OFPPT
مكتب التكوين المهني وإنعاش الشغل
Office de la Formation Professionnelle et de la Promotion du Travail
Direction Recherche et Ingénierie de la Formation

Examen de passage à la 2eme année

Session Juin 2016

Filière : Techniques de Développement Informatique ***Epreuve : Synthèse***
Niveau: TS ***Variante : V1***
Durée : 5 heures ***Barème : / 120***

❖ Partie I : Théorie (40 pts)

➢ Dossier 1: Notions de mathématiques appliquées à l'informatique (12 pts)

1. Convertir en binaire les nombres suivants *(06 pts)*

   $(145)_8$; $(A4BE)_{16}$; $(59)_{10}$

2. Effectuer en binaire l'opération suivante *(02 pts)*

   1011110*11

3. A l'aide du tableau de Karnaugh, simplifier la fonction H définie par sa table de vérité suivante : **(*04* pts)**

E	F	G	H
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

➢ Dossier 2: Techniques de programmation structurée (8 pts)

Ecrire un algorithme qui demande à l'utilisateur de saisir deux tableaux d'entiers **T1** et **T2** de dimensions respectives **A** et **B**, et retourne une matrice d'entiers de **B** lignes sur **A** colonnes où chaque case de coordonnées **i, j** est le résultat de produit **T1[i]** avec **T2[j]** avec le maximum des deux tableaux. (Voir exemple ci-dessous.)

Exemple :

T1

3	7	4	8

T2

4	10	7	3	8

Le maximum des deux tableaux est 10 :
Le résultat est :

	3	7	4	8
4	4*3*10=120	4*7*10=280	160	320
10	300	700	400	800
7	21	490	280	56
3	9	210	120	24
8	240	560	320	64

➢ Dossier 3: Conception et modélisation d'un système d'information (20 pts)

Ce système doit permettre aux utilisateurs de saisir leurs **charges de déplacement** dans un logiciel de gestion afin d'accélérer le traitement de remboursement des charges de déplacement.

Les dépenses des salariés en déplacement peuvent être de types train, taxi, hôtel, ou voiture.

Pour tous types de dépenses, le salarié doit renseigner la date de départ, la date de retour, le libellé de la mission, le lieu de destination, montant avec la prise en compte des plafonds par type de dépense, et la gestion des commentaires expliquant des dépenses exceptionnelles (dépassement de plafond, impondérables...).

Le salarié est identifié par un numéro de matricule, nom, prénom, service, département, et date de recrutement.

Chaque charge de déplacement est identifiée par un numéro unique pour toutes réclamations.

Les dépenses des salariés en déplacement avec les voitures personnelles doivent renseigner la marque, le nombre de chevaux, type de carburant et le numéro de la plaque.

Chaque charge de déplacement doit être validée par le responsable hiérarchique directe et par le directeur de la société.

Le système doit permettre aux salariés de gérer des relances pour garantir des délais de traitement.

Travail demandé.

1. Etablir le dictionnaire de données. ***(06 pts)***
2. Etablir le modèle conceptuel de données correspondant. ***(08 pts)***
3. Etablir le modèle logique de données associé. ***(06 pts)***

❖ Partie II: Pratique (80 pts)

➢ Dossier 1: Langage de programmation structurée (25 pts)

La société souhaite représenter sous forme d'un tableau **les charges de déplacement** de ses salariés.

Chaque enregistrement de type **chargeDéplacement** est composé d'un identifiant, durée, libellé, lieu, montant et statut.

1- Définir une structure **chargeDéplacement** pouvant contenir ces informations. **(2 pts)**

2- Définir un tableau de structure ***HistoriqueChargeDéplacement*** de type **chargeDéplacement** permettant de représenter l'ensemble des charges de déplacement des salariés. (utiliser un tableau de taille maximale 20). **(1 pt)**

3- Ecrire un sous-programme qui permet de saisir un certain nombre de **chargeDéplacement** dans le tableau structuré ***HistoriqueChargeDéplacement***. **(3 pts)**

4- Ecrire un sous-programme qui permet d'afficher toutes les **charges de déplacement** de ***HistoriqueChargeDéplacement***. **(3 pts)**

5- Ecrire un sous-programme qui permet d'afficher les informations (identifiant, duree, mission, lieu, montant, statut) des **charges de déplacement** ayant le statut «***en cours***». **(3 pts)**

6- Ecrire un sous-programme qui permet de calculer le montant global des charges de déplacement. **(3 pts)**

7- Ecrire un sous-programme qui permet de Modifier le montant d'une **charge de déplacement** correspondant à un numéro donné. **(3 pts)**

8- Demander à l'utilisateur de saisir les informations d'une **charge de déplacement**. Ajouter ensuite cette **charge de déplacement** au tableau ***HistoriqueChargeDéplacement***. L'ajout sera fait dans le bon endroit selon le numéro identifiant (Pour avoir un ordre par numéro identifiant). **(4 pts)**

9- Créer un menu pour appeler les sous-programmes précédents. **(3 pts)**

Dossier 2: Langage de programmation Orientée Objet (30 pts)

On souhaite informatiser la gestion des **charges de déplacement**.

Les classes suivantes sont déjà créer :

```
class Salarie
    {
        public int identifiant;
        public string Nom ;
        publi  string Prenom ;
        public string Adresse ;
        public string Genre;
        public float Age ;
        public string service  ;
        public string departement;
        public string Ville ;
        public Salarie() { }
        public Salarie(int id, string nom, string prenom, string adresse, string
Genre, float Age, string service, string dep,string v)
        {       this.identifiant = id;          this.Nom = nom;
                this.Prenom = prenom;           this.Adresse = adresse;
                this.Genre = Genre;             this.Age = Age;
                this.service = service;         this.departement = dep;
                this.Ville = v;
        }

        public override string ToString()
        {
            return "id:"+ this.identifiant  + " Nom\n" + this.Nom + "Prenom\n" +
this.Prenom + "Adresse:"+ this.Adresse + " Genre\n" + this.Genre + "Age\n"
+this.Age+ "service:"+ this.service + "Departement\n" + this.departement  ; "Ville
: " + this.Ville} }
```

```
class Dépense
    {

          public int     Numero;
          public string  Libellé;
          public string  Lieu;
          public string  Commentaire;
          public float   Montant;
        public Dépense() { }
        public Dépense(int num, string libelle, string lieu, string commentaire,
float montant)
        {

            this.Numero = num;
```

```
            this.Libellé = libelle;
            this.Lieu = lieu;
            this.Commentaire = commentaire ;
            this.Montant = montant ;
        }
public virtual int CalculerCharge ()
        {            return montant * taux ;        }

        public override string ToString()
        {
            return "Numero:" + this.Numero + " Libellé\n" + this.Libellé + "Lieu\n"
+ this.Lieu + "Commentaire:" + this.Commentaire + " Montant\n" + this.Montant;
        }
    }
```

Nous aurons besoin des classes suivantes :

1. Classe **ChargeDéplacementVoiture**:
   a. Créer une classe **ChargeDéplacementVoiture** qui hérite de la classe **Dépense** et caractérisée par : **(2 pts)**
      - Salarié : de type salarié.
      - marque: de type chaine de caractères.
      - nombre de chevaux: de type entier.
      - numéro de plaque: de type chaine de caractères.
      - Type de carburant: de type chaine de caractères.
      - Kilométrage : de type entier.
   b. Ecrire un constructeur sans paramètres. **(1 pt)**
   c. Écrire un constructeur avec tous les paramètres. **(2 pts)**
   d. Ajouter la méthode polymorphe **CalculerCharge()** qui retourne le Kilométrage*11. **(2 pts)**
   e. Créer une exception nommée ***cheveauException*** qui se déclenche si le nombre de chevaux est inférieur à 6 ou supérieur à 14. **(2 pts)**
   f. Ecrire la méthode ***ToString()*** permettant d'afficher les informations sur une charge de déplacement. **(1 pt)**

2. Classe **ListeChargeDéplacement**:
   a. Créer la classe **ListeChargeDéplacement** caractérisée par : **(2 pts)**
      - Une collection d'objets **Dépense**.
   b. Ajouter un constructeur par défaut. **(1 pt)**

c. Créer la méthode ***Ajouter*** qui permet d'ajouter une **Dépense** en paramètre, l'application doit afficher un message de confirmation avant l'ajout de la **Dépense. (2 pts)**

d. Créer la méthode ***Afficher*** qui permet d'afficher la liste des **Dépenses. (2 pts)**

e. Créer la méthode ***Supprimer*** qui permet de supprimer les informations d'une **Dépense** en paramètre, l'application doit afficher un message de confirmation avant de supprimer une **Dépense. (2 pts)**

f. Créer la méthode **Rechercher** qui permet d'afficher les Dépenses dont le montant est supérieur à 1000. **(3 pts)**

3. Programme principal.

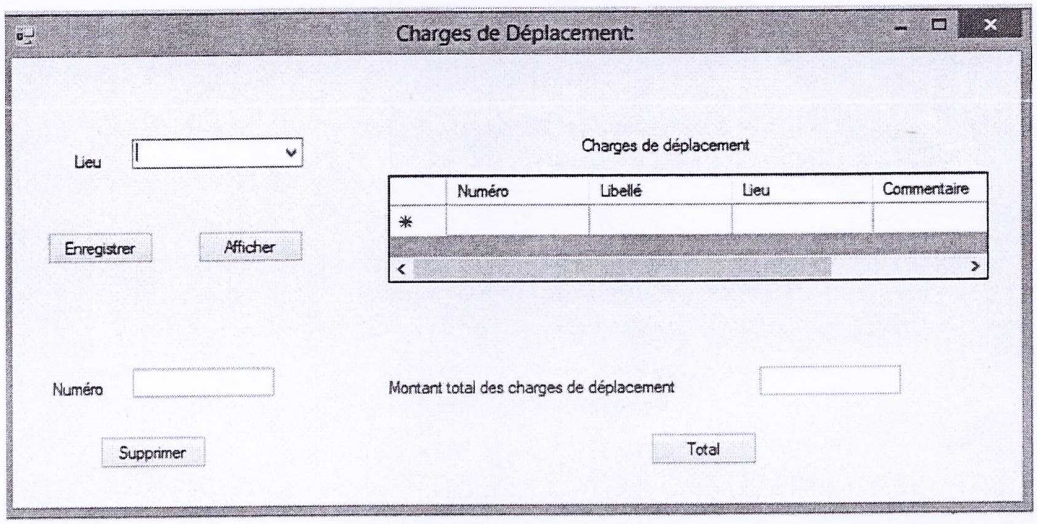
a. Afficher le menu permettant d'accéder aux différentes méthodes de la question précédentes. **(4 pts)**

b. Instancier 3 objets de la classe **Dépense, Salarié, ChargeDéplacementVoiture. (2 pts)**

c. Ajouter 3 objets **Dépense** à la collection **ListeChargeDéplacement. (2 pts)**

Dossier 3 : (25 Pts)

En utilisant les classes définies dans le Dossier 1, nous proposons l'interface graphique suivante permettant de faciliter la gestion des **Charges de Déplacement**:



1. Ecrire le code du bouton ***Enregistrer*** permettant d'enregistrer la liste des **charges de déplacement** dans un fichier texte. **(6 pts)**

2. Ecrire le code du bouton ***Afficher*** permettant d'afficher dans la grille les charges de déplacement d'un Lieu sélectionnée à partir de la zone de liste. **(6 pts)**

3. Ecrire le code nécessaire pour le bouton **Supprimer** qui permet de supprimer la charge de déplacement dont le numéro est saisie dans le textbox, la suppression doit être effectuée à la fois dans la liste et dans la grille. Un message de confirmation doit être affiché avant de procéder à la suppression. **(7 pts)**

4. Ecrire le code nécessaire pour le bouton **Total** permettant d'afficher le montant total des charges de déplacement enregistrées. **(6 pts)**

مكتب التكوين المهني وإنعاش الشغل
OFPPT
Office de la Formation Professionnelle et de la Promotion du Travail
Direction Recherche et Ingénierie de la Formation

Examen de passage à la 2eme année

Session Juin 2016

Filière : Techniques de Développement Informatique
Epreuve : Synthèse
Niveau: TS
Variante : V2
Durée : 5 heures
Barème : / 120

❖ **Partie I : Théorie** (40 pts)

➢ **Dossier 1: Notions de mathématiques appliquées à l'informatique (12 pts)**

1. Convertir en binaire les nombres suivants ***(06 pts)***

   $(321)_8$; $(C0F)_{16,}$; $(45)_{10}$

2. Effectuer en binaire l'opération suivante ***(02 pts)***

   11111010-11

3. A l'aide du tableau de Karnaugh, simplifier la fonction H définie par sa table de vérité suivante : **(*04* pts)**

E	F	G	H
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

➢ Dossier 2: Techniques de programmation structurée (8 pts)

Ecrire un algorithme qui demande à l'utilisateur de saisir deux tableaux d'entiers **T1** et **T2** de dimensions respective **A** et **B**, et retourne une matrice d'entiers de **B** lignes sur **A** colonnes où chaque case de coordonnées **i, j** est le résultat positif de la soustraction de **T1[i]** avec **T2[j]**, le résultat de la soustraction est multiplié par le minimum des deux tableaux. (Voir exemple ci-dessous.)

Exemple :

T1

4	7	3	8

T2

8	3	7	4	10

Le minimum des deux tableaux est 3
Le résultat est :

	4	**7**	**3**	**8**
8	abs (8-4) * 3 = 12	3	15	0
3	3	12	0	15
7	12	0	12	3
4	0	abs(4-7)*3=9	3	12
10	18	9	21	6

➢ Dossier 3: Conception et modélisation d'un système d'information (20 pts)

Un organisme souhaite informatiser la gestion des **ordres de missions** de ses employés, ce système permet aux utilisateurs de saisir leurs ordres de missions dans un logiciel de gestion afin d'accélérer le traitement des ordres de missions.

L'employé est identifié par un numéro de matricule, nom, prénom, service, département, et date de recrutement.

Chaque ordre de mission est identifié par un numéro unique pour toutes réclamations.

Les déplacements des employés, concernant les missions, peuvent être de types train, taxi, hôtel, ou voiture, ils concernent un projet ou un service.

Pour toutes missions, l'employé doit renseigner la date de début, la date de fin de mission, le libellé de la mission, la ville de destination, frais avec la prise en compte des plafonds, et la gestion des commentaires expliquant des Missions exceptionnelles (dépassement de plafond, impondérables...).

Les ordres de mission des employés en déplacement avec les trains doivent renseigner le type de train, classe et le montant.

Chaque ordre de mission doit être validé par le chef de service, par le chef de département et le directeur de la société. L'employé peut effectuer des relances pour accélérer le traitement de l'ordre de mission.

Travail demandé.

1. Etablir le dictionnaire de données. **(06 pts)**
2. Etablir le modèle conceptuel des données correspondant ***(08 pts)***
3. Etablir le modèle logique des données associé. ***(06 pts)***

❖ Partie II: Pratique (80 pts)

➢ Dossier 1: Langage de programmation structurée (25 pts)

L'organisme souhaite représenter sous forme d'un tableau des **ordres de missions** de ses employés.

Chaque enregistrement de type **ordreMission** est composé d'un identifiant, la durée de la mission, le libellé de la mission, la ville de destination, les frais, le commentaire et l'état de l'ordre de mission (Validé, Refusé).

1- Définir une structure **ordreMission** pouvant contenir ces informations. **(2 pts)**

2- Définir un tableau de structure ***HistoriqueOrdreMission*** de type **ordreMission** permettant de représenter l'ensemble des **ordres de missions** des employés. (utiliser un tableau de taille maximale 30) **(1 pt)**

3- Ecrire un sous-programme qui permet de saisir un certain nombre d'ordreMission dans le tableau structuré ***HistoriqueOrdreMission***. **(3 pts)**

4- Ecrire un sous-programme qui permet d'afficher tous les **ordres de missions** de ***HistoriqueOrdreMission***. **(3 pts)**

5- Ecrire un sous-programme qui permet d'afficher les informations des **ordres de missions** ayant le statut « Refusé ». **(3 pts)**

6- Ecrire un sous-programme qui permet de calculer le montant global des frais des ordres de missions. **(3 pts)**

7- Ecrire un sous-programme qui permet de Modifier le montant d'un **ordre de mission** correspondant à un numéro donné. **(3 pts)**

8- Demander à l'utilisateur de saisir les informations d'un **ordre de mission**. Ajouter ensuite cet **ordre de mission** au tableau ***HistoriqueOrdreMission***. L'ajout sera

fait dans le bon endroit selon le numéro identifiant (Pour avoir un ordre par numéro identifiant). **(4 pts)**

9- Créer un menu pour appeler les sous-programmes précédents. **(3 pts)**

Dossier 2: Langage de programmation Orientée Objet (30 pts)

On souhaite informatiser la gestion des ordres de missions.

Les classes suivantes sont déjà créer :

```
class Employé
    {
        int Matricule  ;
        public string Nom ;
        public string Prenom ;
        public string Adresse ;
        public string Genre ;
        public float Age ;
        private string service  ;
        private string departement  ;
        public string Ville ;
        public Employé() { }
        public Employé(int id, string nom, string prenom, string adresse, string
Genre, float Age, string service, string dep, string v)
        {

            this.Matricule = id;
                this.Nom = nom;
                this.Prenom = prenom;
                this.Adresse = adresse;
                this.Genre = Genre;
                this.Age = Age;
                this.service = service;
                this.departement = dep;
                this.Ville = v;
        }

        public override string ToString()
        {
            return "id:"+ this.Matricule  + " Nom\n" + this.Nom + "Prenom\n" +
this.Prenom+"Adresse:"+ this.Adresse + " Genre\n" + this.Genre + "Age\n" +this.Age+
"service:"+ this.service + " Departement\n" + this.departement + " Ville\n" +
this.Ville   ;
       }}
```

```
class Mission
    {

    int      Numero;
    string   Libellé;
    string   Lieu;
    string   Commentaire;
    float Montant;
       public Mission() { }
       public Mission(int num, string libelle, string lieu, string commentaire,
float montant)
        {
            this.Numero = num;
```

```
            this.Libellé = libelle;
            this.Lieu = lieu;
            this.Commentaire = commentaire ;
            this.Montant = montant ;
        }

public virtual int CalculerCharge ()
        {           return montant * taux ;}
        public override string ToString()
        {
            return "Numero:" + this.Numero + " Libellé\n" + this.Libellé + "Lieu\n"
+ this.Lieu + "Commentaire:" + this.Commentaire + " Montant\n" + this.Montant;
        }}
```

1. Classe **OrdreMissionTrain**:
   a. Créer une classe **OrdreMissionTrain** qui hérite de la classe **Mission** et caractérisée par : **(2 pts)**
      - Employé : de type Employé.
      - type de train: de type chaine de caractères.
      - classe: de type entier.
      - montant: de type float.
   b. Ecrire un constructeur sans paramètres. **(1 pt)**
   c. Écrire un constructeur avec tous les paramètres. **(2 pts)**
   d. Créer une exception nommée ***montantException*** qui se déclenche si le montant est inférieur à 10 ou supérieur à 500. **(2 pts)**
   e. Ajouter la méthode polymorphe **CalculerCharge()** qui retourne le montant*classe. **(2 pts)**
   f. Ecrire la méthode ***ToString()*** permettant d'afficher les informations sur un ordre de mission. **(1 pt)**
2. Classe **ListeMissions**:
   a. Créer la classe **ListeMissions** caractérisée par : **(2 pts)**
      - Une collection d'objets **Mission**.
   b. Ajouter un constructeur par défaut. **(1 pt)**
   c. Créer la méthode ***Ajouter*** qui permet d'ajouter une **Mission** en paramètre, l'application doit afficher un message de confirmation avant l'ajout de la **Mission**. **(2 pts)**
   d. Créer la méthode ***Afficher*** qui permet d'afficher la liste des **Missions**. **(2 pts)**

e. Créer la méthode ***Supprimer*** qui permet de supprimer les informations d'une **Mission** en paramètre, l'application doit afficher un message de confirmation avant de supprimer une **Mission**. **(2 pts)**

f. Créer la méthode **Rechercher** qui permet d'afficher les Missions dont le montant est supérieur à 1000. **(3 pts)**

3. Programme principal.

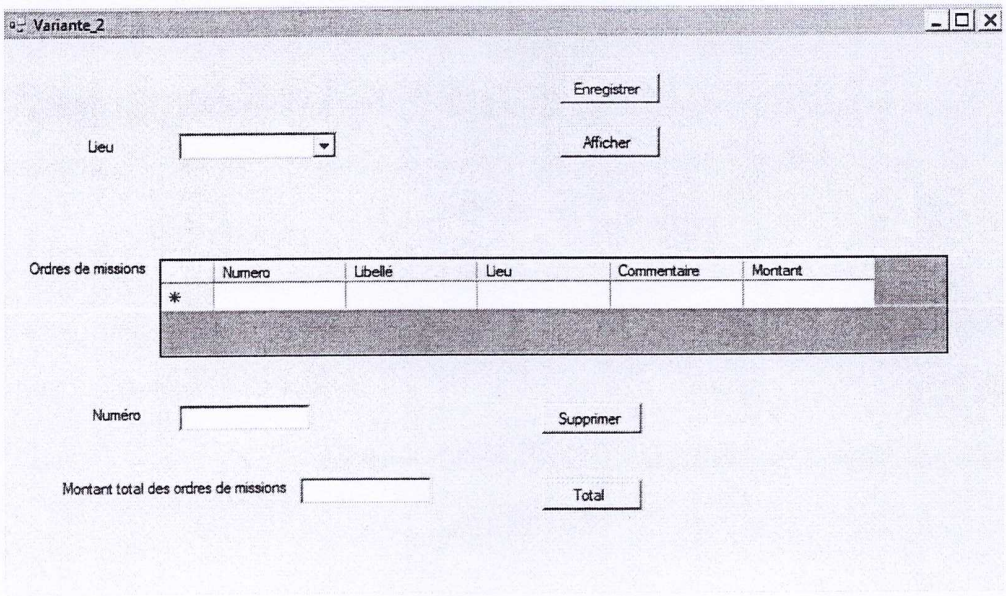
   a. Afficher le menu permettant d'accéder aux différentes méthodes de la question précédentes. **(4 pts)**

   b. Instancier 3 objets de la classe **Mission, Employé, OrdreMissionTrain**. **(2 pts)**

   c. Ajouter 3 objets **Mission** à la collection **ListeMissions**. **(2 pts)**

Dossier 3 : (25 Pts)

En utilisant les classes définies dans le Dossier 1, nous proposons l'interface graphique suivante permettant de faciliter la gestion des ordres de missions:



1. Ecrire le code du bouton ***Enregistrer*** permettant d'enregistrer la liste des ordres de missions dans un fichier texte. **(6 pts)**

2. Ecrire le code du bouton ***Afficher*** permettant d'afficher les ordres de missions d'un Lieu sélectionnée à partir de la zone de liste. **(6 pts)**

3. Ecrire le code nécessaire pour le bouton **Supprimer** qui permet de supprimer l'ordre de mission dont le numéro est saisi dans le textbox, la suppression doit être effectuée à la fois dans la liste et dans la grille. Un message de confirmation doit être affiché avant de procéder à la suppression. **(7 pts)**

4. Ecrire le code nécessaire pour le bouton **Total** permettant d'afficher le montant total des ordres de missions enregistrés. **(6 pts)**