|  |
| --- |
| BTS SERVICES INFORMATIQUES AUX ORGANISATIONS |

E5 : Production et fourniture de services informatiques

**SESSION 2018**

|  |  |
| --- | --- |
| **Durée : 4 heures** | **Coefficient : 5** |

**Cas WebCaisse**

Revu – Version PHP – Document et Internet autorisés

***Travail personnel***

**Dossier documentaire :**

**Barème :**

|  |  |  |
| --- | --- | --- |
| Mission 1 | Souscription en ligne | 30 points |
| Mission 2 | Fidélisation des consommateurs | 25 points |
| Mission 3 | Statistiques de ventes | 30 points |
|  | Total | 85 points |

**L'organisation cliente : Nasdy**

La société Nasdy, implantée en Martinique et en Guadeloupe, est une entreprise de services du numérique créée en 2001 par deux ingénieurs. Elle accompagne les organisations dans leurs projets de dématérialisation et participe au développement local du numérique (animation d’ateliers sur le *big data*, les *Google Apps*, etc.), tout en menant une expansion à l’international.

Elle propose également la création de sites, la gestion de la communication numérique ou encore la réalisation d’études de faisabilité technique de projets *web* ou mobiles.

Nasdy assure une veille technologique sur les nouveaux besoins, ce qui amène la société à développer des applications en interne en vue d’une commercialisation.

**Le logiciel *WebCaisse***

Pour rendre impossible la fraude, l'article 88 de la loi de finances pour 2016 instaure, à partir du 1er janvier 2018, l'obligation pour les commerçants et autres professionnels assujettis à la TVA d’enregistrer les paiements de leurs clients au moyen d'un logiciel de comptabilité ou d'un système de caisse sécurisé et certifié NF525.

Pour répondre à ce besoin réglementaire, Nasdy a décidé de développer un logiciel de caisse enregistreuse en ligne, nommé *WebCaisse,* qu'elle propose à ses clients.

Nasdy mise sur la proximité avec les commerçants des Antilles (épiciers, coiffeurs, boulangers, etc.) qui sont nombreux à tenir leurs comptes sur support papier. Dans un second temps, la *WebCaisse* sera proposée au reste de la clientèle potentielle (Dom,Tom, France métropolitaine, étranger).

La *WebCaisse* proposée par Nasdy est un logiciel en mode *SaaS**(Software as a Service*) destiné à l'encaissement et à la gestion de toute activité d'un commerce indépendant ou d'un réseau de magasins. Elle met à la disposition des points de vente des solutions d’encaissement sur divers supports : caisse, tablette, *smartphone*, etc.

Le logiciel *WebCaisse* et sa base de données sont hébergés pour chaque client chez un hébergeur spécialisé.Il propose différents modules : gestion des ventes, gestion des stocks, statistiques, campagne publicitaire, aide à la décision, gestion de la relation client et e-commerce. D'autres modules pourront, par la suite, être développés en fonction des besoins. Différentes formules de souscription seront proposées, chacune contenant différents modules paramétrables selon les besoins du client.

Une première version du logiciel *WebCaisse* a été développée ; elle contient certains modules proposés par le logiciel.

**L'application *AchatWebCaisse***

Pour conserver les caractéristiques des formules de *WebCaisse* souscrites, une application *AchatWebCaisse* a été développée.

Utilisée uniquement par le service commercial de Nasdy, elle permet actuellement d'enregistrer et de consulter la formule de *WebCaisse* souscrite par chaque client et utilisée par tous ses points de vente.

Le développement du logiciel *WebCaisse* et de l'application *AchatWebCaisse*ont été confiés à une équipe interne à Nasdy composée de trois personnes et dirigée par un chef de projet, M. Renaud Boileau.

Au sein de la société Nasdy, vous prenez part au projet de développement de l’application *AchatWebCaisse* et du logiciel *Webcaisse*.

**Précisions sur les termes "clients" et "consommateurs" utilisés dans ce dossier** :

* les **clients** sont les commerçants qui achètent le logiciel *WebCaisse* ;
* les **consommateurs** sont les personnes qui effectuent des achats chez les commerçants**.**

**Mission 1 - Souscription en ligne**

Nasdy souhaite que chaque client puisse gérer la souscription au logiciel *WebCaisse* pour chacun de ses points de vente, sans être obligé de contacter le service commercial de Nasdy. Pour cela, l'application *AchatWebCaisse* doit être modifiée pour :

* mettre à disposition des clients des formulaires accessibles sur *internet* permettant de souscrire ou modifier une formule *WebCaisse* propre à chaque point de vente du client ;
* gérer les caractéristiques des formules, des modules et des programmes de fidélisation des consommateurs.

Toutes les informations collectées via l'application *AchatWebCaisse* permettront ensuite de réaliser automatiquement le déploiement du logiciel *WebCaisse* à partir des caractéristiques choisies par le client.

**L'application *AchatWebCaisse*** utilise une base de données dont le schéma complet vous est fourni dans le dossier documentaire.

Votre chef de projet vous demande de faire évoluer la structure de la base de données utilisée par l'application *AchatWebCaisse* pour qu'elle intègre les données nécessaires à la souscription en ligne présentée ci-après.

**Partie 1 – Évolution de la base de données pour la souscription en ligne**

***Documents à utiliser : 1 et 2***

La souscription en ligne sera réalisée en trois étapes : choix de la formule, choix du programme de fidélisation et choix des langues pour les modules.

**Choix de la formule**

Pour commander le logiciel *WebCaisse* en ligne, le client devra fournir différentes informations qui seront mémorisées : sa raison sociale, son adresse, son numéro de téléphone et son domaine d’activité (restauration, alimentation, coiffure, etc.).

L'identifiant et le mot de passe qui lui permettront de se connecter seront transmis par messagerie électronique.

Une fois connecté, le client devra renseigner le nom et l'adresse précise de ses points de vente puis, pour chacun d'eux, il sera invité à sélectionner une formule de *WebCaisse* dans un formulaire lui affichant les différents modules compris dans chaque formule. Une maquette du formulaire est fournie dans le dossier documentaire.

Le taux de commission, qui concerne uniquement, pour le moment, le module e-commerce, sera affiché : il permettra au client de connaitre le taux de commission qui sera appliqué au montant des ventes en ligne réalisées.

**Choix du programme de fidélisation des consommateurs**

Le logiciel *WebCaisse* propose trois programmes de fidélisation :

* le programme de type 1 "**Tampon"** ;
* le programme de type 2 "**Montant Achat**";
* le programme de type 3 "**Points"**.

Le programme de fidélisation retenu pour chaque point de vente sera mémorisé dans la base de données.

**Choix des langues pour les modules**

Cette dernière étape permet d'activer les modules dans une ou plusieurs langues.

Chaque module du logiciel *WebCaisse* possède un fichier de traduction pour chacune des langues dans lesquelles il est proposé.

Pour chaque point de vente, le client devra indiquer, pour chaque module souscrit, la ou les langues que les utilisateurs de ce module pourront utiliser.

L'activation des modules sera mémorisée dans la base de données.

**Question 1.1**

Modifier la structure de la base de données utilisée par l'application ***AchatWebCaisse*** afin de permettre la souscription en ligne du logiciel *WebCaisse*.

***IMPORTANT*** *: la candidate ou le candidat présentera les évolutions de la structure de la base de données en adoptant soit :*

*\* Diagramme de classe (papier ou Visual Paradigm)*

*\* Modèle relationnel, même format que le sujet à savoir*

**NomDeLaTable (ensemble des champs)**

*clé primaire :* nomDeLaClefPrimaire

*clé étrangère :* nomDeLaClefEtrangere *en référence à* ***Table XXX*** *(*nomDeLaClefPrimaireXXX)

**Partie 2 - Gestion du changement de formule**

Un client doit pouvoir, à tout moment, changer la formule d'un de ses points de vente en fonction de ses besoins.

Il est nécessaire de garder **l'historique des formules souscrites** pour chaque point de vente, dont un changement de formule ne peut être effectué qu'une seule fois par jour. Bien entendu, rien n'interdit de souscrire, pour un point de vente, une formule déjà souscrite par le passé.

Le changement de formule aura un impact sur le montant à régler mensuellement par le client : ce montant dépend du prix et de la durée d'activation des différentes formules choisies durant le mois de facturation. Le calcul du montant à régler ne fait pas partie de cette mission.

Pour gérer l'historique des formules souscrites, votre collègue Gilles Sabatier vous propose d'ajouter une table dans la base de données ayant la description suivante :

**FormuleSouscrite(idPointDeVente, idFormule)**

*clé primaire : idPointDeVente, idFormule*

*clés étrangères : idPointDeVente en référence à id de PointDeVente*

*idFormule en référence à id de Formule*

**Question 1.2**

Expliquer en quoi la structure de la table ne permettra pas de gérer l'historique des formules souscrites, indispensable à la détermination du montant mensuel à régler par le client.

**Question 1.3**

Proposer une correction de la structure de la table qui réponde au besoin exprimé.

**Mission 2 : Fidélisation des consommateurs**

*IMPORTANT : pour la partie programmation, la candidate ou le candidat programmera en* ***PHP***

**Partie 1 – Amélioration du module de gestion de la relation client (GRC)**

***Documents à utiliser : 3, 4, 5 et 6***

Le logiciel *WebCaisse* propose, pour certaines formules, l’accès à un module de gestion de la relation client (GRC) disposant de fonctions de gestion des contacts, de rapports d’activité, etc.

Le tableau de bord du module GRC doit être complété pour permettre une analyse rapide des ventes réalisées par les consommateurs.

Les éléments d’information suivants seront ajoutés dans ce tableau de bord :

1. la liste des consommateurs (nom, prénom, adresse de courriel) pour lesquels au moins une vente a été réalisée en 2017 ;
2. le nombre de consommateurs ayant souscrit au programme de fidélité et appartenant à la tranche d’âge 18-30 ans ;
3. la liste des consommateurs (nom, prénom, adresse de courriel) avec le montant total des ventes réalisées pour chacun.

**Question 2.1**

Écrire les requêtes permettant d’extraire les informations nécessaires de la base de données fournie dans le dossier documentaire.

Le tableau de bord du module GRC est personnalisable : d'autres indicateurs, prédéfinis dans le logiciel, peuvent être ajoutés sur ce tableau de bord et certains d'entre eux sont paramétrables par l'utilisateur.

L'un d'eux, l'indicateur "Consommateurs réguliers" permet d'obtenir la liste des consommateurs non encore fidélisés pour lesquels au moins une vente a été réalisée lors des **trente derniers jours**.

La méthode *listeConsoAFideliser* de la classe GRC établit la liste de ces consommateurs.

La fonctionnalité de recherche des consommateurs réguliers existante doit évoluer vers davantage de souplesse en permettant au client de définir une période de recherche, en utilisant un formulaire de saisie des critères de recherche. Une maquette de ce formulaire est fournie dans le dossier documentaire. « Une vente lors des trente derniers jours**»** sera le critère proposé par défaut.

La signature de la méthode nommée *listeConsoAFideliser()* a été modifiée pour intégrer les trois paramètres suivants :

*public function listeConsoAFideliser($****seuilVentes, $dateDeb,$dateFin****)*

Un développeur, Sylvain Cho, a proposé un début de solution, avec la requête *SQL* suivante qui sera utilisée dans cette méthode :

String requete = "select C.nom, C.prenom, C.tel, C.mail, count(\*) as nbVentes "

. "from WEBCAISSE.Conso C "

. "INNER join WEBCAISSE.Vente V on V.idConso = C.id "

+ "group by C.nom, C.prenom, C.tel, C.mail "

+ "having count(\*) >" . $seuilVentes ;

**Question 2.2**

Modifier la requête *SQL* de la méthode *listeConsoAFideliser($****seuilVentes, $dateDeb,$dateFin****)* fournie par Sylvain Cho,afin de lister les consommateurs **qui n'ont pas adhéré au programme de fidélisation** et pour lesquels on a enregistré un nombre de ventes supérieur au seuil donné, durant la période donnée (le seuil et la période sont fournis en paramètre).

**Partie 2 – Réalisation de tests unitaires pour la méthode *AddFidelite()***

***Documents à utiliser : 5 et 8***

Le module de gestion des ventes de la *WebCaisse* gère le programme de fidélisation pour chaque point de vente :

* Pour le programme de type 1 (tampon), le module de gestion des ventes permet de compter le nombre de visites d’un consommateur. À chaque achat, le consommateur gagne un tampon. Il peut bénéficier d’une réduction à partir d'un nombre de tampons paramétré dans *WebCaisse* pour le point de vente.

* Pour le programme de type 2 (montant achat), le module de gestion des ventes totalise le total dépensé par chaque consommateur. Lorsque le montant total des achats atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d’une réduction.
* Pour le programme de type 3 (points), le module de gestion des ventes attribue un nombre de points fidélité lors de chaque achat du consommateur en fonction de la tranche d’achat :
  + 10 points (montant d'achat compris entre 100 € et 200 € inclus) ;
  + 20 points (montant d'achat supérieur à 200 € jusqu’à 500 €) ;
  + 50 points (montant d'achat supérieur à 500 €).

Lorsque le nombre de points atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d’une réduction.

La réduction associée au programme de fidélisation est également paramétrée dans *WebCaisse* pour chaque point de vente.

La méthode *addFidelite()* de la classe ConsoFidele, fournie dans le dossier documentaire, est appelée à l’issue de chaque achat d’un consommateur inscrit au programme de fidélisation. Elle permet de mettre à jour l'attribut *pointsFidelite* de la classe ConsoFidele en fonction du programme de fidélisation choisi pour le point de vente.

La classe de test TestPointsFidelite**,** fournie dans le dossier documentaire, a été créée afin de valider la méthode *addFidelite.*

Certains tests proposés dans cette classe ne sont pas terminés. Le test *testInitConso()* ne traite pas le cas de la valeur initiale des points de fidélité, qui devrait être fixée à zéro lors de la création d'un consommateur fidèle.

**Question 2.3**

Compléter la méthode *testInitConso*. Elle doit afficher le message « Bien initialisé à 0 » si la valeur initiale des points de fidélité est bien égale à 0

Votre chef de projet vous demande de compléter le test qui traite le cas du programme de fidélisation par points (type 3).

**Question 2.4**

Compléter la méthode *testAddMontant*permettant de valider les points de fidélité obtenus dans le cas d'un programme de fidélisation par points.

**Mission 3 : Statistiques de ventes**

***Documents à utiliser : 9 et 10***

*Codage : PHP*

Le module de statistiques du logiciel *WebCaisse* permet d'obtenir des statistiques sur les ventes par l'intermédiaire d'une classe Statistique qui contient plusieurs méthodes.

Une méthode de cette classe nommée *statVente* n'a pas été documentée lors de son écriture et votre chef de projet souhaite que vous fassiez le nécessaire.

**Question 3.1**

Rédiger le commentaire de la méthode *statVente* de la classe Statistique expliquant ce qu'elle retourne.

Le responsable de Nasdy souhaite ajouter des statistiques permettant de comparer les ventes en magasin aux ventes en ligne, réalisées sur le site e-commerce.

Dans un premier temps, on vous demande d’implémenter et compléter certaines méthodes.

**Question 3.2**

Écrire la méthode *getNbVentes* de la classe Consoqui retourne le nombre de ventes enregistrées dans la collection des ventes du consommateur.

**Question 3.3**

Écrire les constructeurs des classes VenteEcommerce et VenteMagasin qui permet d’initialiser tous les attributs d'une instance de la classe.

Une des statistiques devra permettre de visualiser rapidement quel canal de vente est le plus utilisé par les clients inscrits au programme de fidélisation, appelés clients fidèles, en comparant le montant total des ventes réalisées en ligne au montant total des ventes réalisées en magasin. Pour cela, l’application utilisera la méthode *compareLieuVente* de la classe Statistiquequi retournera le montant total des ventes réalisées en magasin divisé par le montant total des ventes réalisées en ligne.

**Question 3.4**

Compléter le code de la méthode *compareLieuVente.*

**Question 3.5**

Expliquer en quoi la dernière instruction "**return $totalMag/$ totalEcom**" de la méthode *compareLieuVente* peut poser problème.

On souhaite ajouter à la classe Conso une méthode qui retourne, pour le consommateur, la liste des ventes d’un montant supérieur à un montant passé en paramètre.

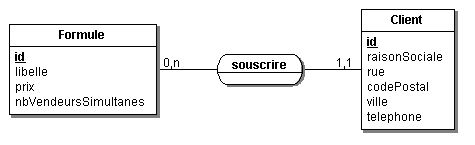
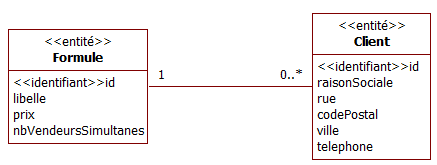
**Question 3.6**

Écrire la méthode répondant à ce besoin.

**Dossier documentaire**

# Document 1 : schéma complet de la base de données utilisée par l'application actuelle *AchatWebCaisse*

**Schéma conceptuel des données Diagramme de classes**



**Schéma relationnel :**

**Formule (id, libelle, prix, nbVendeursSimultanes)**

*clé primaire : id*

**Client (id, raisonSociale, rue, codePostal, ville, telephone, idFormule)**

*clé primaire : id*

*clé étrangère : idFormule en référence à id de Formule*

# Document 2 : maquette du formulaire "Choix de formule pour un point de vente"

**Nom du client : Société A. Boumbe**

**Choix de formule pour le point de vente "Chez Faly" situé à Le Marin :**

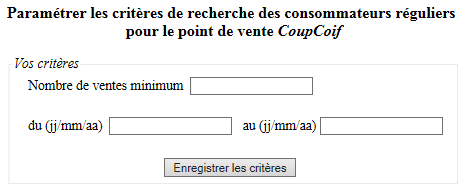
|  |  |  |  |
| --- | --- | --- | --- |
| **FORMULE ->** | **Silver** | **Gold** | **Platine** |
| Nombre maximum de vendeurs simultanés | 5 | 15 | 500 |
| **Modules proposés dans chaque formule** |  |  |  |
| Module gestion des ventes |  |  |  |
| Module gestion des stocks |  |  |  |
| Module statistique |  |  |  |
| Module campagne publicitaire |  |  |  |
| Module aide à la décision |  |  |  |
| Module GRC (gestion de la relation client) |  |  |  |
| Module e-commerce(\*\*) |  |  |  |
| Prix de la formule | 50€/mois | 90€ /mois | 120 €/mois |
| Cochez la case de la formule retenue pour le point de vente | ☐ | ☐ | ☐ |

**(\*\*) : + 2,8% du montant des ventes en ligne au titre du module e-commerce**

Vos points de vente et leurs formules :

|  |  |  |
| --- | --- | --- |
| **Point de vente** | **Adresse du point de vente** | **Formule** |
| Salon Hair Styl | 31, rue de la sérénité – 97224 Ducos | Gold |
| CoupCoif | 3, rue du clou - 97200 Fort de France | Silver |
| Chez Faly | 6, rue du flambeau – 97217 Le Marin | En attente de souscription |

# Document 3 : maquette du formulaire "Recherche des consommateurs réguliers"



# Document 4 : extrait du schéma relationnel pour la gestion des consommateurs fidèles

**Conso**(id, nom, prenom, mail, tel,… )

*clé primaire : id*

**ConsoFidele**(id, dateNaiss, nbPoints, dateInscription)

*clé primaire : id*

*clé étrangère : id en référence à id de Conso*

**Vente**(id, dateVente, montantVente, idConso,…)

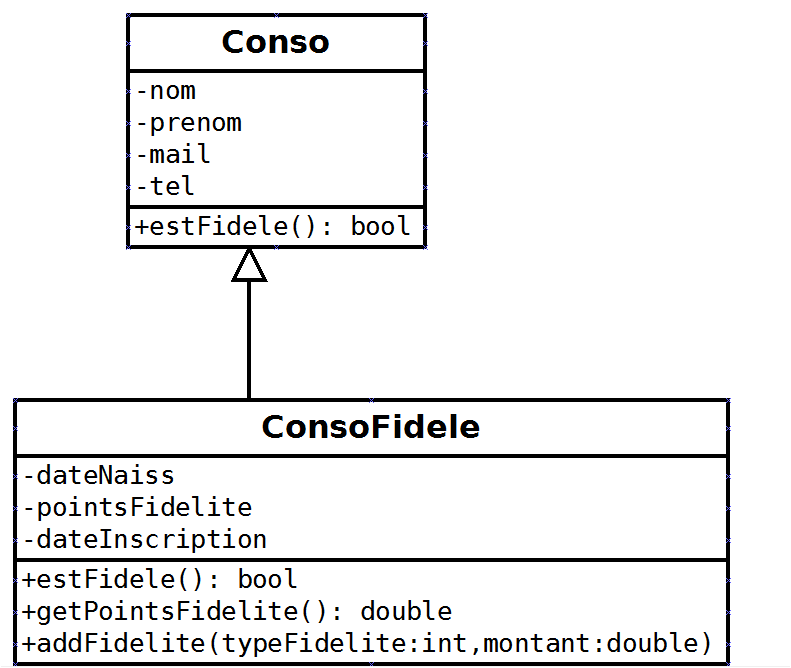
*clé primaire : id*

*clé étrangère : idConso en référence à id de Conso*

**Remarques :**

* La table ConsoFidele ne contient que les consommateurs ayant adhéré au programme de fidélité.
* La fonction *DATEDIFF ( datepart , startdate , enddate )* retourne la différence en valeur absolue et en nombre d'années entre *startdate* et *enddate*.
* La fonction GETDATE( ) retourne la date du jour en DATETIME
* La fonction CONVERT permet des conversions, par exemple de DATETIME en DATE
* La fonction *YEAR(uneDate)* retourne l'année de la date passée en paramètre.

# Document 5 : extrait du diagramme de classes utilisé par le module GRC



class Conso {

private $nom ;

private $prenom ;

private $mail ;

private $tel ;

public function \_\_construct($unNom, $unPrenom, $unmail, $untel) {

…

}

public function estFidele() {

return false;

}

}

class ConsoFidele extends Conso {

private $dateNaissance ;

private $pointsFidelite=0 ;

private $dateInscription ;

public function \_\_construct ($unNom, $unPrenom, $unmail, $untel, $dateNaiss, $dateInsc){ … }

public function estFidele() {

return true;

}

public function addFidelite($typeFidelite, $montant) {

    // typeFidelite contient le type de programme de fidélisation choisi (1, 2 ou 3)

    // montant contient la valeur du montant d’achat réalisé

    switch ($typeFidelite) {

case 1 :

$this->pointsFidelite ++ ;

            break;

case 2 :

$this->pointsFidelite += $montant;

            break;

case 3 :

if ($montant >=100 && $montant <=200) {

$this->pointsFidelite += 10;

}

            elseif ($montant >200 && $montant <=500){

$this->pointsFidelite += 20;

}

            elseif ($montant > 500) {

$this->pointsFidelite += 50;

}

             break;

    }

}

public function getPointsFidelite() {

return $this->pointsFidelite ;

}

}

# Document 6 : extrait de la classe GRC contenant la méthode *listeConsoAFideliser*

class GRC extends PDORepository{

private $lesConsos = array() ;

public function listeConsoAFideliser(){

//Création de la requête

$requeteSql ="select distinct C.nom, C.prenom, C.tel, C.mail"

            . " from WEBCAISSE.Conso C "

            . "INNER join WEBCAISSE.Vente V on V.idConso = C.id"

. " where C.id not in (select id from WEBCAISSE.ConsoFidele) "

. " AND DATEDIFF(DAY, V.dateVente ,CONVERT(date, GETDATE())) < 30";

//Exécution de la requête

$statement = $this->queryList($requeteSql,PDO::FETCH\_ASSOC,NULL) ;

//Pour chaque enregistrement résultat de la requête

while ($consoAFideliser = $statement->fetch()){

//Création d’un objet de la classe Conso avec les données

$leConso = new Conso($consoAFideliser['nom'],$consoAFideliser['prenom'],$consoAFideliser['tel'],$consoAFideliser['mail']) ;

//ajout du Consommateur à la liste des Consommateurs

$lesConsos[] = $leConso ;}

    // fermeture de la connexion à la BD

$statement->closeCursor();

return $lesConsos ;

}

}

# Document 8 : classe pour valider la méthode *addFidelite*

class TestPointsFidelite {

public function testAddFideliteTampon() {

$consoTest = new ConsoFidele("Lifo", "Paul","lifo.paul@gmail.com", "0600000000","1961-01-03","2017-01-05");

$consoTest->addFidelite(1, 50);

echo "Le nombre de points de fidélité doit être égal à 1 ici: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 1 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

$consoTest->addFidelite(1, 20);

echo "Le nombre de points de fidélité doit être égal à 2 ici: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 1 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

}

public function testAddFideliteMontant() {

$consoTest = new ConsoFidele("Lifo", "Paul","lifo.paul@gmail.com", "0600000000","1961-01-03","2017-01-05");

$consoTest->addFidelite(2, 150);

echo "Le nombre de points de fidélité doit être égal au montant cumulé d'achat, c'est à dire 150 ici: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 2 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

$consoTest->addFidelite(2, 250);

echo "Le nombre de points de fidélité doit être égal au montant cumulé d'achat, c'est à dire 150+250 = 400 ici: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 2 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

}

public function testInitConso() {

$consoTest = new ConsoFidele("Lifo", "Paul","lifo.paul@gmail.com", "0600000000","1961-01-03","2017-01-05");

// À compléter sur votre copie

}

public function testAddMontant() {

$consoTest = new ConsoFidele("Lifo", "Paul","lifo.paul@gmail.com", "0600000000","1961-01-03","2017-01-05");

$consoTest->addFidelite(3, 50);

echo "Le nombre de points de fidélité doit être égal à 0, puisque le montant d'achat est compris entre 50 et 100: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 3 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

$consoTest->addFidelite(3, 150);

echo "Le nombre de points de fidélité doit être égal à 10, puisque le montant d'achat est compris entre 100 et 200: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 3 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

$consoTest->addFidelite(3, 250);

echo "Le nombre de points de fidélité doit être égal à 30 (20 + 10), puisque le montant d'achat est compris entre 200 et 500: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 3 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

$consoTest->addFidelite(3, 550);

echo "Le nombre de points de fidélité doit être égal à 80 (50 + 30), puisque le montant d'achat est supérieur à 500: <BR>" ;

echo "Le nombre attribué par la méthode addFidelite programme 3 est: " .$consoTest->getPointsFidelite() ."<BR><BR>";

    }

}

$test = new TestPointsFidelite ;

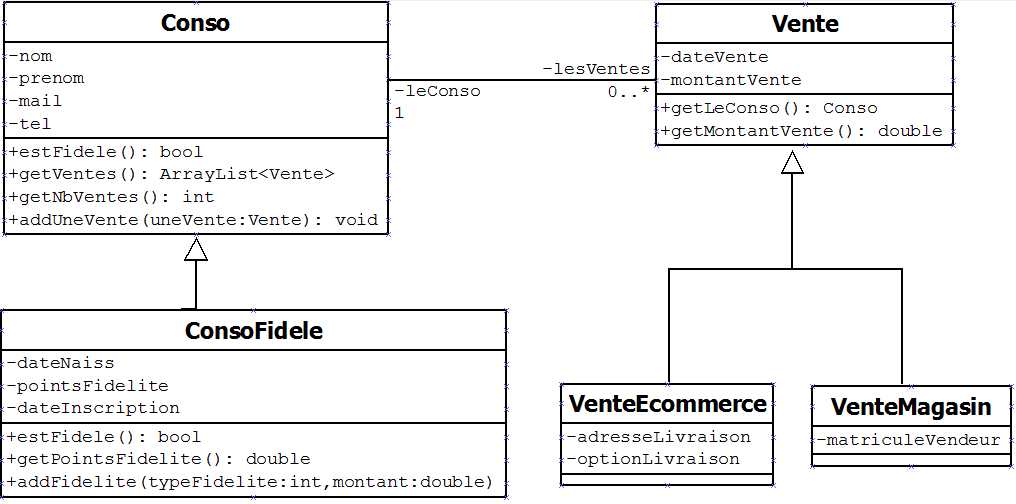
echo "Resultat de la methode de test du programme de fidélité par tampon: testAddFideliteTampon est :<BR>" ;

echo $test->testAddFideliteTampon();

echo "Resultat de la methode de test du programme de fidélité par montant achat: testAddFideliteMontant est :<BR>" ;

echo $test->testAddFideliteMontant();

# Document 9 : extrait du diagramme de classes utilisé par le module de statistiques



class Conso {

private $nom ;

private $prenom ;

private $mail ;

private $tel ;

private $lesVentes = array() ;

public function \_\_construct($unNom, $unPrenom, $unmail, $untel) {

…

}

public function estFidele() {

return false;

}

public function addUneVente($uneVente) {

$this->lesVentes[] = $uneVente;

}

public function getVentes() {

return $this->lesVentes ;

}

public function getNbVentes() {

// A COMPLETER

}

}

class ConsoFidele extends Conso {

private $dateNaissance ;

private $pointsFidelite=0 ;

private $dateInscription ;

public function \_\_construct ($unNom, $unPrenom, $unmail, $untel, $dateNaiss, $dateInsc){… }

public function estFidele() {

return true;

}

public function addFidelite($typeFidelite, $montant) {… }

public function getPointsFidelite() { … }

}

class Vente {

private $dateVente ;

private $leConso ;

private $montant ;

public function \_\_construct ($uneDateVente, $unConso, $unMontant){ … }

public function getMontantVente() { … }

public function getLeConso() { … }

}

*//* ***extends*** *met en oeuvre le mécanisme d'****héritage***

class VenteEcommerce extends Vente {

private $adresseLivraison ;

private $optionLivraison ;

public function \_\_construct … // A COMPLETER SUR VOTRE COPIE

}

class VenteMagasin extends Vente {

private $matriculeVendeur ;

public function \_\_construct // A COMPLETER SUR VOTRE COPIE

# Document 10 : extrait de la classe Statistique utilisée par le module de statistiques

class Statistique {

// méthode statVente à commenter

public function statVente($lesVentesDuJour) {

        $nbVenteFidele=0;

        foreach($lesVentesDuJour as $uneVente) {

            $conso = $uneVente->getLeConso();

            if ($conso->estFidele())

                $nbVenteFidele ++;

        }

        return $nbVenteFidele\*100/count($lesVentesDuJour);

    }

// méthode compareLieuVente

public function compareLieuVente($lesConsosFidele) {

    $totalEcom=0; //cumul des montants des ventes ecommerce

    $totalMag=0; // cumul des montants des ventes en magasin

//parcours de la liste des consommateurs fidèles

    foreach($lesConsosFidele as $unConsoFidele) {

// A Completer sur votre copie

    }

    return $totalMag/$totalEcom ; //calcul de l’indice et retour du résultat

}

}

|  |
| --- |
| **Remarque** :  **instanceof** permet de connaître le type d'instance d'un objet.  if ($uneVente **instanceof Vente**) {    ***// traitement effectué si $uneVente est instance de la classe Vente***  } |