| | | |
|---|---|---|
| ENIG | **République Tunisienne**<br>**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**<br>**Université de Gabès**<br>**École Nationale d'Ingénieurs de Gabès** | **Réf : DE-EX-01**<br>**Indice : 3** |
| | **EPREUVE D'EVALUATION** | **Date : 02/12/2019**<br>**Page : 1/6** |

| | |
|---|---|
| **Année Universitaire** : 2021/2022 | **Date de l'Examen** : 17/05/2022 |
| **Nature** : ☐ DC ☑ Examen ☐ DR | **Durée** : ☐ 1h ☐ 1h30min ☑ 2h |
| **Diplôme** : ☐ Mastère ☑ Ingénieur | **Nombre de pages : 06** |
| **Section :** ☐ GCP ☐ GCV ☐ GEA ☑ GCR ☐ GM | **Enseignant :** I. Azaiez |
| **Niveau d'étude** : ☑ 1ère ☐ 2ème ☐ 3ème année | **Documents Autorisés** :☐ Oui ☑ Non |
| **Matière :** Programmation II | **Remarque :** |

**NB.** ***La qualité de rédaction***, la ***clarté*** et la précision des réponses et l'utilisation des ***commentaires*** seront des éléments d'évaluation très importants. **Le barème est donné à titre indicatif (/ 40 pts)**

## QCM (6 pts)

**Q1.** Quelle instruction initialise un `StringBuffer` à une valeur de 128 ?

A. `StringBuffer sb = new StringBuffer ("128");`
B. `StringBuffer sb = StringBuffer.valueOf(128);`
C. `StringBuffer sb = StringBuffer.getInstance (128);`
D. `StringBuffer sb = new StringBuffer (128);`

**Q2.** Quel est le résultat du programme suivant :

```
public static void main(String[] args) {
          int t[] = {1,2,3,4,5};
          System.arraycopy(t, 2, t, 1, 2);
          System.out.println(t[1]);
          System.out.println(t[4]);
    }
```

A. 14
B. 15
C. 24
D. 25
E. 34
F. 35

**Q3.** Qu'est-ce qui fera compiler et exécuter le programme suivant ?

```
1. public class Simple {
2. public int prix;
3. public static void main(String[] args) {
4.        Simple prix = new Simple();
5.        prix=4;}
6. }
```

A. Remplacez la ligne 2 par la suivante : `public int prix ;`
B. Remplacez la ligne 4 par la suivante : `int prix = new simple ();`
C. Remplacez la ligne 4 par la suivante : `float prix = new simple ();`

D. Remplacez la ligne 5 par la suivante : `prix = 4f;`
E. Remplacez la ligne 5 par la suivante : `prix.prix = 4;`
F. Le code se compile et s'exécute correctement ; aucune modification n'est nécessaire.

**Q4.** Qu'est-ce qui est vrai à propos de la classe Forme ?

```
public abstract class Forme {
     public int air;

     public Forme(int air){
          this.air=air;
     }
     public void air(){ }
     public void perim(){ }
}
```

A. Elle compile sans erreur.
B. Elle ne compile pas car une classe abstraite ne peut pas avoir de méthodes privées.
C. Elle ne compile pas car une classe abstraite ne peut pas avoir de variables d'instance.
D. Elle ne compile pas car une classe abstraite doit avoir au moins une méthode abstraite.
E. Elle ne compile pas car une classe abstraite doit avoir un constructeur sans arguments.

**Q5.** Quel est le résultat du programme suivant :

```
1. public static void main(String[] args) {
2.       List<Integer> liste = new ArrayList<>();
3.       liste.add(1001);
4.       liste.add(1002);
5.       System.out.println(liste.get(liste.size()));
6. } }
```

A. La compilation échoue en raison d'une erreur à la ligne 2.
B. Une exception est déclenchée lors de l'exécution en raison d'une erreur à la ligne 3
C. Une exception est levée au moment de l'exécution en raison d'une erreur à la ligne 5
D. 1002

**Q6.** Quelles sont les trois lignes illégales ?

```
 1. public class StaticMethod {
 2. static void f(){
 3.       g();
 4.       StaticMethod.g();
 5.       h();
 6.       StaticMethod.k();
 7. }
 8. static void g(){ }
 9. void h(){
10.       g();
11.       StaticMethod.g();
12.       k();
13.       StaticMethod.k();
14.       }
15.  void k(){ }
```

A. ligne 3
B. ligne 4
C. ligne 5
D. ligne 6
E. ligne 10
F. ligne 11
G. ligne 12
H. ligne 13

# Problème

On souhaite concevoir et développer une application en java qui permet de gérer les **contacts** ainsi que les **appels** téléphoniques **émis** et **reçu**. Le système enregistre un ensemble de contacts. Chaque **contact** est défini par son **id**, son **nom**, son **numéro téléphonique** et son **solde**. Le système enregistre également un ensemble d'appels. Il existe deux types **d'appels émis** et **reçus**. Chaque **appel concerne un contact**. Un **appel** est défini par son **id**, sa **date** et sa **durée**. Le coût de chaque appel dépend de son type. Pour les **appels émis**, le coût est calculé en multipliant la durée de l'appel par **deux** alors que pour un **appel reçu**, le coût étant nul.

Cette application doit permettre d'effectuer les opérations suivantes :

1. Enregistrer un nouveau contact.
2. Enregistrer un nouvel appel sachant le numéro de téléphone.
3. Consulter un contact sachant son numéro de téléphone.
4. Consulter les contacts.
5. Consulter le coût total des appels sachant son numéro de téléphone.

**NB**. Le format du jour est une chaine de caractère sous la forme de : « 01/01/2022 »

Pour ce faire on doit suivre les étapes suivantes :

## Étape 1 : Développement de la couche modèle (10 pts)

Définir pour chacune des classes identifiées, leurs attributs (déclarés privés) et le(s) constructeur(s) ***seulement***.

## Étape 2 : Développement de la couche dao (13 pts)

Soit l'interface générique **IContact**, cette interface contient la déclaration des méthodes :

- `public void saveContact(Contact c);`
- `public void saveAppelEmi(Appel ap, Contact c);`
- `public Contact getContact(int num);`
- `public List<Contact> getAllContact();`
- `public double coutAppel(int numC);`

Créer la classe **ContactImp** qui implémente l'interface **IContact.**

## Étape 3 : Développement de la couche Vue (11 pts)

1. L'application est démarrée à partir de l'interface graphique « **MonAppli.java** » à laquelle, l'utilisateur est amené à choisir soit de cliquer sur le bouton « **Contacts** » soit sur « **Solde** » ou sur le bouton « **Quitter** ». Une fois il a cliqué sur le bouton « **Contacts** », le système affiche la fenêtre « **AjoutContact.java** » permettant d'ajouter un nouveau contact dans la BDD.

2. Si l'utilisateur clique sur le bouton « **Solde** », il sera dirigé vers la fenêtre « **SuiviSolde.java** » contant la liste de tous les contacts. Ainsi, il peut consulter le solde (restant) d'un contact choisi parmi la liste.
3. Le bouton « **Quitter** » arrête l'exécution de l'application. Tandis que le bouton « **Retour** » permet de retourner à la fenêtre principale de l'application « **MonAppli.java** ».

## Travail demandé : (voir ANNEXE)

1. Implémenter les différentes classes des trois couches (modèle, dao et vue)
2. Apporter les **ajouts nécessaires** aux interfaces graphiques « **Ajout*Contact.java*** » ***et*** « ***SuiviSolde.java*** »

# ANNEXE

## MLD

TABLE `**appelemi**` (`num` int (11) NOT NULL, `dateappel` varchar(45) NOT NULL, `duree` double NOT NULL, `numcont` int(11) NOT NULL )

TABLE `**appelreçu**` (`num` int (11) NOT NULL, `dateappel` varchar(45) NOT NULL,`duree` double NOT NULL, `numcont` int(11) NOT NULL)

TABLE `**contact**` (`num` int (11) NOT NULL, `nom` varchar(45) NOT NULL, `numtel` varchar(45) NOT NULL, `solde` double NOT NULL)

## Connections à la BDD

```
Class.forName("com.mysql.jdbc.Driver");
String url ="jdbc:mysql://localhost:3306/gestionAppel";
Connection con = DriverManager.getConnection(url,"root","root");
```

## Interface graphique de « MonAppli.java »

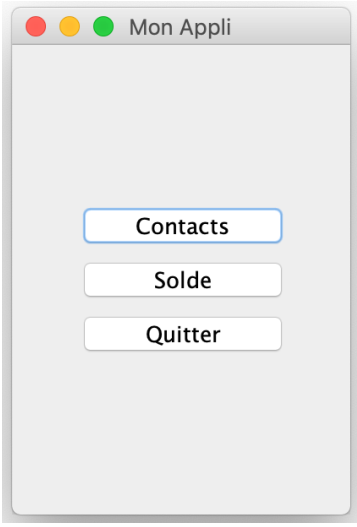


Figure 1. MonAppli.java

## Interface graphique de « AjoutContact.java »

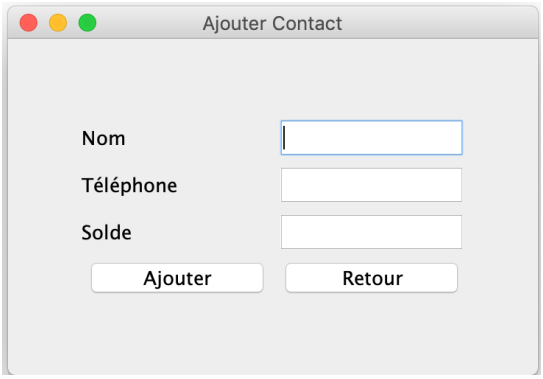


Figure 2. AjoutContact.java

## Code Java de l'interface principale AjoutContact.java

```
public class AjoutContact implements … {

      JFrame f = new JFrame("Ajouter Contact");
      JLabel lnom = new JLabel("Nom");
      JLabel lnum = new JLabel("Téléphone");
      JLabel ls = new JLabel("Solde");
      JTextField txnom = new JTextField();
      JTextField txnum = new JTextField();
      JTextField txs = new JTextField();
      JButton ba = new JButton("Ajouter");
      JButton br = new JButton("Retour");

      public AjoutContact(){
   //Vous n'êtes pas censés développés l'interface graphique
   //…
   }public static void main(String[] args) {
       //… }
      @Override
   public void actionPerformed(ActionEvent e) {
      //… }}
```

## Interface graphique de « SuiviSolde.java »

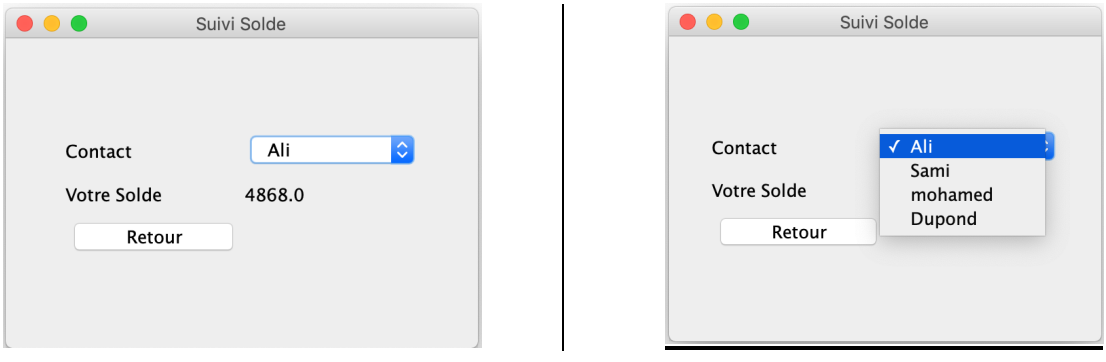


Figure 3. SuiviSolde.java

**Code Java de l'interface SuiviSolde.java**

```
public class SuiviSolde implements … {

      JFrame f = new JFrame("Suivi Solde");
      JLabel lc = new JLabel("Contact");
      JComboBox<String> cmbc = new JComboBox<String>();
      JLabel ls = new JLabel("Votre Solde");
      JLabel lvs = new JLabel(" ");

      JButton br = new JButton("Retour");
      List<Contact> liste = new ArrayList<Contact>();

      public SuiviSolde(){
      //Vous n'êtes pas censés développés l'interface graphique
      //…
   }
```

| Composant | Méthode | Description |
|---|---|---|
| JTextField | String getText() | Récupérer une valeur |
| | setText (String s) | Affecter une valeur |
| JLabel | String getText() | Récupérer une valeur |
| | setText (String s) | Affecter une valeur |
| JComboBox | addItem(Objet item) | Cette méthode va ajouter un nouvel objet à la liste. |
| | Object getSelectedItem() | Cette méthode retourne l'objet qui est actuellement sélectionné. |
| | int getSelectedIndex() | Cette méthode retourne l'indice de l'élément qui est actuellement sélectionné. (Le premier élément d'indice 0) |
| **Autres méthodes/Classes** | | |
| ResultSet, executeQuery(String), executeUpdate(String) | | |