|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M:\ADMIN\henallux_montgolfiere.png  Implantation IESN | C:\Users\DubyLocal\Documents\ABRacine\Coordination\Logo Henallux\IG-v1.1-R1.jpg**Programmation orientée objets**  Section IG, 2e année — Examen — janvier 2022 | | | | |
| *Nom et prénom* | | | | / 20 |
|  | | | |
| *Bloc* |  | *Groupe* |  |

|  |  |  |
| --- | --- | --- |
| **Consignes**   * Lisez les énoncés attentivement et respectez les consignes indiquées ! * Dans votre code, utilisez avec pertinence les outils vus au cours et les particularités du langage C# (types prédéfinis adéquats, propriétés…) et respectez les « bons principes » de la programmation orientée objets. * Pour répondre à l’examen :  |  |  | | --- | --- | | Remplissez les cadres bleutés | Laissez les cadres roses/orangés tels quels : ils sont destinés à la correction ! |  * Complétez la déclaration sur l’honneur ; sans ça, votre questionnaire sera non valide ! |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Principes de l’orienté objet [POIDS = 2]*** | | | | | | |
| TRÈS INSUFFISANT  max 6/20 | INSUFFISANT -2 | JUSTE SUFFISANT 0 | SATISFAISANT +2 | BIEN ACQUIS +4 | | |
| *Mettre en pratique les principes d’encapsulation, d’héritage et de polymorphisme de l’orienté objet standard.* | | | | | | |
| ***Bases du langage et pragma C# [POIDS = 1.5]*** | | | | | | |
| TRÈS INSUFFISANT  max 6/20 | INSUFFISANT -1.5 | JUSTE SUFFISANT 0 | SATISFAISANT +1.5 | BIEN ACQUIS +3 | | |
| *Savoir utiliser la syntaxe de base du langage C# et en respecter les conventions.* | | | | | | |
| ***Éléments avancés du langage C# [POIDS = 1.5]*** | | | | | | |
| TRÈS INSUFFISANT  max 6/20 | INSUFFISANT -1.5 | JUSTE SUFFISANT 0 | SATISFAISANT +1.5 | BIEN ACQUIS +3 | | |
| *Savoir manipuler les classes génériques, les collections et la syntaxe LINQ (version méthodes).* | | | | | | |
| ***Clean code*** | | | | | | |
| TRÈS INSUFFISANT  max 6/20 | INSUFFISANT *pénalité* | SUFFISANT |  |  | | |
| ***Qualité algorithmique du code*** | | | | | | |
| TRÈS INSUFFISANT  max 6/20 | INSUFFISANT *pénalité* | SUFFISANT |  |  | | |
| Cote finale (valeur ***indicative***) = 10 avec ajustements indiqués ci-dessus. Si au moins un TRÈS INSUFFISANT : cote maximale de 6/20. | | | | | | |
| ***Commentaires*** | | | | | | |
|  | | | | | | |
| PARTIE 0 (Déclaration sur l’honneur) nécessaire pour la validité de l’examen | | | | |  |

En participant à cet examen et en soumettant ma copie pour correction, moi,

*(prénom et nom)*

|  |
| --- |
|  |

m’engage à :

* ne pas partager mon questionnaire ou mes réponses avec une autre personne[[1]](#footnote-1) ;
* ne pas aider un autre étudiant participant au même examen à répondre à son questionnaire ;
* ne pas consulter un autre questionnaire du même examen ni solliciter l’aide d’une autre personne1 pour répondre à mon questionnaire.

|  |  |
| --- | --- |
| **PARTIE 1 (Éléments avancés de C#)** |  |

Répondez aux questions suivantes en utilisant des **collections** et des **fonctions de haut niveau** (LINQ).

## QUESTION 1

On dispose d’une classe **Langue** dont chaque objet a deux propriétés en lecture : **Code** et **Nom** (par exemple l’objet représentant le français a pour code "FR" et pour nom "Français").

On dispose également d’une classe **Employé** dont chaque objet possède (au moins) les propriétés **Nom**, **Prénom**, etc.

On désire une classe **Mémo** permettant de représenter des mémos (ou messages envoyés entre les employés d’une compagnie). Chaque mémo possède les attributs suivants :

1. un **identifiant** numérique ;
2. l’**auteur** du mémo (un employé) ;
3. la **date** du mémo (via DateTime) ;
4. les **destinataires** du mémo (qui sont les employés auxquels le mémo est envoyé, dans un ordre relatif à la priorité des destinataires) ;
5. les **mots-clefs** du mémo (qui sont des chaînes de caractères reprenant les thèmes principaux du mémo ; par exemple : "base de données", "projet Absalom", "UML", sans ordre précis et sans doublon) ;
6. le **contenu** du mémo, sachant que le texte d’un mémo peut être disponible en plusieurs langues ; cet attribut retiendra non seulement les langues dans lesquelles le mémo est disponible mais aussi, pour chacune d’elles, le contenu du mémo dans cette langue.

Écrivez les déclarations (avec des types explicites et en utilisant les collections les plus adaptées) pour chacun des attributs de la classe **Mémo**.

|  |
| --- |
| collez votre code ici (police Consolas taille 11 SANS fond noir)  private int identifiant;  private Employé auteur;  private DateTime date;  List<Employé> destinataires;  HashSet<string> mots\_cléfs;  Dictionary<Langue,string> contenu; |
|  |
|  |

## QUESTION 2

On suppose qu’on se trouve à l’intérieur de la classe **Mémo**, à un endroit où tous les attributs cités dans la question 1 sont accessibles. Sachant qu’on dispose de la variable **mémos**, qui est une collection de tous les mémos envoyés ce mois-ci, ordonnés par ordre chronologique, écrivez les bouts de code permettant :

1. d’afficher le nombre de mémos comportant "projet Absalom" parmi ses mots-clefs ;

|  |
| --- |
| Console.WriteLine(mémos.Count(x => x.Mots\_cléfs.contains("projet Absalom"))); |
|  |

1. d’afficher le nombre de destinataires maximal en considérant tous les mémos envoyés ce mois‑ci ;

|  |
| --- |
| Console.WriteLine(mémos.Max(m => m.Destinataies.Count())) |
|  |

1. d'afficher l’identifiant numérique du mémo le plus récent envoyé par l’employé correspondant à l’objet **employé** (une variable déjà garnie).

|  |
| --- |
| collez votre code ici (police Consolas taille 11 SANS fond noir) |
|  |

1. de stocker dans la variable **mémosFR** les mémo pour lequel une version FR existe ;

|  |
| --- |
| collez votre code ici (police Consolas taille 11 SANS fond noir) |
|  |

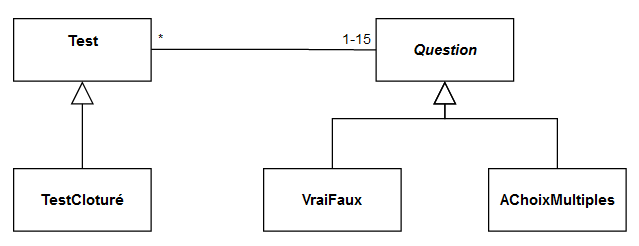
1. de stocker dans la variable **auteursFR** les nom des auteurs (sans répétition) ayant envoyé au moins un mémo pour lequel une version FR existe.

|  |
| --- |
| collez votre code ici (police Consolas taille 11 SANS fond noir) |
|  |

|  |  |
| --- | --- |
| PARTIE 2 (Principes de l’orienté objet et bases de C#) |  |

## CONTEXTE

Dans le cadre d’une application d’évaluation en ligne (simplifiée), on veut pouvoir créer un test avec des questions de type vrai ou faux, des questions à choix multiples, des questions à réponse courte, des questions d’appariement… Voici le diagramme de classes qui reprend les classes utilisées dans la suite des questions.



**Class Question**. Pour chaque question, on prévoit un énoncé sous la forme d’une chaîne de caractères et le nombre de points attribué à la question (1 par défaut), éléments communs à toutes les questions.  
On prévoit également une réponse, qui dépend du type de la question. Celle-ci doit pouvoir être récupérée sous la forme d’une chaine de caractères, quel que soit le type de la question.

Toutes les questions sont d’un type spécifique : vrai ou faux, question à choix multiples, réponse courte, appariement, etc. Il n’y a pas de question sans type. Pour chaque type de question, on définit une sous-classe à **Question**. Dans le cadre de cet exercice, on ne s’intéresse qu’à deux sous-classes : **VraiFaux** et **AChoixMultiples**.

**Classe VraiFaux.** Pour une question du type vrai ou faux, on doit mémoriser la réponse sous la forme d’un booléen.   
Une méthode permet d’afficher la question suivie des propositions vrai ou faux comme ci-dessous.  
Une image contenant texte

Description générée automatiquement

**Classe AChoixMultiples.** Pour une question à choix multiples, on doit mémoriser la réponse sous la forme d’un caractère, celui correspondant à la lettre affichée en regard de la proposition correcte.   
En plus de la réponse, il faut fournir la liste de propositions contenant notamment cette réponse.  
Une proposition est une simple chaîne de caractères.   
Une méthode permet d’afficher la question suivie des propositions comme ci-dessous.  
Une image contenant texte

Description générée automatiquement

## CODE PROPOSÉ.

Voici le code proposé (par un débutant) pour les classes **Question** et **VraiFaux**. Ce code intervient dans les question qui suivent.

class Question

{

private string énoncé;

private int nbPoints;

public string Réponse {

get

{

return "Réponse bidon";  
 }

}

public string Énoncé { get; set; }

public Question(string énoncé, int nbPoints)

{

this.énoncé = énoncé;

this.nbPoints = nbPoints;

}

public string toString()

{

return énoncé;

}

}

class VraiFaux : Question

{

private bool réponse;

public string Réponse { get; }

public VraiFaux(string énoncé, int nbPoints, bool réponse)

{

this.énoncé = énoncé;

this.nbPoints = nbPoints;

this.réponse = réponse;

}

public string toString()

{

return base.toString() + "\no Vrai" + "\no Faux";

}

}

## QUESTION 1

1. Dans le code proposé ci-avant, on s’intéresse à la ligne   
   **public string Énoncé { get ; set ; }** de la classe Question.   
   Indiquez brièvement ce qu’elle signifie et permet, et précisez son rapport (s’il y en a un) avec la ligne **private string énoncé;**.

|  |
| --- |
|  |
|  |

1. Si on vous donne le code ci-avant à « reviewer », quel(s) commentaire(s) feriez-vous ?   
   On vous demande ici de critiquer le code point par point (de façon claire et organisée) en indiquant ses problèmes, pas juste de donner une correction (ce sera pour la question suivante).

|  |
| --- |
|  |
|  |

## QUESTION 2

1. Corrigez le code des classes **Question** et **AChoixMultiples** proposées ci-dessous. L’objectif ici est de garder les fonctionnalités de base mais de potentiellement revoir en profondeur les structures et implémentations choisies.
2. Définissez en plus les trois classes suivantes (de sorte qu’elles soient compatibles avec vos versions corrigées de **Question** et de **AChoixMultiples**).

* La classe **AChoixMultiples** dont la description se trouve ci-avant.
* La classe **Test** permet de représenter un test avec son intitulé (sous la forme d’une chaîne de caractères) et la liste des questions qui le composent. Un test ne peut comporter qu'un maximum de 15 questions toutes différentes (ce nombre, 15, devant être encodé de manière à être facilement modifié).  
  Lors de la création d’un test, seul son intitulé est fourni. En effet, c’est après avoir généré les questions qu’on peut les ajouter au test, soit une à une, soit toutes d’un coup. Un méthode est donc à prévoir pour ajouter une ou plusieurs questions.   
  Une autre méthode doit permettre d’afficher l’intitulé du test suivi des questions qui le composent avec les éventuelles propositions.  
  Enfin, comme à chaque questions est associé un nombre de points, il doit être possible de récupérer le nombre total de points du test.
* Au bout d’un moment, l’enseignant peut mettre fin au test qui devient un **TestCloturé**, un cas particulier de test. Un test clôturé contient les mêmes informations qu’un test, plus une date de clôture, date à partir de laquelle on peut demander la correction du test ainsi que le résultat obtenu. Cette date est initialisée dans le constructeur et ne pourra pas être modifiée par la suite.  
  Pour connaître le résultat (pourcentage) du test, il faut d’abord déterminer les points obtenus au test par l’utilisateur. Une méthode est donc à prévoir à cet effet. Elle reçoit la liste des réponses fournies par l’utilisateur sous la forme de chaînes de caractères ("a" si c’est une question à choix multiple et que la réponse est en a, "vrai" si c’est un vrai ou faux et que la réponse est true…). Lorsque la réponse donnée est correcte, les points de la question sont ajoutés au nombre de points obtenus (son score).  
  C’est sur base de ces points que le résultat est déterminé. Le résultat est exprimé sous la forme du pourcentage des points obtenues par rapport au nombre total de points du test.

## Remarques

* Choisissez les collections les plus adaptées chaque fois que c’est pertinent.
* Vous devrez peut-être ajouter des fonctionnalités non mentionnées aux classes créées.
* Les règles générales du Clean Code restent bien sûr de mise !
* Appliquez les conventions propres au C#.
* Veillez à l’efficacité de votre code.

|  |
| --- |
| collez votre code ici (police Consolas taille 11 SANS fond noir) |
|  |

1. À l’exception des enseignants en charge de l’examen en question. [↑](#footnote-ref-1)