TEK-UP
Ecole Supérieure Privée
Technologie & Ingénierie

Ecole Supérieure Privée Technologies & Ingénierie

Type d'épreuve	: ☐ Devoir	■ Examen
Enseignant	:	
Matière	: Architecture N-tiers avec .Net	
Année Universitaire	: 2021-2022	**Semestre** : 1
Classe	:	
Documents	: ☐ Autorisés	■ Non autorisés
Date	:	**Durée :** 1h30mn
Nombre de pages	: 7	
Barème	: xx	

Exercice 1 (6pts) : Cocher la ou les bonnes réponses

1) En C#, Une variable de classe, commune à toutes les instances d'une classe doit être déclarée
   A. Static ✕
   B. global
   C. public
   D. private

2) En C#, une classe ne peut pas hériter de plusieurs classes mais peut hériter à plusieurs niveaux
   A. Vrai ✕
   B. Faux

3) Dans l'approche Code-First,
   A. Entity Framework crée des classes à partir d'une base de données existante
   B. Entity Framework crée la base de données à partir des classes ✕

4) En ASP.Net MVC, MVC désigne :
   A. Model, Vision, Control
   B. Model, View, Controller ✕
   C. Model, ViewData, Controller

5) Le mot clé "base" en C# désigne :
   A. La classe fille
   B. La classe mère ✕
   C. Un namespace

6) Lequel des énoncés suivants est vrai :
   A. Le contrôleur redirige la requête entrante vers le modèle
   B. Le contrôleur exécute une requête entrante ✕

C. Le contrôleur contrôle les données

Exercice 2 (8 pts) :

On souhaite développer une application ASP.NET MVC avec le framework .Net pour la gestion des employés.

1. C'est quoi MVC ? Quelle est sa particularité ?

C'est le patron de conception Modèle, vue, contrôleur permettant de séparer les préoccupations entre les 3 couches présentation, métier l'accès aux données.

……

2. Décrire brièvement la fonction de chaque couche du MVC.

Couche Modèle : les données de l'application.

Couche Vue : les interfaces utilisateurs de l'application.

Couche Contrôleur : exécute les requêtes entrantes en liant la couche modèle à la couche vue.

……

3. On commence par concevoir la couche modèle du projet. Un employé est caractérisé par sa Matricule, son Nom, son Prénom et sa Date de naissance.
   Implémenter la classe Employé.

……

```
public class Employe
{
 public int Matricule { get; set; }
 public string Nom { get; set; }
 public string Prenom { get; set; }
 public DateTime Data_Naiss { get; set; }
}
```

4. On compte maintenant créer une classe de contrôleur lié au modèle Employé nommé « EmployeController ». Cette classe est constituée d'une action (méthode) appelée « Display ». Au niveau de cette dernière, on créera une liste de 5 employés et retournera finalement cette liste à la vue correspondante à cette action (Display).
   Remplissez les pointillés du code ci-dessous pour implémenter le contrôleur décrit précédemment.

```
public class …… EmployeController … : ……Controller….
    {

        public ActionResult Display()
        {
            List<Employe> employes = new List< Employe >()
             {
            new Employe{Matricule=1, Nom ="emp1", Prenom ="emp11",
            Data_Naiss=  new DateTime(1998,04,30) },
            new Employe {Matricule=2, Nom ="emp2", Prenom ="emp22",
            Data_Naiss=  new DateTime(1998,04,30) },
            new Employe {Matricule=3, Nom ="emp3", Prenom ="emp33",
            Data_Naiss=  new DateTime(1998,04,30) },
            new Employe {Matricule=4, Nom ="emp4", Prenom ="emp44",
            Data_Naiss=  new DateTime(1998,04,30) },
            new Employe {Matricule=5, Nom ="emp5", Prenom ="emp55",
            Data_Naiss=  new DateTime(1998,04,30) },

             };
            return View(…employes);
    }
   }
```

5. ASP.NET MVC invoque les différents contrôleurs et leurs différentes actions selon l'URL entrante. Dans quel fichier définit-on-le format des routes (URLs) correspondantes aux actions du contrôleur ?

……RouteConfig.cs……
……

6. On compte rajouter une nouvelle action au niveau du contrôleur EmployeController nommée «EditEmploye » permettant de modifier la date de naissance d'un employé. Rajouter une route au niveau du fichier RouteConfig pour inclure l'URL suivante :

« Employees/modifier/2020/03 », ayant comme arguments month et year. Remplissez les pointillés dans le code ci-dessous pour mapper l'url de cette action avec le format désiré.

```
public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                ".....Modifier.....",
                "..... Employees/modifier/{year}/{month}"
                   new { Controller = "Employe…", Action="EditEmploye……"}
   );

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index",
id = UrlParameter.Optional }
            );
        }
    }
```

7. Maintenant, on veut sauvegarder les données de notre application dans une base de données. Proposer une technique permettant de faire le mappage entre les objets et la base de données.

……Data Entity Framework ……
……

8. Mentionnez les 3 approches (workflow) qu'on peut les utiliser avec cette technique.

Database first

Code First

Model First

Exercice 3 (6 pts) :

On souhaite créer une application pour la gestion des comptes bancaires. Un compte bancaire est caractérisé par un code incrémenté automatiquement et un solde. Le solde peut être positif (compte créditeur) ou négatif (compte débiteur). A sa création, un compte bancaire a un solde nul et un code incrémenté. Il est aussi possible de créer un compte en précisant son solde initial.
Pour Utiliser son compte, on doit être capable de faire des dépôts et des retraits. Pour ces deux opérations, il faut connaître le montant de l'opération. L'utilisateur peut aussi consulter le solde de son compte par la méthode ToString().

1. Implémenter la classe Compte.

```
/***** Classe Compte ******/
class Compte
    {
        private int code;
        private double solde;
        private static int nb_comptes = 0;

        public int Code
        {
            get { return code; }
        }

        public double Solde
        {
            get { return solde; }
        }

        public static int Nb_Comptes
        {
            get { return nb_comptes; }
        }

        public Compte()
        {
            nb_comptes++;
            code = nb_comptes;
            solde=0 ;
        }

        public Compte(double solde)
        {
            nb_comptes++;
            code = nb_comptes;
            this.solde = solde;
        }


        public virtual void deposer(double somme)
//méthode virtuelle qui peut être redéfinie dans une classe dérivée
        {
            solde += somme;
        }


        public virtual void retirer(double sommme)
////méthode virtuelle qui peut être redéfinie dans une classe dérivée
        {
            solde -= sommme;
        }
```

```
        public override string ToString()
        {
            return "Code: " + code + " Solde: " + solde;
        }
    }
```

2. Un compte Epargne est un compte bancaire qui possède en plus un champ «TauxIntérêt=6 » et une méthode calculIntérêt() qui permet de mettre à jour le solde en tenant compte des intérêts. Implémenter la classe CompteEpargne.

```
/***** Classe CompteEpargne ******/

class CompteEpargne : Compte
    {
        private double tauxinteret = 6;

        public double Tauxinteret
        {
            get { return tauxinteret; }
        }

        public CompteEpargne() : base() { }         //constructeur par
défaut

        public CompteEpargne(double solde) : base(solde) { }
//constructeur d'initialisation

        public void CalculerInteret()  //une nouvelle méthode qui
utilise une méthode héritée
        {
            deposer((Solde * tauxinteret) / 100);
        }

        public override string ToString()    //redéfinition de la
méthode ToString()
        {
            return "Compte Epargne: " + base.ToString() + " Taux
interêt: " + tauxinteret;
        }
    }
```

……

3. Un ComptePayant est un compte bancaire pour lequel chaque opération de retrait et de versement est payante et vaut 5 DH. Implémenter la classe ComptePayant.

```
class ComptePayant : Compte
    {
        public ComptePayant() : base() { }                //Constructeur
par défaut
        public ComptePayant(double solde) : base(solde) { }
//Constructeur d'initon
        public override string ToString()
//ToString redéfinie
        {
            return "Compte Payant: " + base.ToString();
        }

        public override void deposer(double somme)   //La méthode
"deposer" redéfinie
        {
            base.deposer(somme);
            base.retirer(5);
        }

        public override void retirer(double somme)    //La méthode
"retier" reféfinie
        {
            base.retirer(somme);
            base.retirer(5);
        }
    }
```

Bon Travail 😊