des captures d'écran des scénarios d'exécution de ce programme :

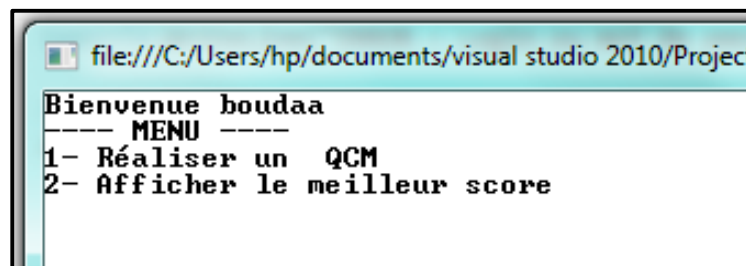
- 1- Authentification : la figure 6 montre le comportement du programme dans le cas d'échec de l'authentification :



```
file:///C:/Users/hp/documents/visual studio 2010/Projects/qcmtest/qcmgui/bi
**** AUTHENTIFICATION ****
LOGIN : ensa
PASSWORD : ensa
ERROR : Login ou mot de passe incorrect. essai : 1
**** AUTHENTIFICATION ****
LOGIN : ensah
PASSWORD : ens
ERROR : Login ou mot de passe incorrect. essai : 2
```

Figure 6 : comportement du programme dans le cas d'échec de l'authentification

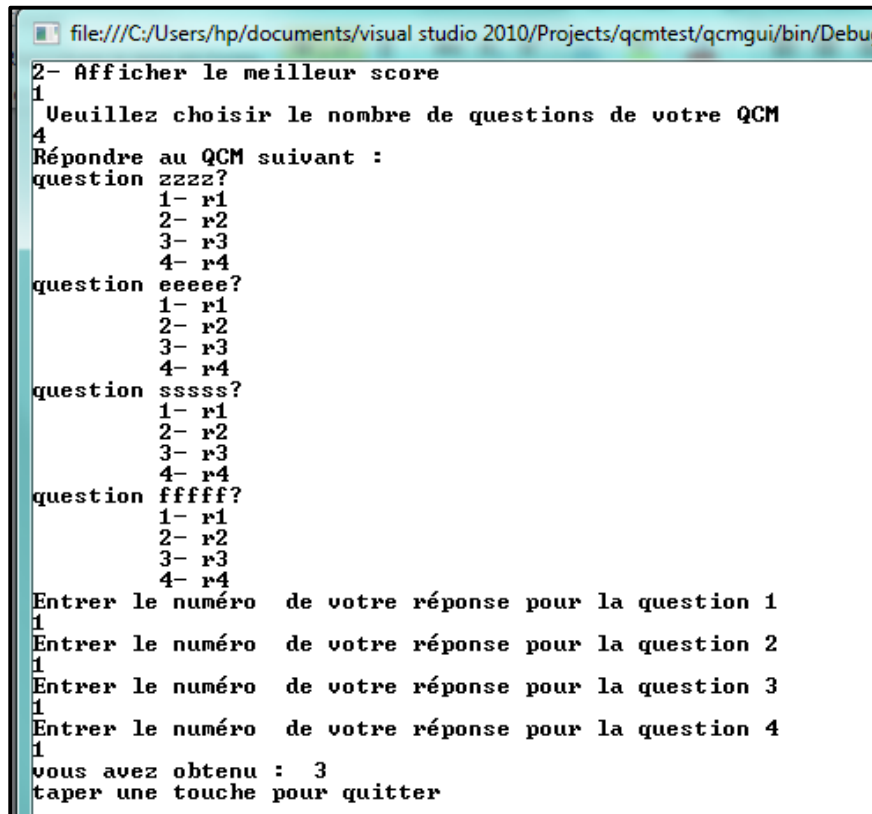
- 2- En cas de succès de l'authentification le programme affiche un message de bienvenue et le menu du programme. (Cf. figure 7)



```
file:///C:/Users/hp/documents/visual studio 2010/Project
Bienvenue boudaa
----- MENU -----
1- Réaliser un QCM
2- Afficher le meilleur score
```

Figure 7 : comportement du programme en cas de succès de l'authentification

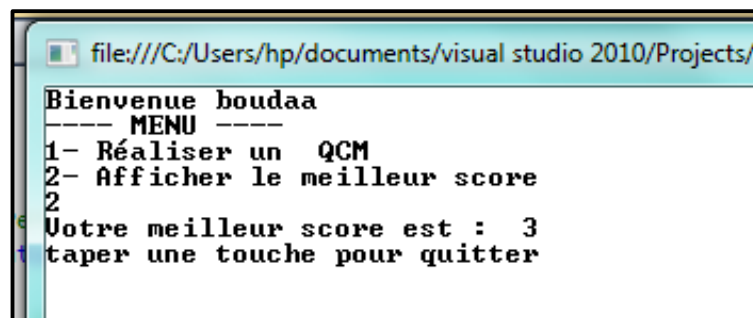
- 3- Réalisation d'un QCM : Si l'utilisateur a choisi l'option « réaliser un QCM » le programme lui demande le nombre de questions qu'il souhaite avoir dans le QCM puis il lui génère un QCM aléatoirement de la base de données. Ensuite le programme lit les réponses de l'utilisateur et à la fin il affiche la moyenne (le score) obtenue. (Cf. figure 8)



```
file:///C:/Users/hp/documents/visual studio 2010/Projects/qcmtest/qcmgui/bin/Debug
2- Afficher le meilleur score
1
Veuillez choisir le nombre de questions de votre QCM
4
Répondre au QCM suivant :
question zzzz?
1- r1
2- r2
3- r3
4- r4
question eeeee?
1- r1
2- r2
3- r3
4- r4
question sssss?
1- r1
2- r2
3- r3
4- r4
question fffff?
1- r1
2- r2
3- r3
4- r4
Entrer le numéro de votre réponse pour la question 1
1
Entrer le numéro de votre réponse pour la question 2
1
Entrer le numéro de votre réponse pour la question 3
1
Entrer le numéro de votre réponse pour la question 4
1
vous avez obtenu : 3
taper une touche pour quitter
```

Figure 8

- 4- La figure ci-dessous montre le comportement du programme si l'utilisateur choisit l'option « Afficher le meilleur score »



```
file:///C:/Users/hp/documents/visual studio 2010/Projects/qcmtest/qcmgui/bin/Debug
Bienvenue boudaa
---- MENU ----
1- Réaliser un QCM
2- Afficher le meilleur score
2
Votre meilleur score est : 3
taper une touche pour quitter
```

Figure 9

Rappels :

Générer un entier aléatoire entre 0 et 10 :

```
Random random = new Random();
```

```
int i = random.Next(0, 10);
```

Etablir une connexion à la base de données

```
string chaineConnexion =
```

```
ConfigurationManager.ConnectionStrings["EntityManager"].ConnectionString;
```

```
EntityManager em = new EntityManager(chaineConnexion);
```

```
Ou : EntityManager em = new EntityManager();
```

Accès aux données d'une table :

```
em.QCMs (récupère les données de la table QCMs)
```

Exemple de parcours des données :

```
foreach ( QCM qcm in em.QCMs)
```

```
{
```

```
Console.WriteLine ("{0} ({1})", qcm.score , qcm.date);
```

```
}
```

Exemple de requête LINQ to Object

Le code ci-dessous se base sur LINQ pour récupérer la liste des personnes ayant le nom « name »

```
IEnumerable<Person> persons = from person in em.PersonSet
```

```
where person.nom.Equals(name)
```

```
select person;
```

Convertir un IEnumerable IE en une List list :

```
List<X> list = IE.ToList<X>();
```

Manipulation des données : (Il y a des différences légères entre les versions des Framework, vérifiez la documentation)

- Ajout des données : on utilise la méthode AddObject / Ou Add puis em.SaveChanges
- Suppression des données : DeleteObject / Ou Delete puis em.SaveChanges().
- Mise à jour des données : modification des objets puis em.SaveChanges().